



# Schiffe Versenken

Siawash Shojai  
Danial Farshchi Andizi  
Wojciech Czaczyk  
Mohammed Sanli  
Amir Ramezankhani

| Betriebssysteme und Rechnernetze | 21.06.2022

# Inhaltsverzeichnis

1. Aufgabenbeschreibung
2. Untersuchung
3. Konzept
4. Realisierung (Problemlösung & Implementierung)

## 1. Aufgabenbeschreibung

Entwicklung eines Schiffe-Versenken-Spiels mit Interprozesskommunikation, in dem zwei Spieler in der Shell gegeneinander spielen können. Durch die Möglichkeit, mit der Interprozesskommunikation, sollen bestimmte individuelle Anforderungen wiedergegeben werden, wie die Anzahl der Felder, Anzahl der Schiffe pro Spieler und deren Größen und Namen, Wiedergabe der Ereignisse (Sieg/Niederlage, Treffer/ deplatziert), und Spielstand genauer definiert, die Anzahl der versenkten Schiffe.

## 2. Untersuchung

Als Erstes wurde überlegt, welche Programmiersprache verwendet werden soll. Wir haben uns schlussendlich für Python entschieden, weil es bestimmte Vorteile mit sich bringt, wie die Ähnlichkeit zu Java in der Objekt orientierten Programmierung, welche denn Anforderungen passend entgegenkommen, noch dazu bietet es weitere Vorteile wie: Es bietet für fast alle Anwendungsprobleme eine Lösung bzw. man lernt eine Sprache für alle Gebiete, Python-Programme sind kürzer und transparenter, und es ist objektorientiert.

Um das Spiel nach den angegebenen Anforderungen zu programmieren, wurde sich das gesamte Spielkonzept genauer angeschaut und in einzelne Bestandteile eingeordnet wie (Regeln, Punktevergabe, Schiffe, Spielfeld) und etc., um sich so einen Überblick zu verschaffen und ein besseres Verständnis zum Spielaufbau zu vermitteln. Somit hatten wir zu jedem Bestandteil Notizen gemacht, um einen gewissen Ablauf beim Programmieren zu haben, weil wir die eingeteilten Kategorien dem Spielaufbau und der Benutzereingabe eingeordnet hatten wir schon einen ersten Meilenstein zu programmieren. Zudem wurde noch überlegt, in welcher einsteigerfreundlichen Weise die Interprozesskommunikation das Spiel an sich und den Ablauf wiedergibt. Infolgedessen wurden weitere Gedanken über zusätzliche Modifikationen und Funktionen gemacht.

### 3. Konzept

Vom Konzept her waren wir eher daraus ausgelegt ein etwas einsteigerfreundliches Spiel zu entwickeln, was daraus angelegt ist, dass auch Anfänger das Spiel schnell verstehen und auch spielen können. Um das wiederzugeben haben wir beschlossen viele Anweisungen und Schritte als Kommentar wiederzugeben. Eine weitere Idee war es von uns, dass wir eine gute Spielansicht anzubieten, um einen Überblick zu erhalten, dazu gehört auch ein verständliches Spielfeld mit verschiedenen Symbolen. Das Konzept wäre aufgestellt, aber das eigentliche Problem hierbei waren die Python Kenntnisse, weil einige nur grundlegende Kenntnisse in Python hatten und andere keine. Jedoch konnten wir unsere Kenntnisse auffrischen und verbessern durch das Buch „Fit fürs Studium Informatik“ und durch eine YouTube Python Lern Playlist.

### 4. Realisierung 1.0

```
1
2 def pruefen(schiff, besetzt):
3     schiff.sort() # .sort() sortiert alle Elemente in der Liste
4     for i in range(len(schiff)):
5         zahl = schiff[i]
6
7         if zahl in besetzt:
8             schiff = [-1] # [-1] bedeutet index von hinten
9             break
10
11        elif zahl < 0 or zahl > 99:
12            schiff = [-1]
13            break
14
15    return schiff
```

