Dokumentation zum Werkstück A in „Betriebssysteme und Rechnernetze": Schiffe-Versenken

Pascal Lupo, Gamachu Tufa, Jean-Gabriel Hanania

28. Juni 2021

Zusammenfassung

In dem vorliegenden Dokument wird beschrieben, wie das Spiel Schiffe-Versenken in Python programmiert wurde. Die größeren Herausforderungen dieses Projektes werden dargestellt. Es wird vor allem auf den Aufbau und die Darstellung des Spiels, die künstliche Intelligenz mit verschiedenen Schwierigkeitsstufen für das Spiel gegen den Computer und das Einbauen einer Zeitbegrenzung für jeden Zug eingegangen.

Im Rahmen des Werkstücks A des Moduls Betriebssysteme und Rechnernetze SS2021 wurde das Spiel Schiffe-Versenken in Python programmiert. Schiffe-Versenken ist ein Spiel, das üblicherweise mit Stift und Papier gespielt wird, in dem jeder Spieler seine „Schiffe" auf einem verdeckten Spielbrett verteilt. Die Spieler „schießen" dann abwechselnd auf Felder des gegnerischen Spielbrettes, bis einer alle Schiffe des Gegners aufgedeckt und somit gewonnen hat.

1 Aufbau des Spiels

Die erste Herausforderung dieses Projektes ist es, einen sinnvollen Aufbau des Spiels und eine klare Darstellung zu finden, mit der jederzeit ersichtlich ist, was der Spielstand ist, welcher Spieler am Zug ist, wohin schon geschossen wurde und welche Möglichkeiten noch offen stehen.

1.1 Speichern des Spielstands

Zuerst ist es nötig, alle wichtigen Informationen zu erkennen und eine Speichermethode zu wählen. Für das Spiel im Allgemeinen ist es sinnvoll, die Standardwerte der Einstellungen zu speichern; z. B. die Größe der Spielfelder und die Anzahl an Schiffen, die jeder Spieler besitzen soll. Dazu wird in Python ein sogenanntes Dictionary hergestellt. Das ermöglicht ein schnelles und übersichtliches

Aufrufen der eingespeicherten Werte mit einem eindeutig genannten Schlüssel. In diesem Fall werden in `settings_values['board_size']` die Spielfeldgröße 10 und in `settings_values['number_of_ships']` die Anzahl 5 gespeichert. Das stellt die üblichen Werte dieses Spiels dar. Diese lassen sich aber nach Belieben in den Einstellungen ändern. Bei dem Start eines neuen Spiels wird dann basierend auf diesen Einstellungen eine Liste aller möglichen Felder erzeugt (Listing 1). Mit dieser lässt sich jede Feldeingabe der Spieler leicht abgleichen, um Fehler abzufangen.

```
1 possible_input = []
2 for y in range(settings_values['board_size']):
3     for x in range(settings_values['board_size']):
4         possible_input.append(chr(65 + y) + str(x + 1))
```

Listing 1: Herstellung einer Liste aller möglichen Felder

Jeder Spieler bekommt auch ein Python-Dictionary. In diesem werden alle wichtigen Informationen zum Spieler und seinem Spielstand gespeichert d. h.:

- den vom Spieler ausgesuchten Namen unter dem Schlüssel `'name'`
- eine Tabelle, auf der die Schiffe platziert und die Schüsse des Gegners eingespeichert werden (`'board'`)
- ein Tabelle der versuchten Schüsse (`'guesses'`)
- die Anzahl an Schiffen die er und sein Gegner übrig haben (`'ships_left'` und `'enemy_ships_left'`)
- eine Liste der möglichen Felder, aus der nach und nach die Felder entfernt werden, auf die der Spieler feuert (`'not_yet_tried'`)

1.2 Speicherung der Schiffpositionen

Am Anfang des Spiels wird eine Liste von Schiffen basierend auf der eingestellten Anzahl an Schiffen pro Spieler ausgesucht. Diese Listen sind vorgefertigt und jeder Eintrag darin ist ein Python-Dictionary, das ein bestimmtes Schiff darstellen soll. Dort steht, wie das Schiff heißt, mit welchem Kürzel es in der Tabelle repräsentiert wird und wie lang es ist.

