Sommersemester 2021

Fachbereich 2 Informatik

Betriebssysteme und Rechnernetze

Leitung: Prof. Dr. Christian Baun

SCHIFFE-VERSENKEN

Dokumentation

Abgabetermin: 28.06.2021

Bearbeitungszeit: ca. 6 Wochen

Von:

Amine Azzabakh (1354357), Dusan Milenkovic (1269073),

Mohamed El Hadien (1356164)

Studiengang IBIS

2.Fachsemester

Inhaltsverzeichnis

Inhaltsverzeichnis

1. Einleitung

Historie über das Spiel Schiffe versenken und der Programmiersprache Python.

Das Spiel Schiffe versenken kommt gattungsmäßig aus dem Bereich der Kriegsspiele. Zu früheren Zeiten wurde das Spiel traditionell mittels Stifts und Papier gespielt. Damals hat man zwei Spieler benötigt, um das Spiel spielen zu können, jedoch ist es in der heutigen Zeit möglich, auch alleine gegen die CPU spielen zu können auf dem Computer. Heutzutage wird das Spiel Schiffe versenken in dutzenden von Varianten ausgeführt, sei es als Brettspiel, als Computerspiel oder als schlichte Variante mit Stift und Papier. Das klassische Spielfeld besteht aus einem Quadrat, bestehend aus Kästchen. An den Seiten werden die Kästchen mit Buchstaben und Zahlen beschriftet. Daraufhin stellt man seine Flotte an Schiffen, ohne dass der Gegner es sieht, auf seinem eigenen Spielfeld aus. Die Größe der Schiffe variiert von Spielvariante zu Spielvariante. Das Ziel ist es, die gegnerische Flotte anhand von Koordinaten zu treffen und zu versenden. Wer zuerst die ganze Flotte lahmgelegt hat, gewinnt das Spiel. Da uns dieses Spiel auch von der Kindheit bekannt ist, haben wir uns deshalb für dieses Werkstück entschieden.

Als nächstes kommen wir, auf das Thema Python als Programmiersprache zu sprechen und warum wir diese gewählt haben. Die Programmiersprache Python wurde in den 1990er-Jahren von Guido van Rossum entwickelt in Amsterdam. Besonderheiten an Python sollte vor allem die Einfachheit und die Übersichtlichkeit des Programmes sein. Python basierende Skripte und Projekte sollen, durch die nicht so große Komplexität einfacher und knapper formuliert werden. Zudem ist Python eine Multiparadigmensprache. Das bedeutet, dass man nicht nur einen einzigen Programmierstil verwenden muss, sondern den optimalen Programmierstil wählen kann zur Lösung der Aufgabe.

Wir haben uns entschieden, die Programmiersprache Python zu verwenden, da wir durch das objektorientierte Programmieren im 1 Semester viele Gemeinsamkeiten gefunden haben, die das Erlernen der neuen Programmiersprache um einiges einfacher gemacht haben. Da im Bezug zu Java vor allem die Syntax sehr ähnlich war, fiel es uns einfacher mit Python klarzukommen. Zudem hat sich Python durch die relativ einfache und simple Lesbarkeit zum schnellen Favoriten bei uns entwickelt. Als wir uns auch andere Programmiersprachen angeguckt haben, ist uns sofort aufgefallen, dass Python Programme in der Sprache kürzer und transparenter sind. Das heißt mit weniger Code dieselbe Funktion erlangen. Im Großen und Ganzen lief die Arbeit mit Python sehr gut und hat zudem einen großen und einfach zu verstehenden Lernfaktor mitgebracht.

2. Grundlegendes

Battleship ist ein Zweispieler Spiel. Jeder Spieler hat ein 10 Felder*10 Felder großes Spielfeld. Die Spielfelder werden durch zwei zweidimensionale Arrays player1Board und player2Board dargestellt.

