Schiffe-Versenken mit Interprozesskommunikation

Namen und Matrikelnummern:
Emine Metin 1307870
Mohammed Matran 1402704
Melek Piskin 1307294
Quoc Cuong Ha 1405659
Selin Kaya 1390955

Studiengang Wirtschaftsinformatik
Betriebssysteme & Rechnernetze

Frankfurt University of Applied Sciences
Nibelungenplatz 1
60318 Frankfurt am Main

Inhaltsverzeichnis

1. Einleitung

Die folgende Dokumentation zeigt die Entwicklung, als auch die Implementierung unseres Projektes, für welches wir uns entschieden haben.
Wir als Gruppe haben uns für die Alternative 7 entschieden „Schiffe–Versenken mit Interprozesskommunikation“. Alle Teammitglieder waren der gleichen Meinung, da es sich sehr interessant angehört hat. Wir hatten die Möglichkeit eine Programmiersprache auszusuchen, um diese dann beim Programmieren zu nutzen. Im ersten Schritt haben wir uns über das Spiel als einzelner informiert, um das Spiel besser verstehen zu können und uns dann auch auf die Programmiersprache Python geeinigt.
Anschließend haben wir uns die Aufgabenstellung durchgelesen und eine erste Aufgabenteilung veranlasst, die wir im nächsten Meeting dann besprochen haben. Wir haben uns versucht wöchentlich einmal zu treffen, um alles besprechen zu können.

2. Das Spiel

2.1 Vorbereitung

Jeder mitspieler platziert sein Schiff auf einem 10x10 großem Feld ohne das der Gegenspieler weiß wo man es hin platziert hat.

2.2 Regeln

1. Das Schiff darf nicht über den Feldrand hinausgehen
2. Das Schiff darf nicht Diagonal liegen.
3. Jeder verfügt über ein Schiff
4. Es wird ausgelost wer zuerst anfängt
5. Der Gewinner steht fest wenn er als erstes das Schiff vom Gegner versunken hat

2.3 Spielverlauf

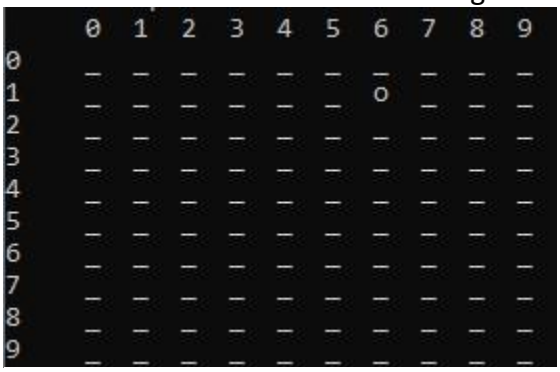
Nachdem ausgelost wurde wer anfängt beginnt der Spieler in dem er eine Koordinate sagt z.B 4 5(Hierbei gibt die erste Zahl die Vertikale Koordinate an und die zweite Zahl die Horizontale Koordinate auf die geschossen wird). Daraufhin sagt der Gegenspieler ob es ein Treffer oder keiner war. Danach ist der andere Spieler dran und es wird immer nach jedem Schuss gewechselt. Versenkt man das Schiff des anderen zuerst gewinnt man das Spiel.

2.4 Und so funktioniert das Spiel in unserem Programm

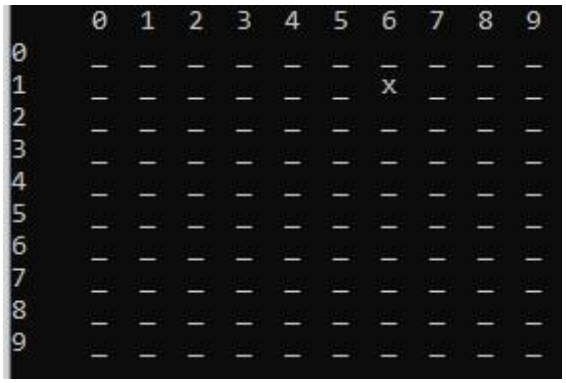
Startet man das Programm fragt das Spiel einen ab ob man spielen möchte, wenn man das Spiel spielen will gibt man eine “1” ein.

