Praktikumsaufgabe 2

Schiffe Versenken[1]

Formalia:

1. Zur Programmentwicklung **verwenden Sie als Modul-Namen (Datei-Namen) für das zweite Praktikum den Namen** `p02NameVorname.py` **und halten sich strikt an die in der Aufgabe vorgegebenen Namen für die Funktionen (Grund: Unit-Test s.u.)**.
2. Sie können mit dem in Moodle bereitgestellten Unit-Test `p02unit.py` überprüfen, ob Ihre eigene Lösung die richtigen Ergebnisse liefert:
   a) Im Modul `p02unit.py` passen Sie die 1-te Zeile gemäss Ihrem Namen an.
   b) Im Anaconda-Prompt können Sie den Unit-Test ausführen durch die Anweisung: `python p02unit.py`
3. **Erst wenn Ihre Lösung den Unit-Test besteht, können Sie Ihre Lösung in der Zoom-Sitzung (gleicher Link wie Online-Sprechstunde/Vorlesung) abnehmen lassen**.
4. Falls Sie selber Ihre Fehler im Unit-Test nicht beheben können, haben Sie in der Zoom-Sitzung zum Praktikum die Möglichkeit, die Fehler mit den anderen TeilnehmerInnen im Breakout-Raum zu besprechen und sich helfen zu lassen.
5. Falls Sie in der Breakout-Raum-Gruppe nicht weiter kommen rufen Sie Dozenten-Hilfe. Beachten Sie dass aufgrund der Anzahl der Breakout-Räume es etwas dauern kann, bis die *Dozenten-Hilfe* kommt. **Bringen Sie deshalb Geduld und Verständnis mit in die Praktikums-Zoom-Sitzung**.
6. **Die Abnahme Ihrer Praktikums-Lösung kann nur in der für Ihre Gruppe stattfindenden Zoom-Sitzung (gleicher Link wie Online-Sprechstunde/Vorlesung) erfolgen. Die Teilnahme an allen Praktikums-Zoom-Sitzungen für Ihre Gruppe ist Pflicht für den Erhalt der Bonus-Punkte.**
7. Sollten Sie in der vorgesehenen Zoom-Sitzung die Abnahme nicht schaffen haben Sie **am Ende** der nächsten Zoom-Sitzung für Ihre Gruppe die Möglichkeit eine zurückliegende Praktikums-Aufgabe abnehmen zu lassen.
8. Die Gruppeneinteilung und die Termine für die Zoom-Praktikums-Sitzungen werden über den Moodle-Kurs bekannt gegeben.

Ein Raster (*Gebiet*, `area`) bestehend aus m Zeilen und n Spalten wird mit Schiffen belegt. Um die Ausgabe des Zeilen-Indexes und des Spalten-Indexes einfach zu halten beschränken wir uns in dieser Praktikumsaufgabe auf folgende Werte für m und n:

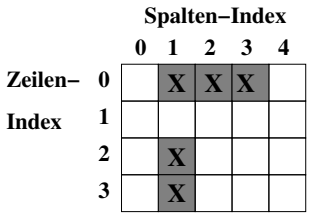
$$2 \leq m \leq 10, \qquad \text{und} \qquad 2 \leq n \leq 10 \tag{1}$$

Die Schiffe bestehen dabei aus horizontal oder vertikal nebeneinander liegenden Feldern, die durch **den Buchstaben** `X` gekennzeichnet sind. Die Anzahl der belegten Felder für ein Schiff wird auch *Länge* (`length`) des Schiffs genannt. Wie in der unteren Abbildung dargestellt seien der Einfachheit halber die Zeilen und Spalten mit 0 startend indiziert (in

[1] In Anlehnung an die Praktikums-Aufgaben der Kollegen Schöttl und Tasin aus dem SoSem 2018.

der Praxis beginnt sehr oft der Index mit 1, dies erfordert dann beim Programmieren eine Index-Verschiebung!).