Auf dem Spielfeld können unterschiedliche Schiffe platziert werden. Dazu stehen die folgenden Schiffe zur Verfügung:

Schiffgröße / Feldern	Anzahl der Schiffe
2	2
3	1
4	1
5	1

Die Schiffsdefinitionen sind in einem Dictionary gespeichert von der Art

shipSizes={“Größe-Anzahl der Schiffe“:Anzahl der Schiffe,}

Auf dem Spielfeld können die folgenden Elemente platziert werden:

Element	Darstellung
Wasser	_
Schiff	||
Treffer	X
Schuss ins Wasser	*

Wir haben uns für eine alternative Platzierungsweise entschieden, in der wir von jedem Boot die Koordinate angeben, sprich man muss die Schiffe logisch setzen, damit es nicht zu Chaos oder Fehlern kommt.

3.Probleme und Lösungen

Wir haben uns zu Beginn des Projekts schwergetan, einen Anhaltspunkt zu finden, da das Internet voll von Tutorials zu dem Thema ist. Darum haben wir uns dagegen entschieden, uns an einer dieser Varianten zu orientieren, weil dies die meiste Zeit zu copy and paste führt. Unser erster Schritt war das Board zu erstellen. Dies führte nicht zu Komplikationen, da wir auf die List comprehension Funktion von Python gestoßen sind, die uns in einem Durchlauf ermöglichte, durch die Schleifen zu Iterieren und unsere Felder mit einem „-“ zu füllen und dies in unserer Arraylist speichern zu können. Vorläufig haben wir eine Methode namens „***def hitMarker***“ erstellt und mithilfe der “.replace“ Funktion

versucht, unsere Treffer mit einem „X“ zu kennzeichnen. Dies stellte sich als Problem heraus, da wir realisiert haben, dass jedes einzelne Feld eine Koordinate erhalten muss. Auf der Suche nach Koordinaten, bekamen wir heraus, dass durch das B1enutzen von simplen ineinander geschachtelten For-Schleifen und das Benutzen unserer x und y Werte, wir schnell unsere Koordinaten erhalten würden. Diese jedoch nicht im Bezug zu unserem Feld waren und wir damit auch keine Möglichkeit gefunden haben diese Koordinaten unseren Feldern hinzuzufügen. Denn dies wären zwei Arraylists die in keinem Zusammenhang stehen würden. Darauf stießen wir auf einen Beitrag, der mithilfe von Enumerate von Values und Keys mittels For-Schleife einem ermöglichte, einzelnen Elementen in einer Arraylist einem Key zuzuordnen. Durch das return Board, konnten wir die Arraylist durch Parameter Übergabe in unsere Vorläufige Funktion “***def hitMarker***“ übergeben, welche uns erlaubte, auf die bestehende Arraylist zuzugreifen und mittels Enumerate durch zwei ineinander geschachtelte For-Schleifen unserem Board an jeder Stelle eine Koordinate zu geben. Im Anschluss wurde die Funktion “***def hitMarker***“ zu “***def displayBoard***“, die uns ermöglichte das Spielfeld zu jederzeit Darzustellen. Im weiteren Verlauf erübrigte sich uns ein Problem. Das Board von Spieler Nummer zwei und das Board des Computers. Dies stellte sich als schlaue Implementierung dar, weil wir durch diese zwei Funktionen im Nachhinein jeweils das gewünschte Board darstellen konnten. Zudem konnten wir darauf zugreifen und dieses auch manipulieren.