Als nächstes wird gefragt ob man gegen einen anderen Spieler spielen möchte, falls ja gibt man wieder die “ 1 “ an.
Danach geben die Spieler ihre Namen ein mit denen sie spielen wollen.
Das Spiel befindet sich auf einem 10x10 großes Feld und jeder Spieler hat ein Boot das 3 Felder groß ist.

Um einen Schuss zu machen gibt man das Feld an auf dem man schießen will (z.B 16, die 16 wird hierbei in 1 und 6 aufgeteilt und die erste Zahl gibt Vertikale Achse an und die zweite Zahl die Horizontale Achse auf dem geschossen wird)



so würde z.B ein Treffer aussehen.



Und so wenn man keinen Treffer landet.

Das Spiel endet sobald das Schiff vom Gegner zuerst versenkt wurde, daraufhin wird das Spiel neugestartet.

3. Implementierung

Funktionen:

Dieses Programm weist eine Klasse und mehrere Funktionen (Methoden) auf, die unterschiedliche Aufgaben im Programm haben. Um die Klasse und die Methoden im Programm zu verstehen, muss zuerst die beigefügte Dokumentation deutlicher beschrieben und verstanden werden.

3.1. Klasse:

class Player():

Mit Hilfe der Objektorientierten Programmierung (OOP) versuchen wir komplexe Objekte aus der realen Welt innerhalb unseren Programms nach zu modellieren, um die Technik innerhalb unseren Programms umsetzen zu können. In diesem Fall ist es unser Programm, in welchem ein Spiel programmiert und gespielt werden kann. Hier macht es möglicherweise Sinn für das Spiel einen eigenen Datentyp zu kreieren. Wenn OOP in Python eingesetzt wird, dann benötigen wir einen Datentypen, indem eine so genannte Klasse definiert wird. Eine Klasse ist wiederum nichts anderes als der Bauplan, in welchem das Prinzip vorgegeben wird, wie das Spiel aussehen soll und welche Funktionen es beinhalten soll. Dafür definieren wir zuerst im ersten Schritt eine Klasse für unser Spiel,

in dem wir zunächst das Schlüsselwort class nutzen, darauf folgt der Name unserer Klasse. In unserem Programm nannten wir diese Player.

3.2. Definition der Klasse:

```
def __init__(self, name):
```

Alles was hier jetzt eingerückt unter der Klasse geschrieben wird, gehört zu der Definition der Klasse. Auf jeden Fall wird geschrieben, was innerhalb dieser Klasse programmiert wird. Der Sinn von so einer Klasse ist, dass wir hier die ganzen Eigenschaften der Spieler modellieren können. Diese Eigenschaften, die in der Programmiersprache als Attribute bezeichnet werden, können nun in der Klasse festgelegt werden. Bevor dies ermöglicht werden kann, benötigen wir innerhalb der Klasse Definition nun eine ganz besondere Funktion, nämlich die sogenannte init Funktion. Am Anfang wird das Schlüsselwort def geschrieben, weil hier eine Funktion definiert wird. Danach muss die init Funktion exakt so __init__ geschrieben werden, damit es funktioniert. Zwischen den Klammern schreiben wir self als Parameter. self hat den Job die Referenz des ausgeführten Objektes zurück zu liefern. Sobald sich die Programmausführung innerhalb der sogenannten init Methode befindet, auf welches konkrete Objekt es aktuell zugreift, damit hilft es den Parameter self zu schreiben. Der Parameter name bezieht sich auf die Variable self.name.

3.3. Attribute:

Nach der Definition dieser speziellen init Funktion wird die Attribute innerhalb des Funktionskörpers definiert, die das Spiel innerhalb unseres Programms charakterisieren soll. Dafür wird self mit einem gefolgten Punkt geschrieben, dann kommt der Name des Attributs, welches wir definieren möchten. Die Attributen haben wir hit, miss und comp genannt und als Werte dieser Attributen haben wir eckige Klammern [] verwendet, mit diesen Klammern können wir unterschiedliche Datentypen in beliebiger Reihenfolge aufnehmen und diese listen. Listen werden in Python durch eckige Klammern gekennzeichnet. Der Attribut mit dem Namen guesses hat den Wert aller anderen Attributen, da es die Liste ist, in dem alle gemachten Schüsse und der Ablauf stehen.

Als weiteren Attribute haben wir die Schiffe(boat1, boat2). Sie werden schon festgelegt, bevor das Spiel anfängt.

