Schiffeversenken in Python - SS2021

Marc Ullmann, David Ruschmaritsch, Anne Lotte Müller-Kühlkamp

Frankfurt University of Applied Sciences
(1971-2021: Fachhochschule Frankfurt am Main)
Nibelungenplatz 1
60318 Frankfurt am Main
marc.ullmann@stud.fra-uas.de, david.ruschmaritsch@stud.fra-uas.de,
anne.mueller-kuehlkamp@stud.fra-uas.de

Zusammenfassung Dieses Dokument behandelt die Problemstellung der Programmierung von Schiffeversenken in Python. Dabei geht es auf den Prozess der Erarbeitung und Lösung der aufkommenden Probleme ein und gibt den Benutzer einige Informationen über den Ablauf des Programms und mögliche Anpassungen.

Das Spiel Schiffeversenken, welches früher nur mit Stift und Papier gespielt wurde und sich seit dem in seiner Grundstruktur kaum verändert hat, ist ein interessantes Problem um auf den Computer übersetzt zu werden. Doch trotz des simplen Konzepts, stellen sich einige Hindernisse in den Weg, die unterschiedlich komplex überwindbar sind. Vom Aufbau bis zum Ablauf und der visuellen Darstellung des Spiels, die durch auftretende Probleme geprägt sind, werden wir im Folgendem behandeln.

1 Grundprinzip Schiffeversenken

Zwei Spieler legen in einem Koordinatensystem Schiffe fest, die sie danach durch abwechselndes Raten der Koordinaten eliminieren sollen. Wenn alle Schiffe eines Spielers getroffen sind, so hat dieser verloren. Für die Implementierung des Spiels wird ein Spielfeld benötigt, welches mit Kommandozeilenargumenten anpassbar sein soll. Die Beschränkung von einer Mindestgröße von 10x10 Feldern und Maximalgröße von 20x20 Feldern erwies sich als sinnvoll, da bei mehr als 20 Feldern die Größe des Standard-Fenster überschritten wird und somit das Programm abstürzt. Bei weniger als zehn Feldern könnte es zu Platzproblemen der Schiffe mit dem zufälligen Platzier-Algorithmus geben und somit unerwünschte Ereignisse hervorrufen. Deshalb wird bei Eingabe von zwei Kommandozeilenargumenten die Begrenzung abgefragt, die bei Abweichung die Standardgröße von 10x10 wählt. Für die Implementierung von Schiffen ist nun die Grundlage des Spielfelds gegeben. Die Schiffanzahl ist von Grund auf festgelegt auf insgesamt zehn Schiffe [1]:

- 1x Schlachtschiff mit Größe 5
- 2x Kreuzer mit Größe 4
- 3x Zerstörer mit Größe 3
- 4x U-Boote mit Größe 2

Nun sind die Methoden zu definieren, die das Spielen ermöglichen.

2 Visuelle Darstellung

Zu Beginn kam es zur Wahl der Bibliothek für die visuelle Darstellung. Zuerst fiel diese auf termbox [2]. Da diese jedoch nicht mehr unterstützt und weiterentwickelt wird, fiel die Entscheidung zuerst auf cursebox [3], für welche die einfachen Befehle und praktische Funktionen sprachen. Da auch diese nicht weiterhin unterstützt wird und wenig Dokumentation zu der Bibliothek vorliegt, wurde das Programm in curses [4] übersetzt.

