Abgabe vom Werkstück A im SS2022

„Schiffe versenken mit Interprozesskommunikation“ Alt. 7

Habib Mustafa Koca, Divya Kharbanda, Eyüp Tolgahan Yildirim, Akash Mehra, Felix-Markus Koschitzky

Frankfurt University of Applied Sciences
Nibelungenplatz 1
60318 Frankfurt am Main

Inhaltsverzeichnis:

I. Einleitung

Die Dokumentation des Projekts „Schiffe versenken mit Interprozesskommunikation“ erklärt, wie das Projekt bearbeitet und fertiggestellt wurde. Das Spiel ist in der Programmiersprache Python in Py Charm entwickelt worden. Es handelt sich dabei um ein Multiplayer-Spiel, wobei die Problemstellung darin liegt, dass die Spieler miteinander „kommunizieren“ also in Verbindung stehen.

Das Spiel funktioniert folgendermaßen. Jeder Spieler muss als Vorbereitung Schiffe auf einem gegebenen Spielfeld platzieren. Anschließend kann das Spiel beginnen. Abwechselnd werden Schüsse gefeuert mit dem Ziel, die Schiffe des Gegenspielers zu versenken. Das Spielfeld ist zunächst leer, dass heißt die Schiffe sind für den Gegenspieler nicht sichtbar. Dieser bekommt lediglich eine Rückmeldung, ob ein Fehlschuss oder Treffer erzielt wurde. Bei diesem Spiel gewinnt der Spieler, der als Erstes alle Schiffe des Gegenspielers versenkt hat.

1 Allgemeines zu diesem Dokument

Dieses Dokument umfasst die Dokumentation des Werkstücks A, erstellt im Rahmen der Portfolioprüfung des Moduls Betriebssysteme und Rechnernetze im SS2022 an der FRA-UAS. In dieser Dokumentation sind Einzelheiten und Details zum Aufbau und zur Entwicklung des programmierten Spiels „Schiffe versenken“ festgehalten. Geschildert ist, wie sich der Code zusammensetzt und welche Hilfsmittel eingesetzt sind. Durch Ausschnitte des Codeblocks ist gezeigt, welche Funktionen konkret umgesetzt sind, um auf das erzielte Endergebnis zu kommen.

II. Hauptteil

2 Ablauf des Spiels

2.1 Start des Spiels

Zunächst erscheint eine Aufforderung, die gewünschte Spielfeldgröße AxA einzugeben. Dies sieht wie folgt aus:

„Bitte geben Sie die gewünschte Spielfeldgröße ein (AxA).“.

Die Eingabe ist mit einem Enterklick zu bestätigen.

2.2 Namenseingabe

Im nächsten Schritt müssen die Spieler jeweils Ihre Wunschnamen eintragen und mit einem Enter bestätigen. Hinzu kommt die Ausgabe:

„Wie heißt Spieler1? -Spieler1:“

2.3 Platzierung

Spieler 1 platziert zunächst seine Schiffe. Hierbei werden drei Fragen/Anforderungen zur Schiffplatzierung gestellt. Die Reihenfolge lautet wie folgt:

1. *„Welche Zeile soll das x. te x-schiff gesetzt werden? Zeile: “*

Eine zulässige Zahl ist hier einzutragen und zu bestätigen (z.B. „1“, „2“, „3“, …).

2. *„In welche Spalte soll das x. te x-schiff gesetzt werden? Spalte: “*

Der Spieler muss hier einen zulässigen Großbuchstaben eingeben und bestätigen (z.B. „A“, „B“, „C“, …)

3. *„Tippe 0 für Horizontal und 1 für Vertikal.“*

Die Ausrichtung der Schiffe wählt der Spieler durch die Eingabe 0: horizontal und 1: vertikal und bestätigt diese mit Enter.

Das Programm prüft jede Eingabe nach Zulässigkeit. Es prüft, ob bereits ein Schiff an der gewünschten Position gesetzt ist, oder ob die Wunschposition außerhalb des Boards liegt.

Nach der Prüfung der Eingabe, platziert das Programm das Schiff an der gewünschten Stelle im Board.

Die Schiffe kennzeichnen sich durch den jeweiligen Anfangsbuchstaben, gesetzt im Board an der gewählten Position, sprich für „Schlachtschiff“ ist ein „s“ in jedem Feld einzusetzen, welches „Schlachtschiff“ belegt.

