

Alternative 7

Schiffeversenken

Prof. Dr. Christian Baun

Frankfurt University of Applied Sciences
(1971-2022: Fachhochschule Frankfurt am Main)
Nibelungenplatz 1
60318 Frankfurt am Main
christianbaun@fb2.fra-uas.de

Ilias Baroudi
Philip Wefers
Daniel Vonhof
Ben Storjohann

26.06.2022

Abstrakt

Die Anforderung der Alternative 7 beinhalteten den Klassiker "Schiffeversenken" zu programmieren, welches mit einer Interprozesskommunikation mehreren Spielern¹ erlaubt miteinander zu spielen. Folgendes Dokument befasst sich nicht nur mit dem kompletten Code der Spielumgebung selbst, sondern auch mit der graphischen Darstellung für die Spieler selbst.

Zuerst geht es um das Erstellen von Spielern, Schiffen und des Spielfelds und um den Spielverlauf. Daraufhin wird darauf eingegangen, wie das Spielfeld, aber auch ein Spielzug, dargestellt wird. Dieser Code befindet sich im Programm in der Main.py.

Als letztes geht es um die IPC und wie diese verwirklicht wurde.

Spielaufbau

Als erstes haben wir uns Gedanken gemacht, wie wir das Spiel konzipieren möchten. Das klassische zehn mal zehn Spielfeld hat uns am Besten gefallen und haben uns damit für folgende Schiffe entschieden: Schlachtschiff (fünf Kästchen), Kreuzer (je vier Kästchen), Zerstörer (je drei Kästchen), U-Boot (je zwei Kästchen). Wir haben uns auch dafür entschieden die Schiffe zufällig setzen zu lassen.

Uns war von Beginn an wichtig, dass die Schiffe zufällig gesetzt werden und die Spieler diese nicht selbst setzen können, damit wir verhindern, dass Schiffe ständig gleichgesetzt werden, oder aktiv verbotene und unangebrachte Zeichen gesetzt werden können.

Um das zufällige Setzen durch manuelles zu ersetzen, müsste der Spieler zu Beginn aufgefordert werden, diese Variablen entweder durch Klicken auf dem Spielfeld oder Eingabe mit der Tastatur manuell vorzugeben.

Wir lassen das Programm den Startpunkt zufällig auswählen mit einer random.choice, sowohl für die x-Achse als auch die y-Achse. Zudem benötigten wir eine weitere random.choice um die Ausrichtung der Schiffe zufällig bestimmen zu lassen. Zudem wird die Länge des Schiffs in der Variablen „Size“ gespeichert.

Um die Indexe (die einzelnen Punkte des Schiffes auf dem Spielfeld) anhand des Startpunktes und der Ausrichtung zu berechnen, nutzen wir eine Hilfsfunktion „indexcalculation“.

Wenn die Ausrichtung horizontal sein sollte, addieren wir zum Startpunkt einfach die Länge des Schiffs. Damit sind die Indexe des Schiffs von links nach rechts aufbauend gesetzt.

Für die vertikale Ausrichtung multiplizieren wir den Startpunkt jeweils mit 10 (respektive 6 für die kleine Variante), da eine Zeile 10 (6-small) Felder enthält. Hiermit ist der jeweils folgende Index genau unter dem vorherigen.²

Spieler:

Hierfür wurde eine Spieler Klasse erstellt. Folgende Eigenschaften muss ein Spieler haben:

- Eigene Schiffe
- Schiffe platzieren
- Gesetzte Schiffe speichern
- Auf andere Schiffe schießen können

¹ Der Einfachheit halber haben wir uns entschieden für die Spieler einheitlich die männliche Form zu verwenden.

² Siehe Anhang 1

- Erhaltene Treffer und Fehlschüsse speichern

Für die Spieler wurde eine Klasse angelegt. Die Spieler benötigen einige Listen zum Speichern von Schiffen, deren Indexes und das eigene Spielfeld.

In der `self.ships` werden die platzierten Schiffe gespeichert. In `self.search` wird das spielereigene Feld gespeichert (mit „Grid“ als Standard für jedes Feld) und zu einem späteren Zeitpunkt auch die Schüsse („Hit“, „Miss“ oder „Sunk“) des jeweiligen Gegners. Die Schüsse überschreiben den zu Beginn gesetzten Standard „Grid“.