Zu unserem Projekt hatten wir uns die verschiedenen Bereiche des Spiels in Unterkategorien eingeteilt, damit wir sie dementsprechend in unserem Code passend einteilen können. Angefangen mit der Überkreuzung der Schiffe, in der Zeile von 1 bis 15 befindet sich die Methode, wo in Zeile 4 eine „for“ Schleife verhindern soll, dass sich die Schiffe [] aus der Liste überkreuzen bzw. sich außerhalb des Spielfelds von 0 bis 99 aufhalten, welche in Zeile 11 als „**elif zahl < 0 or zahl > 99:** „. Einfach formuliert dient es zur Überprüfung, als eine Art Prüfmethode und diese arbeitet mit der nächsten Methode zusammen, welche zur Koordinatenauswahl dient. Wir hatten bei dieser Methode eine Problematik, dass es Schwierigkeiten mit der Verknüpfung gab. Da diese Methode nicht die Koordinaten übernehmen konnte, nach Vielen versuchen haben wir anhand eins Python Snake Spiels von „YouTube“ mitbekommen, dass manche Programme effizienter funktionieren, wenn man die Bedingung für eine Methode, vor der Benutzereingabe schon festlegt, welche bei uns auch funktioniert hat.

## 4. Realisierung 1.1

```
18 def get_schiff(laenge, besetzt):
19     ok = True
20     while ok:
21         schiff = []
22
23         # der Spieler verlegt hier seine Schiffe
24         print("Schiff mit länge", laenge)
25         for i in range(laenge):
26             schiff_num = input("Bitte geben Sie die Koordinaten ein. Erst die Zeile, dann die Spalte: ")
27             schiff.append(int(schiff_num))
28
29         # Überprüfung der Schiffe
30         schiff = pruefen(schiff, besetzt)
31
32         # Die Felder werden besetzt
33         if schiff[0] != -1:
34             besetzt = besetzt + schiff
35             break
36
37         else:
38             print("Hier befindet sich schon ein anderes Schiff")
39
40     return schiff, besetzt
```

Wie in „Realisierung 1.0“ schon erwähnt muss die Methode, die verhindert, dass sich Schiffe außerhalb des Spielfeldes befinden, mit der Methode „def get\_schiff“ welche zur Einsetzung von den Spielschiffen auf das jeweilige Spielfeld dient, verknüpfen. Da diese Problematik schon behoben wurde, haben wir in Zeile 25 mit der „for“ Schleife welche die Schiff [] Liste enthält, die Möglichkeit programmiert, dass der Spieler seine Schiffe beliebig auf das Spielfeld platzieren kann. Zeile 30 überprüft, ob die Schiffe besetzt sind. Zuletzt wird von Zeile 33 bis 40 überprüft, ob die Felder schon von einem anderen Schiff besetzt wird. Falls dies der Fall sein sollte, wird mit der „Break“ Option nochmal darauf hingewiesen, wie in Zeile 38 zu sehen, dass sich hier schon ein anderes Schiff befindet. Bei dieser Methode ist uns die Problematik aufgefallen als das Programm nicht richtig ausgeführt wurde, bzw. wir hatten zufällig gewählte Zahlen genommen, und diese waren jedoch nicht erkennbar. Somit hatten wir die Idee, genauer gesagt das Prinzip eines Fußballspielfeldes zu übernehmen. Ein Fußballfeld hat bestimmte Eingrenzungen, demnach haben wir unser Spielfeld auf ein 10 x 10 Feld beschränkt mit der Zahlenfolge von 0 bis 9 weil man bei Zahlenfolgen in Programmen bei 0 anfängt zu zählen.

## 4. Realisierung 1.2

```
43 def schiffe_erstellen(besetzt, boete):
44     schiffe_list = []
45
46     for boot in boete:
47         ship, besetzt = get_schiff(boot, besetzt)
48         schiffe_list.append(ship)
49
50     return schiffe_list, besetzt
```

Die Schiffe werden durch die List [] erstellt.

```
53 def zeig_spiel_feld_schiffe(besetzt):
54     print("      Schiffe Versenken ")
55     print("    0 1 2 3 4 5 6 7 8 9")
56
57     platz = 0
58     for x in range(10):
59         zeile = ""
60         for y in range(10):
61             zeichen = " _ "
62             if platz in besetzt:
63                 zeichen = " o "
64             zeile = zeile + zeichen
65             platz = platz + 1
66
67     print(x, " ", zeile)
```