Im folgenden Absatz möchten wir auf das Problem eingehen, welches hervortrat, als wir die Funktion für die KI erstellten. Dies jedoch nur, falls die KI einen Treffer landet, welcher in umliegender Nähe platziert wird. Dieses Problem gingen wir an, indem wir eine eigene Funktion erstellten, die das Spielverhalten des Computers definierte. Zunächst gingen wir folgendermaßen vor. Durch die importierte Bibliothek Random generierten wir Zufallswerte für unsere Computer Treffer. Mit diesen Werten haben wir angefangen jeweils durch If-Abfragen Bedingungen zu erstellen, um an der jeweiligen Koordinate eine der beiden Achsen um eins zu erhöhen oder zu verringern. Dies führte jedoch zu nichts, da wir keine Bedingung erstellen konnten, welches für jedes einzelne Feld überprüfen konnte, ob dieses schon getroffen wurde. Somit kamen wir zum Entschluss unserer Arraylist hitposition ein Schlüsselwort hinzuzufügen, um damit einen Ablauf zu generieren, der im Uhrzeigersinn agiert. Die Schüsselwörter dienten der Orientierung (‘D‘= Down, ‘U‘= Up, ‘L‘= Left, ‘R‘= Right). So war es uns möglich mit einer umfangreichen If-Verschachtlung das Schlüsselwort in unserer Arraylist umzuändern. Dies gelang durch den Zugriff auf das Element, welchem wir ein anderes Schlüsselwort gleichgesetzt haben (bsp. [2] == „Schlüsselwort“). Im anschließenden Test ergab sich in manchen Spielversuchen die Fehlermeldung: “list index out of range“. Es stellte sich heraus, dass diese nur manchmal auftaucht, weil der Computer in diesem Spielzug ein Schiff am äußeren Rand des Spielfelds getroffen hat. Um diese Art von Fehler zu vermeiden, mussten wir für jeden Spielzug eine Bedingung stellen, die uns ermöglichte, nicht das Spielfeld zu verlassen. Die Bedingungen waren simpel zu ergänzen, da wir für alle Achsen, Abfragen erstellt haben, die es uns ermöglichten mit dem Erhöhen oder Verringern das Spielfeld wieder zu betreten. Da unsere Achsen jeweils von null bis zehn gingen, mussten wir unsere Funktionen mit größer oder kleiner gleich null oder größer oder kleiner gleich zehn- Abfragen einschränken und in jenem Fall diese durch das Erhöhen oder Verringern um eins, um das Spielfeld wieder betreten zu können (siehe Abbildung 5).

4.Spiel-Aufbau

Im folgenden Abschnitt wollen wir ihnen, den Spiel-Aufbau erklären anhand zwei Verschiedener Varianten. Diese ermöglichen dem User, zwischen zwei Spielmodi wählen, welche feststellen, ob der User, gegen einen Computer oder einen Menschen im Mehrspielermodus spielen will.

4.1 Variante 1 (Mehrspieler) :

1. Die Spielfelder werden initialisiert
2. Spieler 1 kann seine Schiffe verteilen
3. Spieler 2 kann seine Schiffe verteilen

An dieser Stelle wird eine Schleife durchlaufen bis entweder alle Schiffe von Spieler 1 oder von Spieler 2 zerstört sind.

4. Spieler 1 gibt seinen Schuss ein
5. Der Schuss wird überprüft
6. Das Spielfeld wird dargestellt
7. Spieler 2 gibt seinen Schuss ab
8. Der Schuss wird überprüft
9. Das Spielfeld wird dargestellt.

4.2 Variante 2 (Computer) :

1. Die Spielfelder werden initialisiert
2. Spieler 1 kann seine Schiffe verteilen
3. Für den Computer werden die Schiffe automatisch verteilt

An dieser Stelle wird eine Schleife durchlaufen bis entweder alle Schiffe von Spieler 1 oder von Spieler 2 zerstört sind.

4. Spieler 1 gibt seinen Schuss ein
5. Der Schuss wird überprüft
6. Das Spielfeld wird dargestellt
7. Der Computerspieler gibt seinen Schuss ab
8. Der Schuss wird überprüft
9. Das Spielfeld wird dargestellt.