2.1 Spielfeld

Das Spielfeld ist mit Hilfe der Erweiterung numpy [5] entwickelt. Die Erweiterung bietet eine einfache Erstellung mehrdimensionaler Arrays und liefert verschiedene Operationen um diese zu konfigurieren. Ein zweidimensionales Array bietet die optimale Grundlage für das Spielfeld, da dies wie ein 2D-Koordinatensystem zu betrachten ist. Mit einem Null-Array (numpy.zeroes) ist es nur möglich Zahlen zu verarbeiten. Dies hat zur Auswirkung, dass Felder die beispielsweise mit Schiffen belegt sind, nur durch eine „1“ und Felder ohne Schiffe durch eine „0“ darstellbar sind. Dadurch ist es einfacher Felder zu vergleichen, da diese nur mit Zahlen belegt sind und nicht weiter übersetzt werden müssen. Für die grafische Darstellung des Null-Arrays bedeutet dies jedoch, eine Einschränkung um die Objekte ansehnlich darzustellen. Das Chararray, welches Zeichen speichern kann, bietet sich hier an. Jedoch wird bei der Darstellung des Arrays immer ein „b“ für Byte hinzugefügt, da es Bytestrings sind, die in dem Array gespeichert werden. Dies umgeht das Programm, indem der Datentyp des Arrays als Zeichenkette übersetzt und dann ausgegeben wird. Für die Darstellung eines Spielfeldes und nicht der Hinternanderreihung einzelner Objekte des Arrays, gibt es jede Reihe des Arrays einzeln, in einer eigenen Zeile, aus.

```
1 def create_matchfield(ySize, xSize, game_y_pos, game_x_pos,
    player, screen):
2     ''' Creates matchfields '''
3     matchfield_visual = np.chararray((ySize, xSize))
4     matchfield_visual_2 = np.chararray((ySize, xSize))
5     matchfield_visual[:] = "0"
6     matchfield_temp = np.zeros((ySize, xSize))
```

```
7     matchfield_logic = np.zeros((ySize, xSize))
8     matchfield_ship_pos = np.zeros((ySize, xSize))
```

Listing 1.1. Funktion erstellt zweidimensionale Arrays

Um die Vorteile beider Arrays zu benutzen, müssen verschiedene Arrays erstellt werden. Insgesamt gibt es fünf Arrays: Zwei Charrays für die grafische Darstellung und drei Null-Arrays für die logische Verarbeitung.

2.2 Spielfeld: Schiffplatzierung

In diesem Fall benutzt das Programm nur das eine Charray, welches die ersten beiden logischen Arrays der Schiffplatzierung visuell übersetzt. Dafür gibt ein „*“ (logisch=2) die aktuelle Position des Schiffs und ein „X“ (logisch=1) die platzierte Schiffe an. Die leeren Felder werden durch „O“ (logisch=0) dargestellt.

2.3 Spielfeld: Beschuss

Nach der Platzierung der Schiffe feuert man nun abwechselnd. Dafür sind nun durchgehend zwei Spielfelder (durch die beiden Chararrays) angezeigt. Das linke zeigt dabei das Spielfeld des Gegners an, auf welchem die aktuelle Position und schon beschossene Felder angezeigt sind. Dafür gelten die bereits Getroffenen als „*“ (logisch=2), Wassertreffer als „Prozentzeichen“ (logisch=3), das Aktuelle als „X“ (logisch=1) und Leere als „O“ (logisch=0). Das rechte Spieldfeld zeigt das eigene Spielfeld, mit allen Schiffpositionen und Schüssen des Gegners, an. An der oberen Seite wird der aktuelle Spieler und dessen verbleibenden Schiffe angezeigt, an der unteren die des Gegners.

2.4 Sonstige visuelle Elemente

Um die Bedienung zu erleichtern, zeigt es am unteren Ende die aktuell möglichen Eingaben an.

2.5 Die Funktion *update_matchfield()*

Diese Funktion ist für den großteil der visuellen Darstellung verantwortlich. Ihre Aufgabe ist es, den aktuellen Spieler anzuzeigen, die logischen in die visuellen Arrays zu übersetzen und einige visuelle Elemente hinzuzufügen. Der aktuelle Spieler wird der Funktion als Parameter übergeben und dann ausgegeben.