Jeder Spieler entscheidet dann, ob er seine Schiffe selber platzieren möchte oder ob diese zufällig verteilt werden sollen. Wählt er die erste Option, wird er dazu aufgefordert, ein Feld auszusuchen, auf dem die Spitze des Schiffes liegen soll. Das Programm berechnet dann, ob an dieser Stelle genug Platz ist und gibt eine Liste möglicher Felder zurück, auf den das andere Ende des Schiffes liegen kann. Nur wenn der Spieler beide Werte richtig eingibt, wird das Schiff mit dem passendem Kürzel in der Tabelle `'board'` des Spielers eingetragen. Die Zufallsverteilung erfolgt ähnlich. Jeder Wert wird nur zufällig aus den Listen gültiger Eingaben ausgesucht.

1.3 Darstellung des Spieles

Jeder Spieler hat zwei Tabellen, in denen sein aktueller Spielstand gespeichert ist.

Pa1	Su1	
Pa1	Su1_Hit	
	Su1	

(a) Spielfeld

	CG	
WG		

(b) Schussversuche

Abbildung 1: Beispiel eines Spielstandes, das in Tabellen eines Spielers gespeichert ist

In der ersten Tabelle (Abbildung 1a) stehen die Positionen und der Zustand der eigenen Schiffe. Diese werden mit einem einzigartigen Kürzel bezeichnet, um sie voneinander unterscheiden zu können (bspw. Su1 für Submarine 1 und Pa1 für Patrol Boat 1). An die Schiffsteile, die der Gegner schon getroffen hat, wird `_Hit` angefügt. In der zweiten Tabelle (Abbildung 1b) werden die Schussversuche gespeichert. Das Kürzel WG (Wrong Guess) makiert die Schüsse, die verfehlt worden sind und CG (Correct Guess) steht für die Felder, auf denen ein gegnerisches Schiff getroffen wurde.

Daraus wird dann ein Bauplan erstellt. In diesem werden nicht nur die Daten aus den zwei Tabellen eingefügt, sondern auch die Abstände, die bei dem Ergebnis angezeigt werden sollen, die Beschriftung der Zeilen und Spalten und die Linie, die das Raster des Spielfeldes darstellen.

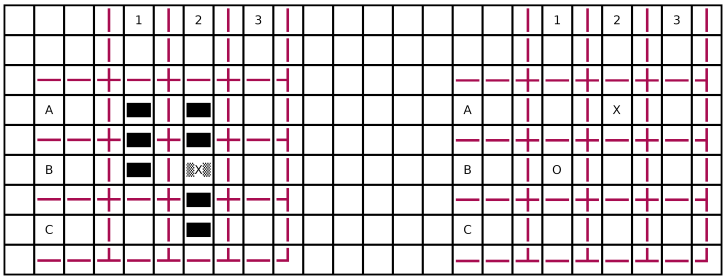

Abbildung 2: Bauplanergebnis aus dem Spielstand von Abbildung 1

Zum Einfügen der verschiedenen Zeichen an der richtigen Stelle werden für alle im Spielfeld liegenden (x,y)-Koordinaten Kombinationen mithilfe der Modulo 2 Funktion berechnet, ob die Koordinaten gerade oder ungerade Zahlen sind. In dem nachfolgenden Listing 2 wird bspw. für jedes Feld, das im richtigen Bereich liegt und ein ungerade x-Koordinate besitzt, eine vertikale Linie eingefügt.

```
for y in range(blueprint_height):
    for x in range(blueprint_width):
        # Filling in the vertical walls for both boards. They all have odd x coordinate
            if x % 2 == 1 and ((2 < x < (len(player['board']) * 2 + 4)) or (
                    (len(player['board']) * 2 + 4 + board_spacing) < x < blueprint_width)):
                blueprint[y][x] = "|"
```

Listing 2: Beispiel zum Einfügen von bestimmten Teilen des Spielfeldrasters

Der Zwischenschritt des Bauplans ermöglicht eine höher Kontrolle der Anzeige; z. B. liegen alle Schiffe durch ihre Länge über mindestens einer Linie des Rasters und würden dadurch in der Anzeige unterbrochen sein. Das würde für Unklarheiten sorgen, sobald zwei Schiffe aneinander liegen würden. Im Bauplan wird für jede Linie aus dem Spielfeld geprüft, ob in den dazu angrenzenden Feldern Teile eines gleichen Schiffes liegen. Damit kann dieses Zeichen ggf. dann mit einem passenden Schiffszeichen ersetzt werden. Da bspw. in der Abbildung 2 die Felder A1 und B1 beide in der Spieler-Tabelle mit "Pa1"versehen sind, wird die Linie zwischen diesen Felder mit einem Schiffsteil ersetzt.