Wie aus der Mathematik-Vorlesung des ersten Semesters bekannt, ist es üblich bei der Positions-Angabe von Feldern einer Tabelle (Matrix) als **ersten Index den Zeilen-Index** und als **zweiten Index den Spalten-Index** zu verwenden. Das vertikale Schiff belegt somit die Felder (2,1) und (3,1).




```
Area
 |01234|
0| XXX |
1|     |
2| X   |
3| X   |
 |01234|
```

1. Überlegen Sie sich eine geeignete Datenstruktur (im Folgenden als `area`-Objekt bezeichnet) für die Repräsentation des Gebiets und seine Schiffsbelegung (Die Ende Markierung des Gebietes und die Nummerierung der Zeilen/Spalten sollen nicht mit abgespeichert werden, sie werden nur beim Ausdruck des Gebietes zusätzlich gedruckt (s.u.)).

   Erstellen Sie eine Funktion `create_area(size)`.

   Der Übergabe-Parameter `size` ist dabei das Tupel (m, n), das die Höhe (Anzahl Zeilen m) und die Breite (Anzahl Spalten n) des Gebietes (`area`) angibt. Für das in der Abbildung dargestellte Beispiel gilt $(m, n) = (4, 5)$.

   Wenn (m, n) in dem durch die obigen Bedingungen (1) festgelegten gültigen Bereich liegen, liefert die Funktion ein leeres (d.h. ohne abgelegte Schiffe) `area`-Objekt zurück, andernfalls wird `None` zurückgegeben.

2. Erstellen Sie eine Funktion `fill_area(area, p0, is_horiz, length)`, die ein Schiff der Länge `length` in das Gebiet `area` ab Position `p0` (ein Tupel (i, j) mit Zeilen-Index i und Spalten-Index j der Start-Position des Schiffes) einträgt. Der Boolsche Parameter `is_horiz` zeigt an, ob das Schiff horizontal (nach rechts) oder vertikal (nach unten) ab Position `p0` eingetragen werden soll.

   In dieser Funktion muss **nicht** überprüft werden, ob ein Schiff überhaupt aus Platz- oder Belegungs-Gründen abgelegt werden kann. Dieser Test ist ausgelagert in eine noch zu erstellenden Funktion `check_area()` (s. u. Aufgabenteil 4)), die nach Ihrer Erstellung vor einem Aufruf von `fill_area()` immer aufzurufen ist.

   Tipp: Für ein horizontales Schiff kann man eine Slice-Zuweisung verwenden, für ein vertikales Schiff geht dies leider nicht so einfach.

3. Um die bisher erstellten Funktionen zu testen, erstelle man eine Funktion `print_area( area, title="Area" )`, die **genau wie rechts oben in der Abbildung dargestellt** die Belegung des Gebietes `area` ausgibt.

   Der optionale Parameter `title` wird erst bei der Programmierung eines vollständigen Spieles von Bedeutung sein, da jeder Spieler dann ein `area`-Objekt für seine eigenen Schiffe und ein weiteres `area`-Objekt für seine Notizen, in dem er dann Terffer (T) oder Wasser (W) eintragen kann, benötigt.

   Ein Zeichen `'X'` können Sie ohne anschließenden Zeilenvorschub durch `print("X", end="")` ausgeben.

   Da die Ausgabe der Spalten-Indizes sowohl oberhalb als auch unterhalb von dem Gebiet erfolgen soll, ist es ratsam hierfür eine eigene Funktion `print_column_numbers( n_columns )` zu erstellen.

Man teste nun z.B. mit den in der Abbildung gezeigten Schiffen die erstellten Funktionen.

Tipp:

Sollte man seltsame Effekte bemerken untersuche man mittels der Funktion `id()` die Situation im Speicher und zeichne ein Box-Pointer-Diagramm des `area`-Objektes.

Eine flache Kopie eines Objektes kann man immer mit dem Slice-Operator `objekt[:]` erzeugen oder mittels `objekt.copy()`.

4. Erstellen Sie eine Funktion

   `check_area(area, p0, is_horiz, length, is_profi=False)`,

   die überprüft, ob ein Schiff der Länge `length` in das Gebiet `area` ab Position `p0` eingetragen werden kann, ohne sich mit einem schon bestehenden Schiff zu überlappen. Im Fall eines möglichen Eintrags soll die Funktion `True` zurückliefern, andernfalls `False`.