Damit wir in der Player Klasse direkten Zugriff auf die Schiffe aus der Ship Klasse haben legen wir eine Liste `listofships` an. In der Liste sind die Schiffe in Unterlisten gespeichert:

```
[ [63, 64, 65, 66, 67], [0, 10, 20, 30], [41, 42, 43], [13, 23, 33] [57, 58] ]
```

Eine `listofships` kann dann beispielsweise wie hier gezeigt aussehen.

Um die `listofships` als einfache Liste und als Eigenschaft von der Player Klasse zu speichern, wurde `self.indexes` geschrieben. Hier werden die Unterlisten (Sublists) quasi entpackt und als einfache Liste gespeichert:

```
[63, 64, 65, 66, 67, 0, 10, 20, 30, 41, 42, 43, 13, 23, 33, 57, 58]
```

Die „entpackte“ Liste sieht dann wie hier gezeigt aus. Die Liste lässt das Programm einfacher vergleichen ob wir ein Schiff getroffen haben oder ob sich Schiff indexe kreuzen und die Schiffe so aufeinander liegen würden.

Ein Problem ist aufgetreten beim zufälligen Setzen der Schiffe. Beispielsweise der deutlichste Fall, wenn die Start Reihe und Spalte jeweils mit 9 ausgegeben wurde. Jetzt ragt das Schiff, bis auf den Startindex komplett über das Spielfeld heraus. Ein ähnliches Problem ist aufgetreten, wenn Schiffe länger sind als die Zeile bspw.. Hier ist das Schiff zu einer Hälfte in der Anfangszeile bis zum Ende des Spielfeldes und der Rest taucht in der Zeile darunter auf.

Um dies zu verhindern, mussten wir Regeln aufstellen, um ein Ship placement zu verhindern, sobald Unregelmäßigkeiten auftreten. Hierfür wurde die Methode „`placeships`“ geschrieben.

In dieser Methode prüfen wir zunächst, ob jeder Index des Schiffes sich innerhalb des Spielfeldes befindet. Das heißt, dass jeder Teil des Schiffes einer Zahl kleiner als 100 (respektive 36 beim kleinen Feld) zugeordnet ist. Sollte dies der Fall sein, prüfen wir ob alle Indexe des Schiffes mit dem Startindex der Zeile oder Spalte identisch sind. Sollten die Indexe weder mit der Zeile noch mit der Spalte übereinstimmen, ist das Schiff nicht in einer Linie gesetzt und das Schiff kann nicht gesetzt werden.

Der letzte Test, der bestanden werden muss ist die Schiffskollision. Hier prüfen wir, ob das zu setzende Schiff irgendeinen Index gemein hat mit einem bereits gesetzten Schiff. Sollte dies nun nicht der Fall sein, sind alle 3 Checks bestanden und das Schiff kann gesetzt werden und der Liste „`ships`“ hinzugefügt werden.

Wir hätten auch andersrum prüfen können. Also alle Fälle aufzählen, in welchen das Schiff richtig platziert worden wäre, aber dies ist viel aufwändiger. So hätten wir geprüft, ob das Schiff komplett in der ersten Zeile liegt, oder in der 2. Zeile oder in der x-ten Zeile, oder ob es in der y-ten Spalte liegt. Dies bedeutet ein erheblich längerer Code. Und dies gilt nur für den 2. Check. Die anderen beiden Checks hätten wir zudem auch noch schreiben müssen. Deshalb haben wir uns für die 3 Tests entschieden, für die ein Schiff Placement nicht korrekt ist.³

³ Siehe Anhang 2

Spielverlauf

Der Code für den Spielverlauf bestimmt die Möglichkeiten eines Spielzuges und somit auch die Spielregeln des Spiels.

```
class Game:
def __init__(self):
    self.player1 = Player() |
    self.player2 = Player()

    self.player1turn = True
    [...]
def playerturn(self, i):
    player = self.player1 if
self.player1turn else self.player2
    inactiveplayer = self.player2 if
self.player1turn else self.player1
    hit = False
```

Nach dem Erstellen von zwei Spielern definieren wir einen Spielzug.

Hier hätte sich auch eine IPC als Basis für unser Mehrspielersystem angeboten.⁴

In der letzten Zeile wird hit false gesetzt, um sicherzustellen, dass bei einem Fehlschlag der nächste Spieler dran kommt.