5.Funktionen

5.1 Die Funktion ***def grid(xaxis,yaxis)*** initialisiert die beiden Spielfelder.

Durch zwei Schleifen jeweils für das Playerboard1 und Playerboard2 konnten wir durch eine List comprehension, das Blanke Spielbrett erstellen und diese in Listen einspeichern, welche wir dem jeweiligen Playerboard zugewiesen haben.

```
8
9   # Das Grid erstellen
10  #  Wir erstellen ein Board in einem 2D array(Liste) mit dem wir spielen
11  def grid(xaxis, yaxis):
12      player1Board = []
13      player2Board = []
14      for x in range(0, xaxis):
15          tmpArry = ['-' for y in range(0, yaxis)] # List comprehension
16          player1Board.append(tmpArry) # Wir fügen die Elemente der Liste hinzu
17      for x in range(0, xaxis): # Zweites Board für den zweiten Spieler (man kann nicht sehen was der andere gemacht hat)
18          tmpArry = ['-' for y in range(0, yaxis)]
19          player2Board.append(tmpArry)
20
21      return player1Board, player2Board
22
```

Abb.1: Initialisierung der Spielfelder

5.2 Die Funktion ***def displayBoard(gridArray,show='menu')*** zeigt das ausgewählte Spielfeld an.

Dazu werden zwei ineinander geschachtelte For-Schleifen durchlaufen. Die äußere Schleife durchläuft die y-Achse. Die innere Schleife durchläuft die x-Achse. In der inneren Schleife werden die in dem Feld gespeicherten Elemente nebeneinander ohne Zeilenumbruch ausgegeben. Bei jeder Iteration der äußeren Schleife wird ein Zeilenumbruch gesetzt. So ergibt sich das quadratische Spielfeld. Durch das enumerieren der x und y-Achse im Spielfeld, konnten wir den Bindestrichen eine genaue Position zuordnen, die uns im späteren Spiel Verlauf erlaubt, durch Koordinaten auf ein Feld zuzugreifen. Zudem haben wir in dieser Funktion Schlüsselwörter eingebaut, wie “menu, Show, Hit“ etc. Das Wort “menu“ wurde in dem Fall default gesetzt, sodass wir durch die Übergabe von Parametern wie “Show“ das leere Spielbrett anzeigen können (oder durch Hit, einen Treffer oder Nichttreffer).

```
24  # die Funktion displayBoard wird das Board unseren Anforderungen nach Anzeigen
25  def displayBoard(gridArray, show='menu'): # menu ist hier default gesetzt
26      for index1, i in enumerate(gridArray): # ["a","b"] -> enumerate -> [(1,"a"),(2,"b")] Tupel von Elementen
27          for index2, j in enumerate(i):
28              #durch das enumerate wurden haben die einzelnen Bindestriche eine Nummer bekommen auf die man zugreifen kann
29              # Durch den Index1&2 wird an der jeweiligen Koorodinate überprüft ob diese schon ein Hit oder Miss ist
30              # um dieses im Nachhinein auf unser Board zu printen
31              # Beim Auswählen von Menu wird durch den Input nix angezeigt
32              if show == 'menu':
33                  print('_', end=' ')
34              elif show == 'Show': # 'Show' um das Board anzuzeigen
35                  print(gridArray[index1][index2], end=' ')
36              elif show == 'Hit': # 'Hit' um die Treffer anzuzeigen oder die Misses
37                  if gridArray[index1][index2] == 'X': # X = Hit
38                      print('X', end=' ')
39                  elif gridArray[index1][index2] == '*': # * = Miss
40                      print('*', end=' ')
41                  else:
42                      print('_', end=' ')
43          print('\n')
44
```