Nachdem beide Spieler Ihre Schiffe gesetzt haben, erscheint eine Aufforderung Enter zu drücken, um das Spiel zu starten:

„Bitte drücke Enter um das Spiel zu Starten.“

2.4 Angriff

Die Steuerung des Angriffs passiert ähnlich, wie bei der Platzierung der Schiffe durch die Eingabe einer zulässigen Zeile bzw. Spalte.

Das aktuelle Board mit dem Angriffsergebnis, wobei „x“ für einen Treffer und „0“ für einen Fehlschuss steht, erscheint auf der Shell.

Beim Wechseln der Spieler, führt das Programm einen Zeilenumbruch von 50 Zeilen mit *def clearscreen* durch, um den gegnerischen Spielzug auszublenden und die Shell optisch zu leeren.

Die Spieler wechseln sich beim Angreifen ab, bis alle Schiffe untergegangen sind. Beim Versenken eines Schiffes, erscheint eine Meldung auf der Shell mit der Information, welches Schiff versunken ist. Wenn ein weiteres gegnerisches Schiff versenkt ist, ist dieser ebenfalls in die Liste der versenken Schiffe hinzugefügt und bei jedem Zug angezeigt.

2.5 Ende

Das Spiel endet, indem einer der beiden Spieler es schafft, vor dem Gegenspieler alle gegnerischen Schiffe zu versenken. Wenn dies einem Spieler gelingt, erscheint folgendes auf der Shell:

„Game Over, (Name des Spielers) hat gewonnen“

3 Funktionen

3.1 Spielename Eingabe - input

In dieser Funktion gibt der Spieler seinen Namen ein und bestätigt diesen mit einem Enterklick. Der Name dient dazu den jeweiligen Spieler direkt anzusprechen, um bei einer Pause beispielsweise zu erkennen, wer am Zug ist.

Code:

```
19  Name1 = input("Wie heißt Spieler1?")
20  Name2 = input("Wie heißt Spieler2?")
```

Mithilfe der Funktion *„Input“* geben die Spieler jeweils Ihre Namen über die Tastatur ein. Diese sind in den Variablen Name1 und Name2 gespeichert und dienen im Verlauf des Spiels der Orientierung.

3.2 Zuweisung - .append

Die Funktion *.append* ermöglicht es, einer Liste einen Wert anzuhängen. Dadurch haben die Spieler jeweils einen Namen, ein Board, eine Möglichkeit zum Angreifen und das aktuelle Board mit gesetzten Schiffen zugeordnet bekommen.

Code:

```
Player1.append(Name1)
Player1.append(Board1)
Player1.append(Guess1)
Player1.append(Gone1)
Player1.append(ships)
```

3.3 Boardausgabe - def print_grid

Diese Funktion dient, der spielgerechten Darstellung des Boards mit der Spalten- und Zeilenbeschreibung.

Code:

```
59  def print_grid(Board):
60      # Buchstaben Notation als Spalte
61      sys.stdout.write("  ")
62      for i in range(spielfeldgröße):
63          sys.stdout.write(" " + chr(i + 65))
64      print(" ")
```

Diese Funktion gibt das aktuellste Board aus. Es sollte so programmiert sein, dass das Board eines jeden Spielers jeweils aktualisiert ist. Man braucht demnach ein neues bzw. überschriebenes Board, da sonst der Gegen-/ Spieler nicht auf dem aktuellsten Stand ist.

Hier ist definiert, welche Zahlen und Buchstaben, wo auf dem Board gesetzt sind. Diese Funktion hält das Grundgerüst des Boards fest.

3.4 Schiffplatzierung – def placeship

Mit dieser Funktion ist es möglich Schiffe zu setzen. Es sind vorhanden:

1x Schlachtschiff - Größe/Länge: 5
2x Kreuzer - Größe/Länge: 4
3x Zerstörer - Größe/Länge: 3
4x U-Boot - Größe/Länge: 2

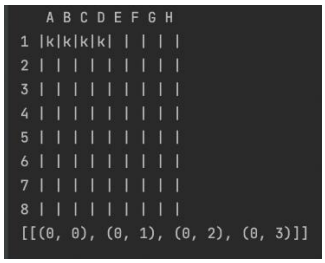
Zunächst setzt der erste Spieler seine Schiffe und bestätigt dies mit Enter. Die Abfrage läuft wie folgt ab:

Das zunächst leere Board erscheint auf dem Display und die Abfrage findet statt (siehe 2.3)

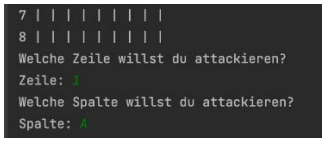
Es ist darauf zu achten, dass die Eingabe korrekt ist. Beim Versuch einer fehlerhaften Eingabe erscheint eine Exception Ausgabe, wie zum Beispiel: *„Falsche Eingabe“* oder *„Unzulässige Eingabe“*. Das Programm fordert den Spieler zu einer erneuten Eingabe auf.