Innerhalb eines Zuges prüfen wir, ob der Spieler am Zug ein Schiff getroffen oder sogar versunken hat. Sonst gehen wir davon aus, dass ein Spieler keinen Treffer erzielen konnte und der nächste Spieler ist am Zug.

```
if i in inactiveplayer.indexes:
    player.search[i] = "Hit"
    hit = True
```

Genauer gesagt, prüfen wir, ob der Spieler am Zug ein Feld mit einem gegnerischen Schiff auswählt und damit einen Treffer erzielt.

```
for ship in
inactiveplayer.ships:
    sunk = True
    for i in ship.indexes:
        if player.search[i]
== "Grid":
            sunk = False
            break
```

Daraufhin prüfen wir, ob das Schiff gesunken ist. Ein Schiff sinkt, wenn alle Felder, auf denen es sich befand, getroffen worden sind. Daher prüfen wir, ob besetzte Felder des getroffenen Schiffs noch nicht ausgewählt worden sind ("Grid"). Ist dies der Fall, ist das Schiff noch nicht gesunken.

```
if sunk:
    for i in
ship.indexes:
        player.
search[i] = "Sunk"
else:
    player.search[i] = "Miss"
```

Falls das Schiff gesunken ist wird dies im Index gespeichert (und dem Spieler später angezeigt).

In allen anderen Fällen hat der Spieler verfehlt und der nächste ist am Zug.

Doch wie wird ein solcher Spielzug einem Spieler angezeigt? Die Benutzeroberfläche ist in der Main gespeichert, die sich mit der Animation des Spiels befasst.

⁴ Eine genauere Erläuterung und unseren Ansatz finden Sie unter "Beispiel Umsetzung IPC"

Graphische Darstellung

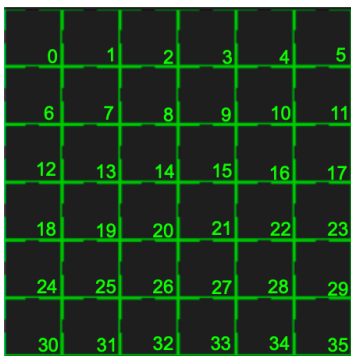
Zur grafischen Darstellung des Grids, in welchem unsere Schiffe und Treffer angezeigt werden, haben wir mit der Methode „gridcreation“ gearbeitet.

```
for i in range(36):  
    x = left + i % 6 * Square  
    y = top + i // 6 * Square
```

In dieser Methode wird durch die Range der gesamten Felder durchgegangen.

Dabei wird der Startpunkt in die Methode übergeben, wenn nicht, beginnt die Zeichnung des Grids bei 0/0 (Linke obere Ecke). Nun werden die Einzelnen Felder mit der Variable Square erstellt. Als Beispiel beim kleinen Feld:

Berechnung für Feld „5“:



0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35

$$X = 0 + 5 \% 6 * 35 = 175$$

$$Y = 0 + 5 // 6 * 35 = 0 \text{ (Weil Int)}$$

Feld 5 beginnt demnach bei dem 5-fachen Wert für ein Square auf der X-Achse und bei 0 auf der Y-Achse.

Berechnung für Feld „32“:

$$X = 0 + 32 \% 6 * 35 = 70$$

$$Y = 0 + 32 // 6 * 35 = 175 \text{ (Weil Int)}$$

Feld 32 beginnt demnach bei dem 2-fachen Wert für ein Square auf der X-Achse und bei dem 5-fachen Wert für ein Square auf der Y-Achse.

Für das Grid von Spieler 2 werden die Parameter für left und top mit Werten übergeben, sodass das 2. Grid diagonal unter dem 1. Grid liegt. Somit bietet das Layout genügend Platz für weitere Informationen neben den Grids, um die Spieler mit dem notwendigen zu versorgen wie zum Beispiel „Your Ships“ und einen optionalen Timer.

```
field = pygame.Rect(x, y, Square, Square)  
pygame.draw.rect(Window, Green, field, width=1)
```

In der Variable field werden nun durch den command „pygame.Rect“, die

Squares erstellt und darauffolgend mit Übergabe der Farben, der Strichbreite sowie der Ausgabefläche auf dem Bildschirm ausgegeben.

```
pygame.draw.circle(Window,  
AssignColors[player.search[i]],  
(x, y), radius=Square // 2)
```

Die „Grid-circles“ werden erstellt und auf dem Feld ausgegeben. Diese werden bei erfolgten Schüssen der Spieler eingefärbt. Zu Beginn haben sie jedoch die Farbe des Hintergrunds beziehungsweise des Grids, um ein noch nicht getroffenes Feld darzustellen.