3.4. Schleifen, Klauseln, Anweisungen und Funktionen:

while-Schleifen werden genutzt, wenn Abschnitte in einem Programm so lange ausgeführt werden sollen, bis eine bestimmte Bedingung eintrifft.
Eine try-Klausel kann eine beliebige Anzahl von except-Klauseln zur Behandlung verschiedener Ausnahmen, es wird jedoch nur eine ausgeführt, wenn eine Ausnahme auftritt.
Die if-Anweisung wird genutzt, um eine Bedingung abzuprüfen. Falls diese Bedingung erfüllt ist, wird ein Block von Anweisungen ausgeführt (der so genannte if-Block), andernfalls wird ein anderer Block von Anweisungen ausgeführt (der so genannte else-Block). Die else-Klausel ist dabei optional.
Die for-Schleife dient zur Iteration über eine Sequenz von Objekten.
Die range-Funktion erzeugt eine Liste an Zahlen, die oft im Zusammenhang mit for-schleifen verwendet werden.

3.5. def geh_shot(guesses):

Das ist eine andere Definition, die den Schüssen bzw. das Attribut(guesses) dient. Diese Definition wird durch while-Schleife, if-Anweisung und (try, except) funktioniert. Durch die while-Schleife werden die Spieler den Platz der Schiffe erraten, bis das Spiel beendet wird. Durch eine Kombination

zwischen der try und die if-Anweisung dürfen die Spieler auf den gewählten Platz schießen, unter der Bedingung, dass die gewählte Zahl zwischen 1 und 99 ist. Sonst wird eine Fehleranmeldung durch except gezeigt.

3.6. def show_board(hit, miss, comp):

Mit dieser Definition wird die Tafel gezeigt und durchgeführt, es gehört ebenso zu den Attributen (hit, miss, comp) dazu. Diese Definition wird durch die for-Schleife, if-Anweisung und range-Funktion funktioniert. Zuerst wird durch print der Spielname und die erste Reihe von der Tafel angezeigt. Mit der Hilfe von for-Schleife und range-Funktion wird die ganze Tafel erstellt. Die range ist in der Größe von 10x10. Darauffolgend wird anhand der if-Anweisung an der Tafel entweder hit " o " oder miss " x " angezeigt.

3.7. def check_shot(shoot, boat1, boat2, hit, miss, comp):

Hier wird überprüft, ob der Schuss hit, miss oder comp aufgetreten ist. Dafür wird die if-Anweisung genutzt. Wenn shoot den boat1 oder boat2 trifft, dann wird dieser Platz von der Tafel gelöscht oder umgekehrt. Wenn der Schuss die Tafel nicht trifft, wird der Platz von der Tafel dennoch gelöscht.

3.8. Das Spiel:

Das Spiel wird anhand mehrerer while-Schleifen durchgeführt, damit das Spiel bei jeder Anforderung als loop weiterläuft. Durch input werden die Anforderungen an die Spieler auf dem Spielerbildschirm angezeigt, um das Spiel zu beginnen. Am Anfang muss wegen der if-Anweisung der Spieler die Zahl "1" tippen, sonst wird eine Fehlermeldung angekündigt. Mit "1" wird das Spiel begonnen. Die beiden Spieler können zunächst ihren Namen eingeben, alternativ werden die Namen der Spieler als (default) also Spieler1 und Spieler2 angewiesen. Durch print und counter wissen die Spieler, wer dran ist, wenn counter eine ungerade Zahl ist dann ist Spieler 2 dran, wenn nicht, ist Spieler1 dran. Ab jetzt hängt das laufende Spiel bzw. dieses Ereignis von den eingegebenen Daten der Spieler ab. Am Ende wird der Spieler, dessen Schiffe auf null sind, durch die if-Anweisung und die len-Funktion gewinnen.

4. Probleme/Ergebnis

Dieses Programm erfordert Kenntnisse über die wichtigsten Klassen, Schleifen und Funktionen. Diese müssen zuerst gelernt und verstanden werden. Da wir die **class Player():** Klasse nicht erstellt hatten, hatten wir auch Probleme mit den dazugehörigen Codes. Wir mussten diese zunächst Nutzen, da diese Klasse angibt, welche Funktionen das Spiel hat.