Um ein Spielfeld zu übersetzen, übergibt das Programm der Funktion ein Charray (visuelles Spielfeld) und ein Null-Array (logisches Spielfeld). Um zwei Spielfelder darzustellen, ruft das Programm die Funktion zwei mal auf. Dies stellt das Problem dar, dass die komplette Löschung des aktuellen Bildschirm nicht möglich ist, sondern aktuelle Elemente nur überschreibar sind. Das heißt alle Elemente die nicht genau übereinanderliegen, sind durch einen leeren String zu ersetzen. Da das Spielfeld des Gegners und des aktuellen Spielers immer an der gleichen Position liegen, überschreiben diese sich automatisch. Andere Elemente, wie die Anzahl der Schiffe und die Anzeige der Eingaben, sind in eigenen Funktionen definiert.

Mit zwei for-Schleifen wird das logische Spielfeld Schritt für Schritt in das Visuelle übersetzt. Zudem gibt es eine Änderung wenn man gegen den Computer spielt. Da das Feld von diesem, auch für den menschlichen Spieler sichtbar ist, werden seine Schiffspositionen verdeckt. Dies ermöglicht dem Spieler direkt zu sehen, worauf der Computer schießt. An den Rändern der Spielfelder grenzen Sonderzeichen das Spielfeld ab, wodurch der Überblick einfacher ist. Zudem sind die Seiten für die Texteingabe durchgehend nummeriert.

3 Mausimplementation

Grundsätzlich soll das Spiel mit der Tastatur spielbar sein, jedoch gab es in den Anforderungen auch die Wahl eine Maussteuerung einzubinden. Die benutzte Programmbibliothek „curses“, welche zur Darstellung zeichenorientierter Benutzerschnittstellen unabhängig vom darstellenden Textterminal ist, bietet hierzu die Funktion „getMouse()“ an. Diese Funktion liefert ein 5-stelliges Tupel. Darin enthalten sind unter Anderem die x- und y-Koordinaten der Maus, welche mit einem Tastenereignis (hier ein Mausklick) abzufragen sind. Nun gilt es die in dem curses-Terminal dargestellten Texte, welche als Knöpfe dienen sollen, per Mausklick zu aktivieren. Dazu müssen die Koordinaten der Maus mit denen der dargestellten Texte übereinstimmen. Dies erwies sich als umständlich und komplex für die weitere Implementierung des Spiels, da die dargestellten Strings keine Objekte sind und damit keine Funktionen zur Abfrage und dem damit verbundenen Vergleich der Mausposition hatten und man diese aufwendig herausfinden müsste. Deshalb funktioniert die Maus nur im Anfangsmenü.

4 Benutzereingaben

Für die Entgegennahme der Eingaben des Benutzers exisitiert die Funktion „userinput()“, welche die gedrückte Taste (oder Maus) entgegen nimmt. Da alle Spielwichtigenfunktionen, also jene die immer wieder aufgerufen werden, sowohl WASD als auch die Pfeiltasten für die Steuerung entgegen nehmen, erleichtert

diese Funktion die Entgegennahme dieser. Indem die jeweiligen Richtungen von WASD und den Pfeiltasten auf eine Eingabe gesetzt sind, muss man dies in anderen Teilen des Programms nur einmal abfragen. Somit spart es einiges an Tipparbeit. Kommt eine andere Eingabe, wie etwa für die Texteingaben der Koordinaten, nimmt die Funktion diese und gibt sie roh wieder zurück.

```
1  def userinput(screen):
2  ''' Checks user input '''
3  input_key = ""
4  curinput = ""
5  screen.keypad(1)
6  curses.mousemask(-1)
7
8  curinput = screen.get_wch()
9
10 if curinput == 'd' or curinput == curses.KEY_RIGHT:
11     input_key = "right"
12 elif curinput == 's' or curinput == curses.KEY_DOWN:
13     input_key = "down"
14 elif curinput == 'a' or curinput == curses.KEY_LEFT:
15     input_key = "left"
16 elif curinput == 'w' or curinput == curses.KEY_UP:
17     input_key = "up"
18 elif curinput == '\n':
19     input_key = "enter"
20 elif curinput == curses.KEY_MOUSE:
21     input_key = "mouse"
22 else:
23     input_key = curinput
24 return input_key
```