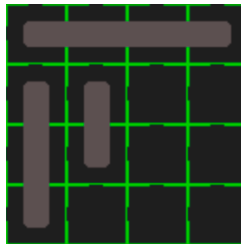
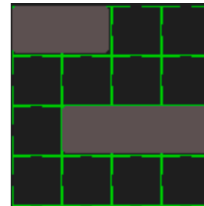
```
for ship in player.ships:  
    x = left + ship.col * Square  
    y = top + ship.row * Square  
    if ship.orientation == "Horizontal":  
        width = ship.size * Square  
        height = Square  
    else:  
        width = Square  
        height = ship.size * Square  
    rectangle = pygame.Rect(x, y,  
width, height)  
pygame.draw.rect(Window, Grey,  
rectangle, border_radius=3)
```

Die Darstellung der Schiffe innerhalb des Grids ist mit der Methode drawships realisiert. Hier wird ebenso wie in der gridcreation-Methode ein Startwert im x,y-Format übergeben.

Für die Darstellung der Schiffe innerhalb der Squares, werden die columns und

die rows von den jeweiligen Schiffen mit der Squaregröße multipliziert, um die Startwerte zu ermitteln. Daraufgehend wird bei Horizontalen Schiffen die Schiffslänge mit der Squaresize multipliziert. Bei Vertikal angeordneten Schiffen wird die Breite mit der Höhe vertauscht.

Aus dem obigen Code ergeben sich nun sehr breite und klobige „Schiffe“.



Um das Programm etwas ansehnlicher zu gestalten verwenden wir die Variable „Smallschips“.

Wird diese jeweils von der Breite und höhe des Schiffes abgezogen, erhalten wir ein Schiff, welches Zentriert dargestellt wird.

Der Spielerwechsel ist mit der Methode switchplayer verknüpft. Diese Methode wird bei einem „Miss“ getriggert. Dadurch ist das Spiel, an einem Gerät, in einem Fenster, spielbar. Hat der aktive Spieler einen Schuss ins Wasser gesetzt, wird das Fenster Schwarz gefüllt und zeigt dem Spieler, mit Text, dass er vorbeigeschossen hat. Nun wird er dazu aufgefordert die Kontrolle zum nächsten Spieler zu geben, der mit der Tastatureingabe das Spiel fortsetzen oder beenden kann. Eine mögliche Meldung, dass der Spieler, der das Spiel vor Spielende abgebrochen hat, automatisch verliert, haben wir uns ebenfalls überlegt. Diese Meldung könnte man in der switchplayer-Methode zusätzlich implementieren.

Der Spielstart wird über die starter.py realisiert.

```
if event.key == pygame.K_s:
    from mainsmall import Gamesmall
    starting = False
    Gamesmall()
if event.key == pygame.K_n:
    from main import Game
    starting = False
    Game()
```

Durch den Startscreen wird der Benutzer gefragt, welches Spiel er starten möchte. Um das Spiel dynamischer zu gestalten haben wir uns dafür entschieden, zwei Varianten zur Auswahl zu stellen. Ein schnelles und ein normales Spiel. Das „normale“ Spiel hat eine Gridgröße von 10x10 Feldern, in dem 5 Schiffe liegen. Das „schnelle“ Spiel kommt mit einer

Gridgröße von 6x6 aus und lediglich 3 Schiffen. Eine weitere Option sowie eine Anpassung der Anzahl der Schiffe ist leicht zu implementieren und kann bei Bedarf jederzeit umgesetzt werden.

```
if event.type == pygame.MOUSEBUTTONDOWN:
    x, y = pygame.mouse.get_pos()
    if not game.over and game.player1turn
    and x < 10 * Square and y < 10 * Square:
        row = y // Square
        col = x // Square
        index = row * 10 + col
        game.player1turn(index)
    if not game.player1turn:
        switchplayer()
```

Sobald der Spieler mit der Maus drückt, wird die Position festgestellt und geprüft, ob der Mausklick sich innerhalb des Spielfelds befindet. Indem wir durch die Feldgröße teilen, finden wir heraus welches Feld (im „Grid“) der Spieler für seinen Zug auswählt. Damit kann dies übergeben und im Spielerindex gespeichert werden.