   Tipp:

   a) Die Zeile `m, n = len(area), len(area[0])` zu Beginn der Funktion ist sinnvoll.

   b) Den optionalen Parameter `is_profi` kann man zunächst ignorieren (s. u).

   c) In den beiden Fällen `is_horiz == True/False` ist zuerst ein **Schnelltest**, der überprüft ob die Werte von `p0` und `length` zu `m` und `n` passen, sinnvoll. Z. B. kann man im Fall `is_horiz == True` sofort `False` zurückgeben, wenn

   ```
   p0[0] < 0 or p0[0] >= m or p0[1] < 0 or p0[1] + length > n
   ```

   gilt *(warum? können Sie dies mit der Hilfe einer Skizze erklären?)*.

   d) Nur wenn der Schnelltest bestanden wurde muß dann noch überprüft werden, ob alle vom Schiff beanspruchten Felder frei sind. Im horizontalen Fall kann man einfach überprüfen ob

   ```
   "X" in area[???][???:???]
   ```

   gilt, wobei für `???` passende Werte verwendet werden müssen. Der vertikale Fall ist leider nicht so einfach zu lösen.

Für Profis: *(Hier kann sich jeder angesprochen fühlen, Programmieren lernt nur durch Programmieren)*

Die offiziellen Schiffe-Versenken-Regeln verlangen ein leeres Feld Abstand zwischen zwei Schiffen. Modifizieren Sie die Funktion `check_area` so, dass sie im Fall `is_profi==True` die offiziellen Schiffe-Versenken-Regeln bei der Überprüfung anwendet.

5. Erstellen Sie eine Funktion

   `generate_boat(area, boat_spec, is_profi=False)`,

   die ein Schiff der Länge `boat_spec` im Gebiet `area` zufällig ablegt. Testen Sie die Funktion.

   Tipp:

   a) Eine gleichverteilte zufällige ganze Zahl im Intervall von a bis einschließlich b erhalten Sie mittels `zahl = random.randint(a,b)`. Hierfür muß das Modul `random` importiert werden.

   b) Sowohl die Ausrichtung des Schiffes als auch die beiden Koordinaten der Start-Position `p0` wird man mit Hilfe von Zufalls-Zahlen bestimmen.

c) Vor dem Ablegen des Schiffes durch die Verwendung der Funktion `fill_area()` muß man natürlich mit der Funktion `check_area()` überprüfen, ob das Schiff mit der zufällig ermittelten Ausrichtung und der zufällig ermittelten Start-Position abgelegt werden kann.

d) Ggf. sind also mehrere Versuche nötig um ein Schiff abzulegen. Um eine Endlos-Schleife zu vermeiden, ist es ratsam die Anzahl Versuche durch eine festgelegte obere Schranke zu begrenzen und beim Überschreiten der Schranke mit einer Fehler-Meldung abzubrechen. Mit der Anweisung `sys.exit(errorMessage)`, die das Importieren des Modules `sys` erfordert, kann das Programm abgebrochen werden. (*Bem. sauberer wäre es eine Ausnahme (Exception) zu werfen, diese Technik wird aber erst später in der Vorlesung vorgestellt)*

6. Erweitern Sie die Funktion

   ```
   generate_boat(area, boat_spec, is_profi=False)
   ```

   um die Möglichkeit mehrere Schiffe abzulegen. Hierzu kann die dynamische Typisierung von Python elegant ausgenutzt werden. `boat_spec` soll **neben einer ganzen Zahl auch ein Dictionary** enthalten dürfen, das die Länge und die Anzahl der gewünschten Schiffe als Schlüssel und Inhalt (Wert) des Dictionaries angibt.

   So soll z. B. der Aufruf `generate_boat(area, {2:2, 3:1}, False)` zwei Schiffe der Länge 2 und ein Schiff der Länge 3 ablegen. Testen Sie die Funktion.

   Tipp:

   a) Es ist ratsam zuerst den Typ von `boat_spec` abzufragen.

   b) Im Fall eines Dictionaries kann man mit der Hilfe der Schleife

      ```
      for (laenge, anz) in boat_spec.items():
      ```

      durch die Einträge des Dictionaries laufen. In einer weiteren Schleife über die Anzahl `anz` wird man dann die Möglichkeit ausnutzen auch die Funktion `generate_boat()` mit einem Integer-Wert `laenge` als `boat_spec` aufzurufen.

7. **Für Profis:**

   Erweitern Sie das Programm um die Funktionalität Schiffe versenken gegen den Computer zu spielen.