Listing 1.2. Funktion userinput() nimmt Eingaben des Benutzers an

5 Die Klasse Ship

Alle Schiffe sind als Objekte definiert. Objekte bieten hier einen erheblichen Vorteil durch deren Attribute. In jedem Schiff werden die Größe, Rotation, Startposition und Koordinaten gespeichert. Die Koordinaten berechnen sich durch die Objektmethode „ship_cords“, welche je nach Ausrichtung (horizontal, vertikal) eine Schleife bis zur Größe des Schiffs durchläuft und von der Startkoordinate aus in die jeweilige Richtung läuft.

6 Schiffplatzierung

Das erste logische Array speichert die temporären Daten, welche für die Berechnungen von der Spielfeldbegrenzung dienen. Das zweite speichert tempöräre

Daten, wenn diese die Regeln nicht verletzen. Für die Platzierung der Schiffe überprüft die Funktion in dem tempörären Array, ob die Schiffsgröße die Spielfeldgröße überschreitet. Sind platzierte Schiffe bereits vorhanden, gleicht es zudem mit dem zweiten logischen Array die Position dieser ab und berechnet so die Möglichkeit der Platzierung. Bei der Platzierung reserviert die Funktion die umliegenden Felder, da die Schiffe sich nicht direkt berühren dürfen. Das dritte logische Array speichert hingegen nur die Schiffspositionen ab, um diese später zu übergeben.

6.1 Die Funktion *set_ships_comp()*

Um gegen den Computer spielen zu können, braucht es eine Funktion, die Schiffe mit einem zufälligen Algorithmus platziert. Zu Beginn fügt es alle Schiffe einer Liste hinzu. Diese Liste wird durchlaufen und dabei jedes Schiff zufällig platziert. Für jedes Schiff generiert es die Ausrichtung (horizontal, vertikal) zufällig und anschließend eine zufällige x oder y Koordinate gewählt. Die jeweilige Zeile, beziehungsweise Spalte, prüft die Funktion auf genügend Platz für das jeweilige Schiff. Ist kein Platz vorhanden, erhöht sich die andere Koordinate. Prinzipiell sucht es so schrittweise Platz für das nächste Schiff. Sobald Das Schiff im Rahmen des Spielfelds und der anderen Schiffe Platz hat, fügt die Funktion die Felder zum logischen Array hinzu und reserviert umliegende Plätze, da Schiffe nicht nebeneinander platzierbar sein dürfen. Das Programm versucht dies solange, bis ein Counter erreicht ist, wonach sich die Rotation ändert.

Dieser Implementierung liegt das Problem der Laufzeit zugrunde. Zuerst wählte diese Funktion beide Koordinaten und die Rotation jeden Versuch zufällig neu. Mit einer geringen Anzahl an Schiffen war dies auch möglich, jedoch lief es desto mehr Schiffe platziert waren, gegen Chance von eins zu mehreren Millionen einen Platz zu finden. Die Wahl von nur einer Zufallskoordiante, welche konstant bleibt, bis die komplette Reihe durchgelaufen ist, verhinderte dies.

7 Felder beschießen

Nachdem der Spieler die Schiffe platziert hat, beginnt das eigentliche Spiel, nämlich der Beschuss des Gegners. Der Spieler kann mit WASD, Pfeiltasten oder durch Texteingabe (e) entscheiden, welches Feld beschossen werden soll. Dies bestätigt die Taste „Enter“. Mit dem Tastendruck überpüft die Funktion, ob dieses Feld bereits beschossen wurde. Dies gilt nicht für den ersten Schuss, da kein Feld bisher beschossen sein kann. Trifft der Schuss kein Schiff aber ein gültiges Feld, belegt die Funktion das Feld im logischen Array mit einer „3“. Trifft der Schuss jedoch ein Schiff, identifiziert es die Koordinate des getroffenen Schiffs und löscht das getroffene Feld aus den Koordinaten des Schiffs. Sobald alle Felder des Schiffs getroffen sind, löscht es das Schiffobjekt aus der ship_list_placed

Liste. Ist ein Schiff zersört belegt es die naheliegenden Felder mit leer, da sich dort keine Schiffe mehr befinden können.