Falls Spieler 1 nicht am Zug ist wird der Vorgang für Spieler 2 ausgeführt. (Siehe unten Seite 7 unter Beispiel Umsetzung IPC mit TCP-IP sockets)

Falls der Spieler sich entscheiden sollte, das Spiel zu verlassen, gibt es mehrere Möglichkeiten das Spiel zu verlassen.

```
if event.type == pygame.KEYDOWN:
    if event.key == pygame.K_ESCAPE:
        animating = False
```

```
    if event.key == pygame.K_RETURN:
        game = Game()
```

```
if animating:
    Window.fill(Black)
    gridcreation(game.player1)
    gridcreation(game.player2,
left=(Width - Square) // 2 + Square,
top=(Height -
Square) // 2 + Square)
    if game.player1turn:
        drawships(game.player1,
left=(Width - Square) // 2 + Square,
top=(Height - Square)
// 2 + Square)
        textplayer1turn =
"<- Player 1, your turn!"
```

Solange der Spieler das Spiel nicht beendet wird folgende Umgebung animiert. Der Hintergrund ist schwarz und beide Gitter (Grid) werden angezeigt. Je nachdem wer am Zug ist, werden seine Schiffe dargestellt und ein entsprechender Hinweis angegeben.

Falls das Spiel vorbei ist, wird ein entsprechendes Fenster angezeigt, welches den Spieler darüber unterrichtet und ihm die Möglichkeit gibt das Spiel neu zu starten.

Beispiel Umsetzung IPC mit TCP-IP sockets

Class Communication:

Diese Klasse soll mit Hilfe von TCP() Sockets die IPC() implementieren. Hierbei ist Spieler eins der Server und Spieler zwei der Client. Es wichtig zu beachten, dass für eine funktionierende Verbindung zuerst der Server und dann der Client aufgerufen werden.

Ob es sich um Spieler eins oder zwei handelt, wird über den Skriptparameter ermittelt.

z.B.: Bearbeitung von Server und Client Parameter

- <Pythonprogrammpfad>\python "<Pythonskriptpfad>\main.py" server
- <Pythonprogrammpfad>\python "<Pythonskriptpfad>\main.py" client

Import sys

```
if sys.argv[1] == "server":
```

```
    # configure Server Socket
```

```
elif sys.argv[1]=="client":
```

```
    # configure Client Socket
```

Die __init__ Methode initialisiert die Socket Instanzen als Server oder Client mittels eines Parameters und einer if – elif – Abfrage.

Um die Nachrichten zu senden und zu empfangen, wird zwischen Client und Server differenziert.

Es werden für den Server folgende Methoden benutzt:

- `send_message_to_client`
- `get_message_from_client`

und für den Client:

- `send_message_to_server`
- `get_message_from_server`

JSON ist ein schlankes Datenaustauschformat, um Objekte zu enkodieren und dekodieren, welches für einen strukturierten Datenaustausch benutzt werden kann.

Die Funktion `json.dumps()` konvertiert ein Python-Objekt in einen String und mit `json.loads()` wird der String in ein Objekt konvertiert.

Hier wird das Dictionary Object `Dictionary` verwendet.

Beim Beenden des Programms muss die Funktion `Socket.close()` verwendet werden, um Ressourcen frei zu bekommen wie zum Beispiel der Port.

Referenzen:

[Python JSON \(w3schools.com\)](https://www.w3schools.com/python/python_json.asp)

[Python Dictionaries \(w3schools.com\)](https://www.w3schools.com/python/python_dictionaries.asp)

[How to use sys.argv in Python with examples \(knowledgehut.com\)](https://www.knowledgehut.com/python/python-sys-argv-examples/)

Implementierung:

Eine große Herausforderung beim Projekt war es eine Interprozesskommunikation (IPC) zu implementieren. Dies erwies sich als besonders schwierig, weil wir zuerst den Code für das Schiffeversenken Spiel geschrieben haben und uns danach der IPC zuwendeten. Somit standen wir vor der komplexen Aufgabe aus dem zusammenhängenden Code des Spiels zwei Prozesse zu bilden.