Abb.2: Darstellung des Spielfeldes

5.3 Die Funktion ***def placeShipsManually(playerBoard)*** ermöglicht das Platzieren der Schiffe.

Hier kann der Spieler seine Schiffe manuell auf dem Spielfeld verteilen.
Eine äußere Schleife durchläuft alle möglichen zu setzenden Schiffe.
Eine innere While-Schleife wird so lange durchlaufen, bis das Setzen erfolgreich war.
Dazu gibt der Spieler die Reihe und Spalte an, in welche sich das Schiff befinden soll. Die übergebenen Koordinaten werden anschließend überprüft, indem das Programm kontrolliert, ob das Feld ungleich Boot ist (Siehe abb 3; Zeile 65). Dies geschieht, um sicherzugehen, ob die Position nicht bereits belegt ist.

```
42  # Funktion um die Schiffe manuell zu plazieren durch user input
43  def placeShipsManually(playerBoard):
44      ships = {}
45      count = 1
46      # erstellen von Schiffen und diese im Dictionary speichern
47      for allShips, sizeOfShip in shipSizes.items():
48          # .items() holt sich Elemente aus Dictionaries
49          # allShips = Schlüßel von shipSizes
50          # sizeOfShip = Wert von shipSizes (Zeile 19)
51
52          allShip = int(allShips.split('-')[1])  # mit .split nimmt er das '-' aus dem Schlüsselpaaren raus
53          for ship in range(allShip):
54              tmpArr = []
55              displayBoard(playerBoard, 'Show')  # Board wird mit show geprinted
56              print('Platziere {} Schiff/e der länge {}'.format(allShip, sizeOfShip)) # hier wird durch format die Anzahl und laenge angezeigt im Print
57              for size in range(sizeOfShip):
58                  while True:
59                      x = int(input('Reihe : '))  # input Reihe
60                      y = int(input('Spalte : '))  # input Spalte
61                      if playerBoard[x][y] != '||':  # überprüft ob die Postion bereits belegt ist
62                          playerBoard[x][y] = '||'
63                          tmpArr.append((x, y))
64                          print('erfolgreich hinzugefügt\n')
65                          break
66              print('Schiff erfolgreich platziert')
67              ships[count] = tmpArr  # Jede Position des Schiffes wird im Dictionary abgespeichert
68              count += 1
69      displayBoard(playerBoard, 'Show')  # in jedem Durchlauf der Schleife wird das Board neu geprinted
70      return playerBoard, ships
```

Abb.3: Platzieren der Schiffe

5.4 Die Funktion ***def createComputerShips(playerBoard,xaxis,yaxis)*** setzt die Schiffe des Computers.

Dazu wird der Startpunkt des Schiffes zufällig gewählt und durch import Random, den Variablen x und y zufällige Integer Werte übergeben. Durch zwei ineinander geschachtelte For-Schleifen lässt das Programm die Anzahl der Verfügbaren Schiffe platzieren. Mit demselben Code lassen wir an der Stelle nochmal prüfen, ob das jeweilige Feld verfügbar ist.

Ausgehend von dem Startpunkt wird geprüft, ob eine Zeile drunter noch frei ist. Sollte dies der Fall sein, wird der nächste Baustein dorthin gesetzt. Falls nicht wird das Schiff nach oben erweitert. Das wird so lange durchgeführt, bis das ganze Schiff gesetzt wurde.