7.1 Die künstliche Intelligenz für das Beschießen der Felder

Damit der Computer die Möglichkeit hat Felder zu beschießen, braucht er eine Funktion um Felder zufällig beschießen zu können. Wichtig dabei ist es, dass wenn ein Schiff getroffen wurde, der Computer solange umliegende Felder beschießt bis dieses Schiff zerstört ist. Der erste Schuss oder nachdem ein Schiff zerstört wurde, erfolgt durch eine zufällige Position auf dem Spielfeld. Trifft dieser nicht, werden neue zufällige Koordinaten ausgewählt. Andernfalls wird dies abgespeichert und nun die umliegenden Felder beschossen. Dafür wird überprüft, ob diese außerhalb des Spielfelds liegen oder ob dieses Feld schon beschossen ist. Es schießt dabei solange in eine Richtung, bis ein leeres Feld getroffen wird, wonach sich die Richtung ändert. Ein Problem stellte sich, wenn der erste Schuss in der Mitte des Schiffes trifft und nun erst in eine Richtung und dann in die andere, über die schon beschossenen Felder, wieder gehen muss. Um dies zu lösen, unterscheidet man zwischen leeren und schon beschossenen Feldern. Schon beschossene Felder können nur von dem aktuell beschossenen Schiff sein, da zwischen jedem platzierten Schiff mindestens ein Leerfeld liegen muss. Sieht die Funktion also, dass ein Feld schon beschossen wurde überspringt es dieses solange, bis sie ein noch nicht getroffenes Feld gefunden hat. Die Funktion unterscheid zudem auch, ob erst ein Feld oder zwei hintereinader getroffen wurden. Wurden zwei hintereinader getroffen, behält die Funktion die aktuelle Richtung bei.

8 Benutzeranpassungen

Das Spiel kann durch veränderungen des Quellcodes aufgerufen werden. Folgend möglich Anpassungen:

- Beim Aufruf des Programms kann mit zwei Kommandozeilenargumentenargumenten die Größe des Spielfelds angepasst werden. Die Zahlen sind dabei zwischen 10-20 wählbar. Um dies zu zu ändern kann jeweils für den Mehrspielermodus und den Einzelspielermodus die minimum und maximum Eingabe verändert werden (Zeile 3 für y-Größe und Zeile 6 für x-Größe). Auch die Standardgröße kann verändert werden (Zeile 10,11, bzw. 5,8)

```
1 if len(sys.argv) == 3:
2 # Takes in system arguments for the game size; if it exceeds
    20 it sets it to twenty (due to the window size)
3     if int(sys.argv[1]) >= 10 and int(sys.argv[1]) <= 20:
4         yGameSize = int(sys.argv[1])
```

```
5       else: yGameSize = 10
6       if int(sys.argv[2]) >= 10 and int(sys.argv[1]) <= 20:
7           xGameSize = int(sys.argv[2])
8       else: xGameSize = 10
9   else:
10      xGameSize = 10
11      yGameSize = 10
```