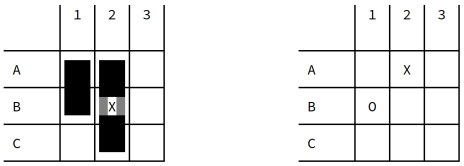

Abbildung 3: Anzeigeergebnis

Die Abbildung 3 ist das im Spiel angezeigte Ergebnis aus dem Spielstand der Abbildung 1. Zum Darstellen der Schiffe und der Rasterlinien kommen die sogenannten "Box-drawing character"[1] des Unicode Standards zum Einsatz. Bei der Ausgabe wird über die Spielfelder angezeigt, welcher Spieler dran ist und wie viele Schiffe er und sein Gegner noch besitzen. All diese Werte lassen sich aus dem jeweiligen Python-Dictionary vom aktiven Spieler auslesen.

2 PvE Game

Der PvE-Modus ist dafür da, dass eine Person gegen einen computergesteuerten Spieler, einen sogenannte Bot, spielen kann. In diesem Modus kann man außerdem auswählen, wie gut die künstliche Intelligenz des Computers sein soll. Diese Einstellung soll Gegenspieler unterschiedlicher Stärken imitieren können und damit das allgemeine Spielerlebnis abwechslungsreicher machen. Die KI des Bots ist in drei Schwierigkeitsgrade unterteilt: "Einfach", "Mittel" und "Schwer".

[1] https://en.wikipedia.org/wiki/Box-drawing_character

2.1 Probleme bei der Umsetzung

Eins der größeren Probleme bei der Umsetzung des PvE-Modus war es, eine auf dem Level des Spielers anpassbare künstliche Intelligenz zu integrieren. Folgende Fragen stellen sich:
Wie sollten die Abstufungen der verschiedenen Schwierigkeitslevel aussehen? Soll der Bot auf die umliegenden Felder schießen, wenn er ein Schiff getroffen hat und nicht einfach weiter zufällig auf das gegnerische Spielfeld schießen? Wie kann der Computer im schwersten Modus taktisch vorgehen, um möglichst wenige Spielzüge zu tätigen, um ein Schiff zu finden und zu versenken, wenn der Spieler den schweren Modus einstellt?

2.2 Umsetzung

Der Schwierigkeitsmodus "Einfach" soll einen unerfahrenen Spieler imitieren, der willkürlich auf das Feld feuert. Daher schießt der Bot komplett zufällig, ohne irgendwelche Kriterien zu beachten. Dies erfolgt mit der `random.choice()`-Funktion von Python, die aus der `'not_yet_tried'`-Liste des Bots ein zufälliges Feld auswählt und es beschießt.

Der Schwierigkeitsmodus "Mittel" stellt einen Gelegenheitsspieler dar. Dafür schießt der Bot auch zuerst zufällig auf das Feld. Trifft er aber ein Schiff, wird geschaut, welche anliegenden Felder noch nicht beschossen wurden und diese zu der `next_shot` Liste hinzugefügt. Daraus werden in den nächsten Zügen Felder zufällig gewählt. Alle Treffer speichert der Computer in seiner Liste `'current_target'` ab. Falls es beim Beschießen der umliegenden Felder wieder zu einem Treffer kommt, schaut das Programm, ob die beiden Treffer die gleiche y- oder x-Koordinate haben. Im ersten Fall liegt das Schiff wahrscheinlich horizontal auf dem Spielfeld und im zweiten Fall wahrscheinlich vertikal. Nun schießt der Bot auf die anliegenden Felder mit der gleichen y- bzw. x-Koordinate. Wenn nun das Schiff versenkt wird, leert der Bot seine `'current_target'`-Liste und fängt wieder an, zufällig zu schießen, bis er erneut trifft. Falls es keine freien anliegenden Felder mehr gibt - d. h. die `next_shot` Liste kommt leer zurück - aber kein Schiff versenkt wurde, geht der Computer davon aus, dass es sich hier um mehrere aneinander liegende Schiffe handelt. Nun kopiert er die Felder aus `'current_target'` in eine neue Liste namens `'possible_target'`. Der Computer behandelt jedes dieser Felder als getrenntes Schiff und geht jedes davon durch, bis jeweils ein Schiff versenkt wird. Danach wird zufällig geschossen.