Wenn ein Schiff gesetzt wurde, nimmt dieser den nächsten Wert aus der Arraylist und durchläuft diesen Code von vorne.

```python
# diese Funktion erstellt die Schiffe sowie das Board für den Computer
def createComputerShips(playerBoard, xaxis, yaxis):
    shipsSize = [2, 2, 3, 4, 5]
    ships = {}
    shipCounter = 1
    count = 0
    # es werden wieder 5 Schiffe erstellt wie in shipsSize gegeben

    for size in shipsSize:
        tmpArr = []
        for i in range(size):
            if count == 0: # Macht im ersten Durchgang (bis counter größer als 0)
                while True:
                    x = randint(0, xaxis - 1)
                    y = randint(0, yaxis - 1)
                    # hier werden zufalls Koordinaten erstellt
                    try: # try, except und pass wurden eingebaut falls an der Stelle errors auftreten das diese ignoriert werden
                        if playerBoard[x][y] != '||': # diese werden genau wie davor überprüft
                            playerBoard[x][y] = '||'
                            tmpArr.append((x, y))
                            count += 1
                            break # falls erfolgreich wird die Schleife unterbrochen
                        else: # falls nicht erfolgreich wird dies solang wiederholt bis eine freiepostion gefunden wurde
                            x = randint(0, xaxis - 1) # Zufallszahl zwischen 0 und 9
                            y = randint(0, yaxis - 1)
                    except:
                        pass
            else: # nimmt die xachsen postion und arbeitet mit ihr weiter
                y = y + 1 # y wird um 1 erhöht, dass das Schiff in richtiger reihenfolge aufgestellt wird
                while True:
                    try:
                        if playerBoard[x][y] != '||':
                            playerBoard[x][y] = '||'
                            tmpArr.append((x, y))
                            count += 1
                            break
                        else:
                            y = y - 1 # falls das schiff an der yachse besetzt ist versuchen wir mit der Schleife
                            # diese dann um 1 zu verringern sodass das Schiff richtig steht
                    except:
                        y = y - 1
        ships[shipCounter] = tmpArr # Schiffe werden pro Iteration hinzugefügt bsp ships = { 1: [2,4], 2: [5,8] }
        shipCounter += 1

        count = 0 # count wird auf 0 gesetzt und die Schleife fängt von neu

    return playerBoard, ships
```

Abb.4: Erstellen der Computer Schiffe

5.5 Die Funktion ***def computerPlay(xaxis,yaxis,hitPosition,hit)*** enthält das Spielverhalten des Computers.

Wenn der Schuss des Computers ein Treffer war, wird die nächstliegende Koordinate geprüft. Dabei kann die zu prüfende Richtung übergeben werden. Dies haben wir in unserer Funktion mit hitposition, in der unsere zufälligen Werte von x und y reingeladen werden, sowie eine Startposition, die wir mit Buchstaben definiert haben. Um mögliche Fehler/Errors zu vermeiden, haben wir in unserer If-Abfrage erstmal überprüft, ob wir uns am Spielrand befinden durch kleiner Null (siehe Code Zeile 137) oder größer 10 (siehe Code Zeile 150). Durch Abfrage konnten wir sichergehen, falls der Hit des Computers ein Boot am Rand getroffen hatte, dieser den nächsten Hit nicht außerhalb des Spielfeldes setzte und somit eine Fehlermeldung ausgelöst hätte. Falls dies nicht zutrifft, soll dieser jeweils die nächstumliegenden Felder in Visier nehmen und diese im Uhrzeiger Sinn angreifen.

```
121  # Funktion welche die Computer züge macht
122
123  def computerPlay(xaxis,yaxis,hitPosition,hit):
124      # fängt mit zufälligen Koordinaten an
125
126      if hit == 'Yes':  # falls Hit gleich Yes ist werden zufällige Koordinaten für den nächsten Hit bestimmt
127          x = randint(0,xaxis-1)
128          y = randint(0,yaxis-1)
129          hitPosition = [x,y,'D']# hitposition ist in dem Fall unsere Start Position und er prüft immer von D als erstes
130          hit = 'No'# wenn hit nicht gleich Yes ist sondern No wird er durch den Code aufgefordert den nächsten Hit
131          # in einem Umliegendem Feld zu setzen
132      elif hitPosition[2] == 'D':# wenn das dritte Element gleich 'D' ist was in unserem Fall stimmt (zeile 133)
133          if hitPosition[1]-1 < 0:# wenn y Wert kleiner 0 wenn wir uns am Rand befinden
134              hitPosition[1] = hitPosition[1]+1# wird dieser um plus 1 erhöht
135          else:
136              hitPosition[1] = hitPosition[1]-1# falls nicht wird dieser um 1 verringert
137          hitPosition[2] = 'L'# Hitpostions drittes Element wird 'L' gleichgesetzt
138      elif hitPosition[2] == 'L':
139          if hitPosition[0]-1 < 0: #falls der x Wert kleiner als Null ist
140              hitPosition[0] = hitPosition[0]+1# wird dieser um 1 erhöht
141          else:
142              hitPosition[0] = hitPosition[0]-1# falls nicht wieder verringert
143          hitPosition[2] = 'U'# drittes Element wird zu 'U'
144
145      elif hitPosition[2] == 'U':
146          if hitPosition[1]+1 > 10:# falls der y achsen Wert größer 10 ist wird um 1 verringert weil wir uns ausserhalb
147              #des Spielfelds befinden
148              hitPosition[1] = hitPosition[1]-1
149          else:
150              hitPosition[1] = hitPosition[1]+1  # der gleiche Prozess für die Restlichen
151          hitPosition[2] = 'R'
152
153      elif hitPosition[2] == 'R':
154          if hitPosition[0]+1 > 10:
155              hitPosition[0] = hitPosition[0]-1
156          else:
157              hitPosition[0] = hitPosition[0]+1
158          hitPosition[2] = 'D'# dritte Element wird wieder D gesetzt, damit der Prozess beim nächsten Hit von vorne beginnen kann
159      return hitPosition, hit
```