Hier wurde eine Art Übersichtsspielfeld erstellt, um nochmal einen Einblick zu erhalten, wo die eigenen Schiffe platziert sind. Da wir Wert auf Einsteigerfreundlichkeit legen. Um das zu erstellen hatten wir die Hilfestellung von einem Forum namens „Python-lernen.de“, weil wir Probleme hatten einen visuelle Variante zu erstellen. Wir hatten anfangs überlegt, ob wir das Programm in eine Art Textform wiedergeben sollen, jedoch schien das eher weniger Einsteigerfreundlich zu wirken, und durch die visuelle Darstellung von Symbolen sollte alles schnell erkennbar sein.

```
70 def zeig_spiel_feld(treffer, verfehlt, versenkt):
71     print("      Schiffe Versenken ")
72     print("    0 1 2 3 4 5 6 7 8 9")
73
74     platz = 0
75     for x in range(10):
76         zeile = ""
77         for y in range(10):
78             zeichen = " _ "
79             if platz in treffer:
80                 zeichen = " o "
81             elif platz in verfehlt:
82                 zeichen = " x "
83             elif platz in versenkt:
84                 zeichen = " 0 "
85             zeile = zeile + zeichen
86             platz = platz + 1
87
88     print(x, " ", zeile)
```

Das hier ist ein ergänztes Übersichtsfeld, aber so angepasst, dass es als Spielfeld dient. Es wurde so angepasst das Spieler 1 ein eigenes Spielfeld besitzt mit eigener Übersicht und Spieler 2 ebenso. Hier war der Vorteil, dass wir unser eigentliches Übersichtsfeld zum eigentlichen „Guessboard“ programmiert hatten.

```

91 def check_shot(schuss, schiffe, treffer, verfehlt, versenkt):
92     missed = 0
93     for i in range(len(schiffe)):
94         if schuss in schiffe[i]:
95             schiffe[i].remove(schuss)
96             if len(schiffe[i]) > 0:
97                 treffer.append(schuss)
98                 missed = 1
99             else:
100                 versenkt.append(schuss)
101                 missed = 2
102     if missed == 0:
103         verfehlt.append(schuss)
104
105     return schiffe, treffer, verfehlt, versenkt, missed

```

Hier wurde wieder eine Überprüfungsmethode erstellt, mit if & else anweisungen, um sozusagen die Trefferanzahl zu visualisieren.

```

108 def get_schuss(versuch):
109     ok = "n"
110     while ok == "n":
111         try:
112             schuss = input("Geben Sie die Koordinaten ein")
113             schuss = int(schuss)
114             if schuss < 0 or schuss > 99:
115                 print("Die Koordinaten befinden sich nicht auf dem Spielfeld. Versuchen Sie nochmal")
116             elif schuss in versuch:
117                 print("Falsche Koordinaten. Sie wurden schon eingegeben. Versuchen Sie nochmal")
118             else:
119                 ok = "y"
120                 break
121         except:
122             print("Geben Sie bitte nur die Koordinaten ein, die existieren")
123
124     return schuss

```

In dieser Methode wird der Spieler darauf hingewiesen, Koordinaten für einen Angriff einzugeben. Der Input bzw. die Eingabe wird mit der „if, elif, else“ Anweisung in Zeile 114 aus Zeile 112 überprüft und mit der „return“ Eingabe wiedergegeben, je nach variierte Eingabe. Diese Methode weist viel Ähnlichkeit mit Java Zuweisungen auf, daher hatten wir hier keinerlei Problematik.

```

125
126 # Überprüft, ob es noch übrige Schiffe gibt
127 def pruefen_ob_leer(alle_schiffe):
128     return all([not elem for elem in alle_schiffe])
129
130
131 # vor dem Spielbeginn
132 treffer1 = []
133 verfehlt1 = []
134 versenkt1 = []

```

In Zeile 126 bis 128 wurde eine kleine Prüfmethode programmiert. Durch den „return“ Wert wird einem angezeigt, falls Schiffe übriggeblieben sind.