Um eine schnelle Implementierung zu realisieren, haben wir in die Klasse `Gamesmall` nur einen Spieler eingefügt und ein `Socket` davon abhängig initialisiert, ob Player 1 bzw. Server oder Player 2 bzw. Client dran ist.⁵

Um die Daten zwischen Spielern auszutauschen, nutzen wir zwei Funktionen von der `Gamesmall` Klasse:

- **`make_movesmall(i)`**

Diese Funktion wird verwendet, wenn der Player dran ist, um die Koordinate des Schusses zu schicken und die daraus resultierenden Ergebnisse zu bekommen.⁶

⁵ Siehe Anhang 3

⁶ Siehe Anhang 4

- **remote_movesmall()**

Diese Funktion wird ausgeführt, wenn der User auf die Koordinaten eines Schusses wartet. Die Funktion schickt zurück, ob der Schuss einen Treffer war oder nicht.⁷

Aktives Warten findet statt, wenn Player auf eine Nachricht wartet.

Zusammenfassung

Uns ist es gelungen ein komplett fehlerfrei funktionierendes Schiffe-Versenken-Spiel zu erstellen. In diesem Programm fehlt die Interprozesskommunikation (IPC) allerdings vollständig. Wir haben dann in einer zusätzlichen Datei in unser Spiel die IPC versucht zu implementieren, was uns auch gelungen ist. Hierbei kam es allerdings zu diversen Bugs und Problemen, sodass die Spielmechanik nicht mehr fehlerfrei arbeitet. Um sowohl unser Programm mit der IPC zu zeigen und das fehlerfreie Spiel befinden sich beide Versionen in unserem GitHub. Zum einen um zu zeigen, wie das Spiel mit einer IPC in Form von Sockets funktionieren würde, allerdings mit Bugs und Spielfehlern, und zum anderen wie eben dieses Programm aussehen würde, wenn es Fehlerfrei funktionieren würde.

Schlusswort

Nun eine kurze Zusammenfassung unserer gemeinsamen Arbeit am Werkstück A Alternative 7 ("Schiffe versenken").

Was ist gut gelaufen?

Die Aufteilung der einzelnen Aufgaben lief super, jeder im Team hat seine Aufgaben hervorragend erledigt, die 2 Meetings jede Woche in denen wir uns auf den aktuellen Stand brachten waren hervorragend gesetzt und dies werden wir in Zukunft wieder so fortführen.

Was ist nicht gut gelaufen?

Die letzten 3 Tage vor der Abgabe waren die Stressigsten. Viel Kleinkram wurde erst hier erledigt und angegangen. Uns fiel es schwer, nachdem wir das Spiel ohne Interprozess-Kommunikation (IPC) fertig gestellt hatten, die richtigen Ansätze zu finden die IPC in unser Programm zu implementieren. Letztlich ist es uns doch gelungen mit Hilfe von Sockets eine IPC in unser Programm einzubauen. Das Programm läuft mit der IPC aktuell leider nicht fehlerfrei, doch es läuft und es kommunizieren 2 Prozesse (2 Spieler) miteinander.

Wir sind als 5er Gruppe gestartet. 3 Wochen vor Abgabe hat sich ein Teammitglied aus der Gruppe verabschiedet, ohne Ergebnisse zu hinterlassen. Dies mussten wir nun als 4-er Gruppe abfangen.

Das GitHub Set up war umfangreicher als erwartet. Wir wollten vermeiden, dass durch die neue Umgebung unser allgemeines Zusammenarbeiten leidet. Daher haben wir uns entschieden auf die Plattform zu verzichten und weiter wie gewohnt auf Discord zu kommunizieren.

Was würden wir das nächste Mal anders machen?

⁷ Siehe Anhang 5

Um den stressigen Tagen kurz vor der Abgabe entgegenzuwirken ist eine eigene Deadline 7 Tage vor der eigentlichen Deadline sinnvoll um die letzte Woche vor der Abgabe, wenn nötig, sinnvoller nutzen zu können.

Wie war die Zusammenarbeit mit den Mitstudierenden?

Die Zusammenarbeit lief mit der zuvor erwähnten Ausnahme sehr gut. Wir arbeiten als 4er-Gruppe gut zusammen und können uns auf den jeweils Anderen jederzeit verlassen.