Abb.5: Computer Spielverhalten

5.6 Die Funktion ***def checkHit(playerBoard, ships, cordinates, totalShipsDestroyed)*** prüft, ob ein Schuss gleichzeitig auch ein Treffer ist.

Es wird geprüft, ob an der Koordinate es ein Schiff gibt, welches bereits getroffen wurde. Wenn dies der Fall ist, handelt es sich um einen Treffer.

Wenn ein Schiff getroffen wurde, wird zunächst geprüft, ob der Treffer ausreicht, um das ganze Schiff zu zerstören.

Dies haben wir folgendermaßen implementiert. Wir haben aus unserer Liste Playerboard die Koordinaten entnommen und diese in Schlüssel und Werte unterteilt . Diese wurden danach in einer For-Schleife durchlaufen. Anschließend wird den Koordinaten ein Key zugewiesen, mit denen wir anschließend prüfen können, ob die Position einem Hit gleicht. Mit der Abfrage “If value.count(‘X’) == len(Value) (länge vom Schiff)“ prüfen wir, ob der User oder Computer ein ganzes Schiff zerstört hat. Dies wird dann genutzt, um den Spielstand der beiden Spieler oder den Spielstand des Computers anzuzeigen.

```
# Die Funktion überprüft ob es sich um einen Hit or Miss handelt
def checkHit(playerBoard, ships, cordinates, totalShipsDestroyed):
    x = cordinates[0] # Koordinaten die der Spieler eingibt (Reihe und Spalte)
    y = cordinates[1]
    # wenn sich an der Koordinaten Stelle ein Schiff befindet sagen wir dass es ein Hit ist
    if playerBoard[x][y] == '||':
        print('Hit')
        hit = 'No'
        playerBoard[x][y] = 'X' #an der Stelle wird das Schiff mit dem X ersetzt

        displayBoard(playerBoard,'Hit') # das Board wird abgerufen mit der Anzeige für einen Hit
        totalShipsDestroyed = 0 # Counter für die zerstörten Schiffe
        for key,value in ships.items(): # hier werden wieder die Schlüssel und Werte aus unseren playerboard entnommen (Koordinaten)
            # ships = { 1: [2,4], 2: [5,8] }
            # 2,5 => [2,5] nicht in cordinates
            # 2,4 => [2,4] in cordinates

            if cordinates in value:
                index = value.index(cordinates)  #index kriegt den Wert des Values
                value[index] = 'X' # Value prüft ob die Position gleich ein Hit ist
            if value.count('X') == len(value): # Hier wird dann geprüft ob value Count der länge vom Schiff entspricht
                # wenn dies der fall ist kann ein ganzes Schiff als zerstört angeben werden
                totalShipsDestroyed += 1

    # Hier wird überprüft ob die Stelle zuvor schon getroffen worden ist
    elif playerBoard[x][y] == 'X':
        print('bereits getroffen')
        hit = 'Yes'
    # otherwise its a miss
    else:
        playerBoard[x][y] = '*'  # falls die Stelle weder eins von beidem ist wird diese mit einem Miss '*' gekennzeichnet
        print('Miss')
        hit = 'Yes'
    print('Total Ship Destroyed:',totalShipsDestroyed)  # anezeige der Zerstörten Schiffe

    return playerBoard, ships, totalShipsDestroyed,hit
```