## 4. Realisierung 1.3

```
131 # vor dem Spielbeginn
132 treffer1 = []
133 verfehlt1 = []
134 versenkt1 = []
135 versuch1 = []
136 missed1 = 0
137 besetzt1 = []
138
139 treffer2 = []
140 verfehlt2 = []
141 versenkt2 = []
142 versuch2 = []
143 missed2 = 0
144 besetzt2 = []
145
146 # Anzahl und gröÙe der Schiffe
147 schiffe = [4, 3, 2, 1]
148
149 # Spielfeld für Spieler 1
150 schiffe1, besetzt1 = schiffe_erstellen(besetzt1, schiffe)
151 zeig_spiel_feld_schiffe(besetzt1)
152
153 # Spielfeld für Spieler 2
154 schiffe2, besetzt2 = schiffe_erstellen(besetzt2, schiffe)
155 zeig_spiel_feld_schiffe(besetzt2)
156
157 # Wenn man z.B.: bis Runde 71 spielt, beendet das Programm alles
158 for i in range(70):
```

Dieser Abschnitt beschäftigt sich sozusagen mit der Wiedergabe der Daten. Wie die Angaben, ob man einen Treffer gelandet hat oder nicht. Weiter Daten wie die Anzahl und die Größe der Schiffe sind in Zeile 147. In Zeile 149 bis 155 befindet sich das Spielfeld für Spieler 1 und 2 welche schon das „Guessboard“ von den vorherigen Anhängen erläutert wurde. Wir mussten diesen Teil am Ende einfügen, da wir schon zwei vorherige Prototypen des Spieles programmiert hatten und dabei, das System Probleme hatte diese zu erkennen bzw. zu bearbeiten. Nach vielen weiteren Versuchen, sind wir im „Deutschen-Python Forum“ auf eine Aussage gestoßen, dass die Reihenfolge nicht unbedingt eine Rolle in Python spielt wie in Java, genauer gesagt die Anordnung und die Verknüpfung der Methoden. Dadurch konnten wir dieses Problem korrigieren. Eine Sache, welche wir hinzugefügt hatten, weil wir uns spaßeshalber über die Theorie mit dem unendlich tippenden Affen gesprochen haben, dachten wir uns, um das Spiel nicht in die Länge zu ziehen eine Abbruchbedingung zu bauen.



```

160     # spieler1 schießt
161     versuch1 = treffer1 + verfehlt1 + versenkt1
162     schuss1 = get_schuss(versuch1)
163     schiffe1, treffer1, verfehlt1, versenkt1, missed1 = check_shot(schuss1, schiffe1, treffer1, verfehlt1, versenkt1)
164     zeig_spiel_feld(treffer1, verfehlt1, versenkt1)
165
166     # Überprüft, ob es noch übrige Schiffe von Spieler 1 gibt
167     if pruefen_ob_leer(schiffe1):
168         print("VICTORY - Spieler 2 gewinnt")
169         break
170
171     # spieler2 schießt
172     versuch2 = treffer2 + verfehlt2 + versenkt2
173     schuss2 = get_schuss(versuch2)
174     schiffe2, treffer2, verfehlt2, versenkt2, missed2 = check_shot(schuss2, schiffe2, treffer2, verfehlt2, versenkt2)
175     zeig_spiel_feld(treffer2, verfehlt2, versenkt2)
176
177     # Überprüft, ob es noch übrige Schiffe von Spieler 2 gibt