Kontrolle:

Die Kontrolldurchführung passiert folgendermaßen: Das *„Board3“* dient der Platzierung von Schiffen. Die Liste *„safe“* speichert die Koordinaten der Schiffsteile. Es folgt eine Überprüfung der Koordinaten, ob diese bereits in der Liste „Gone“ vorhanden sind, denn „Gone“ enthält die Koordinaten der Schiffe, die schon platziert sind. Falls einer der Koordinaten, mit denen in der Liste „Gone“ übereinstimmt, bedeutet das, dass die Schiffe sich überlappen. Darauf folgt die Meldung, dass man dieses Schiff nicht platzieren kann und eine neue Eingabe durchführen muss.



Nach einer Platzierung eines Schiffes, speichert *„Gone“* die Koordinaten der Schiffsteile als Liste in Form von Tupeln. Das wird so lange fortgeführt, bis alle 10 Schiffe platziert sind. *„Gone“* hat demnach 10 Listen als Elemente, mit den jeweiligen Koordinaten der Schiffsteile, aus denen ein Schiff zusammengesetzt ist.



Ist ein Schiffsteil getroffen, entfernt die Liste *„Gone“* das entsprechende Tupel. Die Spieler greifen sich so lange gegenseitig an, bis alle 10 Listen von *„Gone“* leer sind. Sobald eine Liste leer ist, erzeugt das Programm eine Meldung, dass ein Schiff gesunken ist.

Beispiel mit einzelnen Schritten: Alle Koordinaten eines Schlachtschiffs sind vom gegnerischen Spieler erfasst → Liste ist leer → Ausgabe: *„Schlachtschiff zerstört!“*.

3.5 Angreifen def attack

Es ist von äußerster Bedeutung, dass es in dem Spiel auch eine Funktion zum Angreifen gibt, da man ohne diese Funktion das Spiel nicht beenden könnte.

Die Funktion *def attack* dient dem Angriff, auf gegnerische Schiffe. Das angegriffene Feld, ist auf einem separaten Board dargestellt. Für den Fall, dass der Schuss ein Treffer ist und ein Teil oder das ganze Schiff mit diesem Angriff versenkt, dann zeigt das Programm dies explizit an und informiert so den Spieler über eine entsprechende Ausgabe.

Es gibt 5 verschiedene Mögliche Ausgaben, die auftauchen können:

1. *„Schlachtschiff zerstört!“*
2. *„Kreuzer zerstört!“*
3. *„Zerstörer zerstört!“*
4. *„U-Boot zerstört!”*
5. *„Bis jetzt keine Schiffe zerstört.“*

Die letzte Ausgabe wird nur solange erzeugt, bis ein gegnerisches Schiff zerstört ist. Gekennzeichnet sind Treffer auf gegnerische Schiffe mit einem „x“. Damit der Spieler nicht weiß, um was für ein Schiffstyp es sich handelt, sieht man nicht, anders als bei der Platzierung, den ersten Buchstaben des jeweiligen Typs, sondern nur ein „x“.

3.6 def clearscreen

Diese Funktion dient der Übersichtlichkeit und der Fairness des Spiels. Sie ermöglicht einen das Spielen an einem einzigen Terminal. Die Funktion generiert nach jedem Zug ein Zeilenabstand von 50 Zeilen. Somit ist die Einsicht in den Spielstand des Gegners ausgeschlossen.

4 Fehleingaben und Exceptions

Es kann zu Komplikationen, zwischen den Benutzern und dem Programm kommen. Deshalb ist es sinnvoll, Exceptions mit einzubeziehen, da diese genau dieses Problem mit Hinweisen und Eingabeaufforderungen lösen.