Abb.6: Überprüfung der Treffer und nicht Treffer

6.Fazit

Um zum Fazit zu kommen, wäre zu sagen, dass wir durch unsere Erfahrungen im Bereich der Objektorientierten Programmierung aus dem 1 Semester eine gute Ausgangslage hatten, die neue Programmiersprache Python zu erlernen. Durch mehrere YouTube Videos haben wir uns mehr und mehr in die Sprache reingefunden. Durch das Experimentieren mit verschiedenen Methoden (trial und error) kam es zu der Zusammensetzung unseres Programmes. Jedoch schien uns für eine lange Zeit nicht plausibel, wie das Implementieren einer KI im Zusammenhang mit dem Spiel auf unserem Codeniveau aussehen würde. Durch diese Projektarbeit haben wir einen neuen Einblick in die Programmiersprache Python sowie ein tieferes Verständnis im Bereich der Softwareentwicklung erhalten. Abschließend wäre jedoch zu sagen, dass die Projektarbeit der ganzen Gruppe im Großen und Ganzen gefallen hat und wir gerne an dem Projekt gearbeitet haben.

7.Quellenverzeichnis

- *Use Python to create 2D coordinate*. (2013, September 15). Stack Overflow. https://stackoverflow.com/questions/18817207/use-python-to-create-2d-coordinate/18817249
- *13. Enumerate — Python Tips 0.1 documentation*. (n.d.). Python Tips 0.1 Documentation. Retrieved June 28, 2021, from https://book.pythontips.com/en/latest/enumerate.html
- *Python Syntax*. (n.d.). W3schools. Retrieved June 28, 2021, from https://www.w3schools.com/python/python_syntax.asp
- *Programming for Advanced Beginners: Battleships - EP 1*. (2019, July 28). YouTube. https://www.youtube.com/watch?v=Gi0Fdyhk1_0&t=49s
- *random — Generate pseudo-random numbers — Python 3.9.5 documentation*. (n.d.). Docs.Python. Retrieved June 28, 2021, from https://docs.python.org/3/library/random.html
- *5. Data Structures — Python 3.9.5 documentation*. (n.d.). Docs.Python. Retrieved June 28, 2021, from https://docs.python.org/3/tutorial/datastructures.html
- *5. Data Structures — Python 3.9.5 documentation*. (n.d.-b). Docs.Python. Retrieved June 28, 2021, from https://docs.python.org/3/tutorial/datastructures.html
- *Python Tutorial deutsch /german (Crashkurs)*. (n.d.). YouTube. Retrieved June 28, 2021, from https://www.youtube.com/playlist?list=PL_pqkvxZ6ho3u8PJAsUU-rOAQ74D0TqZB
- *5. Data Structures — Python 3.9.5 documentation*. (n.d.-c). Docs.Python. Retrieved June 28, 2021, from https://docs.python.org/3/tutorial/datastructures.html
- *Python - List Comprehension*. (n.d.). W3schools. Retrieved June 28, 2021, from https://www.w3schools.com/python/python_lists_comprehension.asp