```

In diesem Teil des Codes geht es um die Aktionen von Spieler 1 und 2, bzw. um die Züge, die vom jeweiligen Spieler gemacht werden. Je nachdem ob man Trifft oder verfehlt wird einem Angezeigt welche Aktion ausgeführt wurde. Durch die Theorie mit dem „Unendlich tippenden Affen“ haben wir unsere Abbruchmethode nochmal eingebaut, falls vom Gegenüber keine Schiffe übrig bleiben sollte, wie in Zeile 166 bis 169 zu sehen ist, ist ein Sieger entschieden.

```

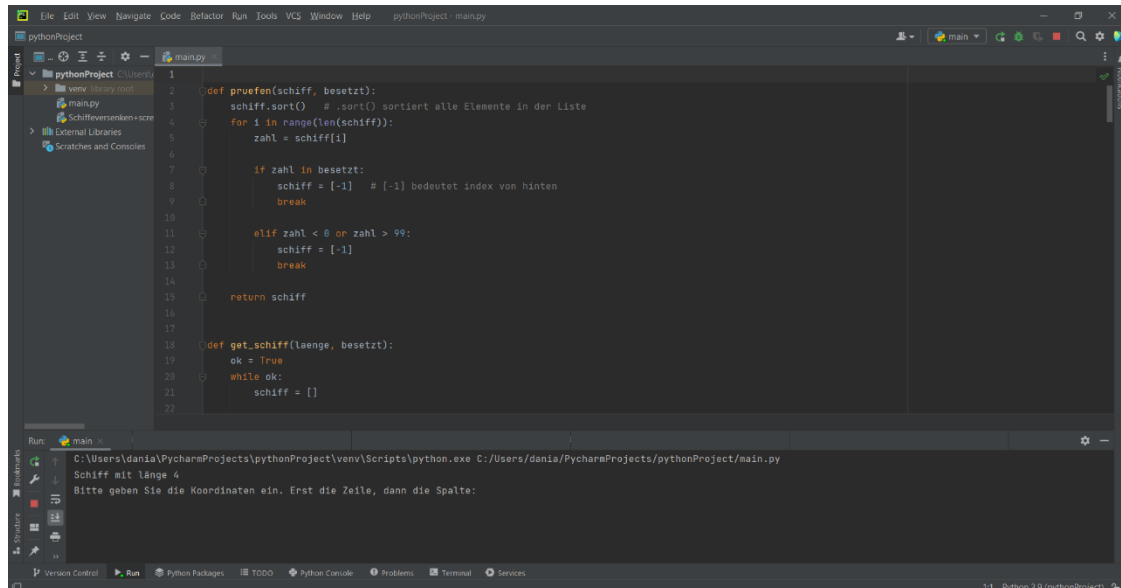
177     # Überprüft, ob es noch übrige Schiffe von Spieler 2 gibt
178     if pruefen_ob_leer(schiffe2):
179         print("VICTORY - Spieler 1 gewinnt")
180         break
181
182

```

Das gleiche wie im Anhang zuvor mit der gleichen Methode, nur für Spieler 2.

## 5. Programmbeispiel

### Eine Kurze Spieldemonstration



```
1 def pruefen(schiff, besetzt):
2     schiff.sort() # .sort() sortiert alle Elemente in der Liste
3     for i in range(len(schiff)):
4         zahl = schiff[i]
5
6         if zahl in besetzt:
7             schiff = [-1] # [-1] bedeutet index von hinten
8             break
9
10        elif zahl < 0 or zahl > 99:
11            schiff = [-1]
12            break
13
14        return schiff
15
16 def get_schiff(laenge, besetzt):
17     ok = True
18     while ok:
19         schiff = []
20         # ... (rest of the code is partially visible)
```