Es handelt sich hierbei um „Ausnahmebedingungen“, die durch falsche Eingaben entstehen. Beispielsweise ist eine unzulässige Eingabe bei der Auswahl einer Zeile zur Platzierung eines Schiffes eine sog. „Ausnahmebedingung“, wenn keine Ziffer eingegeben wird. Entsprechend zur Situation folgen Hinweise und Aufforderungen.

Hier einige Beispiele:

- ➔ Platzierung -> Zeilenangabe -> except ValueError: *„Falsche Eingabe! Bitte nochmal eingeben“*
- ➔ Platzierung -> Zeilen-/Spaltenangabe -> except Index Error: *„Kein Platz! Spielfeld wurde überquert. Nochmal versuchen! “*

An diesen und vielen anderen Beispielen ist gezeigt, dass eine gegebene Hilfestellung bei Falscheingaben nützlich ist und dadurch das Spiel nicht durch eine Fehlermeldung abbricht und endet.

5 Timer / Probleme

Einer der optionalen Anforderungen ist die Implementierung eines Timers. Diese Anforderung ist in dieser Arbeit nicht vollständig eingebunden.

Der Timer dient der Limitierung der Eingabezeit für den Angriff auf das gegnerische Feld und ist auf 15 Sekunden festgelegt. Nach Ablauf der 15 Sekunden, werden zwei Werte per Zufall aus einer zugehörigen Liste ausgewählt, wobei der erste Wert für die Zeile mit dem Wertebereich von beispielsweise: 1 bis 12 und der zweite Wert für die Spalte mit einem Wertebereich von beispielsweise: A bis L, abhängig von der Größe des Spielfeldes. Idee ist es den Timer in das Programm zu integrieren und nach Ablauf der Zeit einen Zufallsschuss in das “fertige Spiel” zu generieren. Dies ist nicht gelungen. Es gab diverse Komplikationen, die die Funktionsweise des Spieles zu Nichte machten. Das Programm ist nicht darauf ausgelegt, einen Timer im Nachhinein einzufügen. Das Spiel setzt sich aus fertigen Programmteilen bzw. Codefragmenten zusammen, die am Ende des Projekts zusammengesteckt wurden.

6 Interprozesskommunikation / Probleme

Der Versuch die Interprozesskommunikation vollständig zu implementieren ist gescheitert. Es ist möglich das Spiel fehlerfrei zu spielen, jedoch ist dieses nicht in einzelne Prozesse geteilt. Das Spiel besteht aus einem einzelnen, nicht parallellaufenden Prozess.

Die ersten Schritte zur Erstellung des Spiels waren die Codierung einzelner Fragmente, wie die Erstellung der Spieler, des Boards, das Setzen und Attackieren der Schiffe. Diese sollten mit der Integrierung von Zwei Prozessen, abhängig von den Spielern, zusammengesetzt werden und so das fertige Spiel ergeben. Die gewählte Vorgehensweise, stellte sich als ungünstig heraus, da das Implementieren der Prozesse als einer der letzten Schritte behandelt und die Integration erschwert wurde. Woran es dann letztendlich gescheitert ist.

Alternativ hat man versucht, die optionale Anforderung des Timers, als Prozess zu implementieren, welche allerdings aufgrund der fehlgeschlagenen Synchronisation gescheitert ist (siehe 5 Timer/Probleme)

7 Zusammenfassung

In diesem Projekt wurde das Spiel Schiffe versenken programmiert, welches einwandfrei zu spielen ist. Die größten Probleme haben sich bei der Interprozesskommunikation und dem Timer ergeben.

Trotz der fehlenden Prozessimplementierung und dem gescheiterten Timer ist das Spiel zu zweit spielbar.

8 Fazit

Trotz aller Schwierigkeiten bietet dieses Projekt, eine gute Grundlage, um eine neue Programmiersprache zu lernen. Die Umsetzung aller Anforderungen ist nicht gelungen, wie beispielsweise die der Interprozesskommunikation.

Eine Integration des Timers ist auch nach Fertigstellung des Programms möglich, sowie die Erstellung von eigenständigen Prozessen für jeden Spieler, jedoch hat hier die Zeit zur Implementierung und zur Recherche, wie man es machen sollte, sowie externer Input gefehlt.

9 Quellenangabe / Literatur

https://stackoverflow.com

https://realpython.com

https://www.geeksforgeeks.org

https://www.python-lernen.de

https://programmieren-starten.de

https://www.delftstack.com

https://de.wikipedia.org/wiki/Schiffe_versenken