Beim Schwierigkeitsmodus "Schwer" wird ein erfahrener Spieler simuliert, der effizient spielt und versucht in möglichst wenigen Zügen alle Schiffe des Gegners zu versenken. Dafür wird erkannt, dass der Bot nur auf jedes zweite Feld schießen muss, da das kleinste Schiff zwei Felder lang ist. Falls er schon alle Zwei-Felder Schiffe versenkt hat, ändert sich seine Vorgehensweise und er schießt nur noch auf jedes dritte Feld. Das wird alles in der `smart_random_shot()`-Funktion definiert. Diese prüft, was das kleinste Schiff des Gegners ist und baut darauf

basierend verschiedene "Grids". Wenn das kleinste Schiff zwei Felder groß ist, sehen die zwei möglichen Grids so aus, wie die schwarzen und weißen Felder eines Schachbrettes. Der Computer zählt nach, wie viele offene Felder die jeweiligen Grids haben und sucht sich die mit der kleinsten Anzahl aus. Indem er seine zufälligen Schüsse nur auf dieser Grid betätigt, minimiert er die Anzahl an nötigen Versuchen, um alle Schiffe zu finden. Wenn das kleinste übrige Schiff drei Felder lang ist, ergeben sich die sechs mögliche Grids (Abbildung 4). Diese bestehen aus diagonalen Linien, die zwei Kästchen Abstand voneinander haben.

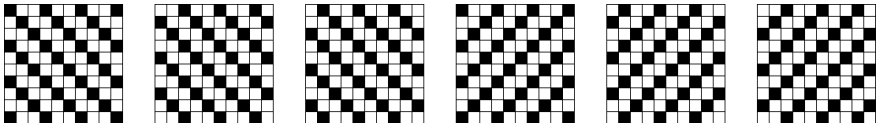

Abbildung 4: Mögliche Grids, wenn das kleinste Schiff drei Felder lang ist

Es kann passieren, dass die ideale Grid sich beim Spielen ändert. Ein Beispiel dazu zeigt die Abbildung 5: Zuerst beschießt der Bot die weißen Kästchen, da es von diesen Feldern weniger gibt als von den Schwarzen. So beginnt der Bot hier mit B1 und erzielt einen Treffer. Nun erkennt der Computer, dass dort ein Schiff sein muss und beschießt die umliegenden Felder. Der zweite Schuss geht auf B2 daneben. Mit den nächsten beiden Schüssen A1 und C1 versenkt er das Drei-Feld-Schiff. Jetzt sucht der Bot nach dem letztem Schiff und wechselt dafür zur schwarzen Grid, denn jetzt sind nur noch zwei schwarze Kästchen und drei weiße Kästchen übrig. Damit braucht er nur noch zwei Schüsse, um sicher zu sein, dass er das Schiff zu trifft.

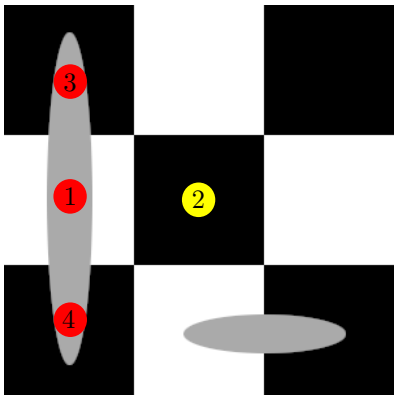

Abbildung 5: Beispiel eines Grid-Wechsels

Im Listing 3 ist ein Teil der `smart_random_shot()`-Funktion zu sehen, die bei dem schweren Modus zum Einsatz kommt, wenn das kleinste Schiff noch am Leben ist. Dort wird gezeigt, wie die zwei möglichen Grids berechnet und dann miteinander verglichen werden:

- Zuerst werden zwei leere Listen erstellt, die die zwei möglichen Grids darstellen sollen. Diese werden `hunting_grid00` und `hunting_grid01` benannt.
- Nun wird für jedes leere Feld nachgeschaut, zu welcher Grid es gehört und wird der entsprechenden Liste hinzugefügt. Dafür berechnet das Pro-

gramm, ob die x- und y-Koordinaten durch Modulo 2 das gleiche Ergebnis haben, also entweder beide 0 oder beide 1 ergeben. Diese Felder werden alle in der `hunting_grid_00`-Liste gespeichert. Die anderen Felder, also die, bei denen die x- und y-Koordinaten nicht den selben Wert ergeben, wenn sie mit Modulo 2 verrechnet werden, speichert der Computer in der `hunting_grid_01`-Liste.