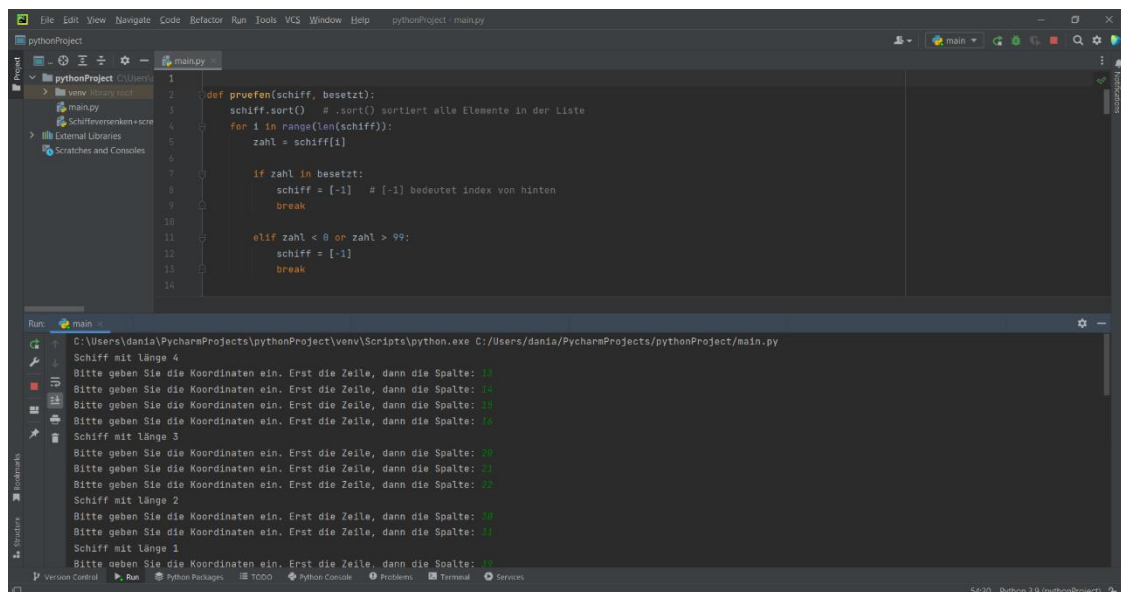
Run: main

C:\Users\ania\PycharmProjects\pythonProject\venv\Scripts\python.exe C:/Users/dania/PycharmProjects/pythonProject/main.py

Schiff mit Länge 4

Bitte geben Sie die Koordinaten ein. Erst die Zeile, dann die Spalte:

Konsole fragt nach den gewünschten Koordinaten.



```
1 def pruefen(schiff, besetzt):
2     schiff.sort() # .sort() sortiert alle Elemente in der Liste
3     for i in range(len(schiff)):
4         zahl = schiff[i]
5
6         if zahl in besetzt:
7             schiff = [-1] # [-1] bedeutet index von hinten
8             break
9
10        elif zahl < 0 or zahl > 99:
11            schiff = [-1]
12            break
13
14        return schiff
15
16 def get_schiff(laenge, besetzt):
17     ok = True
18     while ok:
19         schiff = []
20         # ... (rest of the code is partially visible)
```

Run: main

C:\Users\ania\PycharmProjects\pythonProject\venv\Scripts\python.exe C:/Users/dania/PycharmProjects/pythonProject/main.py

Schiff mit Länge 4

Bitte geben Sie die Koordinaten ein. Erst die Zeile, dann die Spalte: 11

Bitte geben Sie die Koordinaten ein. Erst die Zeile, dann die Spalte: 14

Bitte geben Sie die Koordinaten ein. Erst die Zeile, dann die Spalte: 19

Bitte geben Sie die Koordinaten ein. Erst die Zeile, dann die Spalte: 16

Schiff mit Länge 3

Bitte geben Sie die Koordinaten ein. Erst die Zeile, dann die Spalte: 10

Bitte geben Sie die Koordinaten ein. Erst die Zeile, dann die Spalte: 21

Bitte geben Sie die Koordinaten ein. Erst die Zeile, dann die Spalte: 22

Schiff mit Länge 2

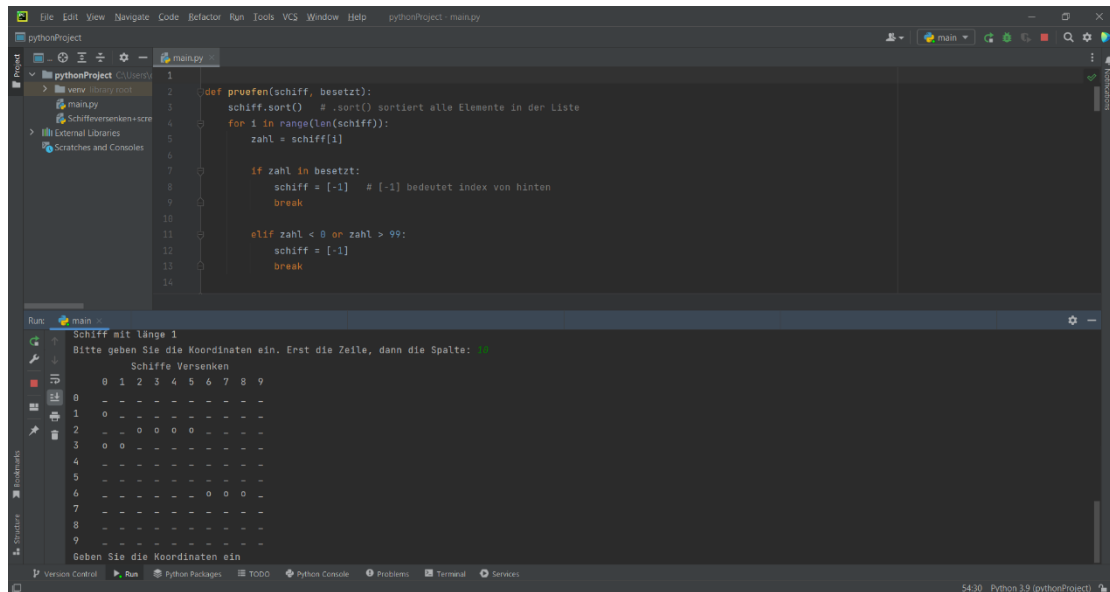
Bitte geben Sie die Koordinaten ein. Erst die Zeile, dann die Spalte: 10