Listing 1.3. Annahme der Kommandozeilenargumente

– Die visuelle Darstellung des Spiels kann in der Funktion *update_matchfield()* angepasst werden. Dafür das Symbol zwischen den Anführungszeichen ändern.
  - Zeile 5: Position der Schiffe des Computers
  - Zeile 7: Position des eigenes Schiffeversenken
  - Zeile 9: Treffer
  - Zeile 11: Fehlschuss
  - Zeile 13: Fadenkreuz
  - Zeile 17: Leeres Feld des Computers
  - Zeile 23: Leeres Feld

```
1  if matchfield_temp[y,x] == 1:
2  # Places a x if temporal matchfield has a 1
3  if player == "comp":
4      if mode == "secondary":
5          matchfield_visual[y,x] = "?"
6  else:
7      matchfield_visual[y,x] = "X"
8  elif matchfield_temp[y,x] == 2:
9  matchfield_visual[y,x] = "*"
10 elif matchfield_temp[y,x] == 3:
11 matchfield_visual[y,x] = "%"
12 elif matchfield_temp[y,x] == 4:
13 matchfield_visual[y,x] = "X"
14 else:
15 if player == "comp":
16     if mode == "secondary":
17         matchfield_visual[y,x] = "?"
18     if mode == "third":
19         pass
20 elif mode == "third":
21     pass
22 else:
23     matchfield_visual[y,x] = "O"
```

Listing 1.4. Anpassung der visuellen Darstellung

- Die Eingabemöglichkeiten können in der Funktion *userinput()* verändert werden. Dafür entweder die vorgebenen Tasten ersetzen oder mit einer „or“-Abfrage, der jeweiligen Richtung, hinzufügen
- Die Anzahl der Schiffe und deren Größen können in der Funktion *set_ships_comp()* (für den Computer) und *set_ships()* (für die Spieler) verändert werden. Dafür am Anfang der Funktionen Schiffe löschen, beziehungsweise hinzufügen. Diese müssen jeweils auch in der „ship_list“ hinzugefügt oder gelöscht werden. Um ein neues Schiff zu erstellen, muss eine neue Zeile eingefügt werden „'Schiffsname' = Ship.Ship('Größe des Schiffs')“
- Die Namen der Schiffe können in der Funktion *current_ships()* angepasst werden. Dafür die Schiffsnamen unten bei „ships_own_string“ und „ships_enemy_string“ ändern.

9 Fazit

Die strukturelle Vorgehensweise, Problem nach Problem abzuarbeiten, war sehr hilfreich für die Erstellung des Programms. Vom Startbildschirm ausgehend, kamen die ersten Probleme, welche durch eine Aufteilung in kleinere Probleme nach und nach gelöst wurden. Die Platzierung der Schiffe stellte einige Probleme da, wie die Laufzeit, welche beachtet werden musste und die generelle Logik dahinter. Die Reduzierung von komplett zufälligen Prozessen, zu kleineren Teilen und die Erhöhung von logischen Abläufen, wie das systematische Durchlaufen der Zeilen und den Nachfolgetreffern bei dem zufälligen Beschuss, half dieses Problem zu bewältigen. Es ergab sich, dass in komplexeren Prozessen Zufallsabläufe zu Zeitintensiv sind und durch logische Abläufe ersetzt werden müssen. Zudem ergab sich die Problematik mit der Maus, welche schwer in der Shell, über die eingesetzte Libary, einzubinden ist. In zukünftigen Projekten ist dies ein Problem, welches weiter erforscht werden kann.

Abschließend kann man sagen, dass das Projekt sehr spaßig aber auch gleichzeitig herausfordernd war. Besonders herausgestochen hat die Implementierung des Computers, da die Arbeit mit Zufallsfunktionen diverse Probleme hervorgerufen hat. Zudem war die Zurechtfindung mit der neuen Umgebung (Python, curses) zu Beginn schwierig, da nicht genügend Vorkenntnisse vorlagen. Das Hereinarbeiten in die neue Umgebung und das dadurch folgende Erreichen von gesetzten Zielen, erbrachte aber Spaß.

Literatur

1. Schiffeversenken. wikipedia.org. 2021
   https://github.com/nsf/termbox
2. nsf. Termbox. Github. 2020
   https://github.com/nsf/termbox

3. Tenchi2xh. Cursebox. github.com. 2018
https://github.com/Tenchi2xh/cursebox
4. A.M. Kuchling, Eric S. Raymond, Curses, python.org. 2021
https://docs.python.org/3/howto/curses.html
5. Numpy, numpy.org. 2021
https://numpy.org/