- Im nächsten Schritt wählt er einen der zwei Listen zufällig als sein "Haupt"-`hunting_grid` aus. Damit soll im Falle, dass die zwei Listen gleich lang sind, sichergestellt werden, dass nicht immer die Gleiche benutzt wird.
- Diese zufällig ausgewählte Liste wird dann mit den beiden anderen Listen verglichen. Der Computer wechselt anschließend nur zur anderen Grid, wenn diese weniger Felder in ihrer Liste stehen hat.
- Der Computer kann dann ein Feld zufällig aus seiner `hunting_grid`-Liste ziehen um dorthin zu feuern.

```
1  hunting_grid_00 = []
2  hunting_grid_01 = []
3
4  # Checks every fields, and adds the empty one to the grid its part
       of
5  for y in range(len(bot['guesses'])):
6      for x in range(len(bot['guesses'])):
7          if bot['guesses'][y][x] == "0":
8              if (y % 2 == 0 and x % 2 == 0) or (y % 2 == 1 and x %
       2 == 1):
9                  hunting_grid_00.append(chr(65 + y) + str(x + 1))
10             else:
11                 hunting_grid_01.append(chr(65 + y) + str(x + 1))
12
13 # Finally, it checks which list is the smallest of the 2, and sets
       it as our hunting grid
14 # One Grid is chosen at random to reduce predicting possibilities
15 hunting_grid = random.choice([hunting_grid_01, hunting_grid_00])
16 if len(hunting_grid) > len(hunting_grid_01):
17     hunting_grid = hunting_grid_01
18 elif len(hunting_grid) > len(hunting_grid_00):
19    hunting_grid = hunting_grid_00
20
21 return random.choice(hunting_grid)
```

Listing 3: Herstellung der zwei möglichen Grids und deren Vergleich

3 PvP Game

Beim PvP-Modus ist es möglich, dass zwei Personen an einem Gerät gegeneinander spielen können. In diesem Modus können beide Spieler einen Namen auswählen und müssen sich miteinander am Computer abwechseln. Dabei soll

sichergestellt werden, dass die Spieler zu keinem Zeitpunkt das Spielfeld des Gegners zu sehen bekommen.

3.1 Umsetzung

Dieser Spielmodus sieht in der Umsetzung dem PvE-Spiel sehr ähnlich. Damit die Spieler die Position der Schiffe des anderen nicht erfahren können, wird zwischen jedem Spielzug der Bildschirm "geleert" und die Spieler werden dazu aufgefordert, die Plätze zu tauschen, bevor es weiter geht. Da ein Löschen der ganzen Konsole sehr umständlich wäre, wurde dies stattdessen mit der `clear_screen()`-Funktion gewährleistet. Diese gibt 50 leere Zeilen aus, um somit das Spielfeld des Gegners zumindestens aus dem direkt sichtbaren Bereich der Konsole zu entfernen.

4 Counter

Die Funktion ist dafür da, die Zeit der Benutzereingabe zeitlich zu begrenzen. Sobald der Spieler aufgefordert wird ein gegnerisches Feld zu beschießen, läuft ein Countdown, der aufzeigt, wie viel Zeit noch übrig bleibt. Der Zeitraum wurde auf 15 Sekunden festgelegt. Sollte der Spieler innerhalb der gegebenen Zeit ein gegnerisches Schiff beschießen, wird der Countdown beendet und das Spiel fortgesetzt, indem der andere Spieler nun aufgefordert wird, ein gegnerisches Feld zu beschießen. Sollte der Spieler bis Ablauf der Zeit noch kein Feld ausgewählt haben, wird dem Spieler angezeigt, dass seine Zeit abgelaufen ist und er nun die Enter-Taste drücken soll. Durch das Drücken der Enter-Taste wird ein zufälliges Feld beschossen und das Spiel fortgesetzt. Diese Funktion wird jedes Mal aufgerufen, wenn der Spieler ein Feld beschießen muss.

4.1 Problemstellung

Ein Problem der Aufgabenstellung war, dass zwei Prozesse gleichzeitig laufen müssen. Zum einen muss der Countdown von 15 bis 0 herunterlaufen, zum anderen muss gleichzeitig auf eine Benutzereingabe gewartet werden. Zu Beginn war das Problem, dass der Countdown erst gestartet ist, nachdem es eine Benutzereingabe gab. Dies machte den Countdown jedoch unbrauchbar, da beides parallel laufen muss.