Bitte geben Sie die Koordinaten ein. Erst die Zeile, dann die Spalte: 11

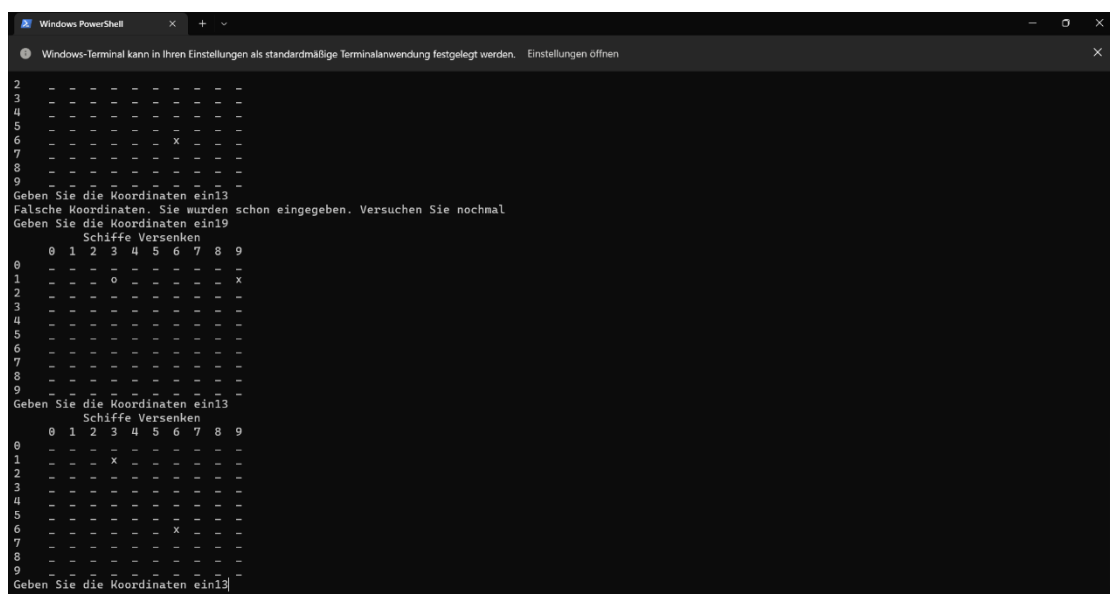
Schiff mit Länge 1

Bitte geben Sie die Koordinaten ein. Erst die Zeile, dann die Spalte: 17

Die Koordinaten des platzierten Schiffes des Spielers werden eingegeben, wie in der Konsole in Grün markiert zu sehen ist.



Das Spielfeld an sich mit den individuell platzierten Schiffen des Spielers. -> o markiert die stelle des Schiffes hier zur Übersicht.



Auch in der Windows PowerShell läuft das Programm einwandfrei. Hier im Anhang wird ein genaueres Spielgeschehen visualisiert, wo man anhand der Symbole o, erkennen kann, wo die Angriffe auf die Schiffe getroffen haben, und bei x verfehlt haben.