Ein weiteres Problem, welches wir hatten, war den Ausdruck **def** nicht vor den Funktionen hinzuzufügen. Da wir also die Funktion, die wir geschrieben haben, nicht definieren konnten, fing das Programm an, Fehlermeldungen anzuzeigen, weil es die Funktionen und den Code, den wir geschrieben haben, nicht lesen konnte. Unser Programm funktionierte nicht, weil wir es vor vielen Funktionen nicht benutzt hatten, und dann konnten wir dieses Problem lösen, indem wir diese Funktionen identifizierten.

Zuerst haben wir die **while-Schleife**, die wir verwenden mussten, um das Spiel innerhalb eines bestimmten Zyklus fortzusetzen, nicht verwendet, sodass es ein einseitiges oder sogar ein unspielbares Spiel wurde. Da der Code, den wir geschrieben haben, nicht in einem

bestimmten Zyklus war. Man konnte nicht mit einer einzigen Arbeit spielen. Sobald alle unsere Codes vollständig waren, lief unser Spiel reibungslos und wurde spielbar.

Wir hatten ein weiteres Problem bei der Einführung des **Interprozesskommunikations,** da wir nicht wussten wie wir diese in unseren Codes einbringen sollten. Die Interprozesskommunikation dient dazu, mehrere Prozesse miteinander zu verknüpfen. Interprozesskommunikation ist ein sehr umfassendes Thema und enthält viele Funktionen und Methoden. Da wir nicht wussten, welchen Teil dieses Themas wir verwenden sollten, konnten wir es nicht in unser eigenes Programm aufnehmen, also konnten wir dieses Problem nicht lösen. Wir haben ein paar Ideen entwickelt, wie wir es integrieren könnten, dennoch ist es uns nicht gelungen, es in unser Programm zu nutzen. Einige unserer Ideen waren:

1. In der Interprozesskommunikation ging es um die **Ausführungsreihenfolge**. Prozess B wartet, wenn das Signal von Prozess A kommt, und dann läuft B mit den Informationen von Prozess A. Wir haben versucht, es für zwei Spieler in unserem Spiel zu verwenden. Entsprechend der Antwort von A bewegt sich B und so entsteht eine Schleife.
Wir dachten, wir könnten es mit **set-** und **reset-Codes** machen, aber es gelang uns nicht, also benutzten wir alternativ die Schleife.

2. **Aktives und passives Warten**. Wir wollten schreiben, dass ein Spieler auf den anderen Spieler wartet, während er spielt, und umgekehrt, mit aktives und passives Warten, wir haben es aber nicht geschafft. Wir hätten die Funktionen **sigsuspend, pause** und **sleep** verwenden können, aber das konnten wir nicht.

5. Fazit / Ausblick

Am Anfang hatten wir ein bisschen Probleme mit dem Verständnis. Es fiel uns zunächst schwer in das Thema reinzukommen. Sobald wir uns an die Arbeit gemacht haben, wurde es jedoch immer verständlicher. Wir haben uns mit der Programmiersprache Python auseinandergesetzt. Unseren Quellcode haben wir zunächst grob geschrieben und haben uns währenddessen versucht über die Interprozesskommunikation zu erkundigen. Das coden war zunächst machbar, wir hatten jedoch bei der Implementierung einige Probleme. Es traten einige Fehlermeldungen auf, die wir im Nachhinein verbessern mussten. Einige Fehlermeldungen haben uns zum Nachdenken gebracht, jedoch waren es keine, die wir nicht beheben konnten. Die Interprozesskommunikation war für uns eine sehr große Herausforderung. Wir wussten für was dieses genutzt wird, aufgrund der Informationen, die wir gesammelt haben, hatten jedoch sehr große Probleme bei der Umsetzung. Wir haben für die Lösung dieses Problems bei anderen Gruppen angefragt. Diese haben uns jedoch alle was anderes gesagt, womit wir rein gar nichts anfangen konnten. Die Gruppenarbeit war sehr angenehm und wir konnten uns aufeinander verlassen. Wir haben uns die Aufgaben regelrecht aufgeteilt. Jeder hat seinen Teil bearbeitet und ist bei Fragen auf die Gruppe zugegangen. Am Ende haben wir uns alles angeguckt und uns gegenseitig ergänzt.

6. Literatur

- Foliensatz 6 der Vorlesung Betriebssysteme und Rechnernetze im SS2022