4.2 Lösungsversuch

Um nun mehrere Prozesse gleichzeitig laufen zu lassen, musste eine andere Lösung her. Durch Recherche wurde die Benutzung von "threads" als Möglichkeit gefunden. Diese sollten einen parallelen Ablauf von mehreren Prozessen ermöglichen. Die Datei ist wie folgt aufgebaut:

- Es gibt die `ask()`-Funktion. Diese Funktion nimmt die Benutzereingabe auf und gibt sie am Ende zurück.

- Die Funktion `exit(msg)` wird aufgerufen, wenn eine Benutzereingabe erfolgt ist oder wenn keine Eingabe erfolgt ist. In beiden Fällen gibt die Funktion einen Textwert zurück, der als Parameter gegeben werden muss. Es wird beispielsweise ausgegeben, dass die Zeit abgelaufen ist und nun ein zufälliges Feld beschossen wird oder es wird ausgegeben, auf welches Feld geschossen wurde.
- Die Funktion `countdown()` gibt die verbliebene Zeit an. Innerhalb der Funktion ist eine Endlosschleife die läuft, bis der Stop-Wert null erreicht wird. Der Wert beginnt bei 15 und wird jede Runde, mit einer Verzögerung von 2 Sekunden um den Wert 2 verringert. Bei jedem Schleifendurchgang wird die aktuell verbliebene Zeit ausgegeben. Wichtig ist hierbei zu sagen, dass in der gesamten Zeit eine Benutzereingabe parallel möglich ist.
- Die Funktion `close_if_time_pass(seconds)` wird als letztes aufgerufen und gibt aus, dass die Zeit abgelaufen ist. Diese ist nur notwendig, wenn der Fall eintritt, dass es keine Benutzereingabe gab.
- Alle diese Funktionen werden in der Wichtigsten von allen, der `main()` Funktion, aufgerufen.

In der Funktion `main()`, wurde, wie bereits erwähnt, mit "threads" gearbeitet. Es waren zwei "threads" nötig. Der Erste ist dafür da, um nach Ablauf der Zeit (in unserem Fall 15 Sekunden) auszugeben, dass die Zeit abgelaufen ist und nun vom Spiel selbst, ein zufälliges Feld ausgewählt wird. Der zweite "thread" ist dazu da, den Countdown zu starten. Sobald die "threads" gestartet werden, wird die Funktion `ask()` aufgerufen, welche auf die Benutzereingabe wartet. Der Rückgabewert aus der Funktion `ask()`, wird gespeichert in der Variable `user_input`. Es wird nach Ablauf der Zeit geprüft, ob die Variable keinen Wert hat. Sollte dies der Fall sein, wird ein zufälliges Feld beschossen, mit der Funktion `random_ship_attack`, aus der Datei gamefunctions. Sollte die Variable jedoch einen Wert haben, wird dieser Wert als Rückgabewert genutzt und somit das eingegeben Feld beschossen.

```
1  def main(player, language):
2      global start_sign, check
3
4      # bool variables that are needed to start
5      start_sign = True
6      check = False
7
8      # define close_if_time_pass as a threading function, 15 as an
       argument
9      t = threading.Thread(target=close_if_time_pass, args=(15,
       player, language,))
10     t2 = threading.Thread(target=countdown, args=(language,))
11
12     # start threading
13     t2.start()
14     t.start()
15
16     # ask for input
```

```
17      user_input = ask()
18
19      # if there was no user input, the player will attack a random
        field automatically
20      if len(user_input) < 1:
21          user_input = random.choice(player['not_yet_tried'])
22
23      # bool variables that are needed to stop
24      check = True
25      start_sign = False
26
27      return user_input
```

Listing 4: main()-Funktion des Countdowns

5 Schlusswort

Dieses Pojekt war eine gute Herausforderung, um zu lernen, mit der Programmiersprache Python umzugehen. Ein paar Erweiterungsmöglichkeiten des Programms wären bspw. die Option, die Eingabe der Felder über die Maus betätigen zu können oder das Spiel über das Internet auf zwei verschiedenen Computern spielen zu können. Außerdem könnte die künstliche Intelligenz verbessert werden, indem man einen Algorithmus integrieren würde, um ein Wahrscheinlichkeitsfeld der Verteilung der Schiffe aufzustellen, sodass der Bot immer nur auf das Feld schießen könnte, auf welchem ein Schiff am wahrscheinlichsten platziert ist.