

Buzzword-Bingo-Spiel mit Interprozesskommunikation

Fabian Matzollek, Jakub Naruszko, Louisa Bahr, Michael Nguyen, Mustafa Islek

Frankfurt University of Applied Sciences
(1971-2014: Fachhochschule Frankfurt am Main)
Nibelungenplatz 1
60318 Frankfurt am Main
FJLMBingoSpiel@proton.me

Zusammenfassung Diese Dokumentation beschreibt die Entwicklung eines Buzzword-Bingo-Spiels, welches Python als Programmiersprache nutzt. Das Spiel spielt man in der Shell, mit der Möglichkeit, eigene Wörter zu wählen, das Spielfeld zu bestimmen und mehrere Spieler gleichzeitig gegeneinander antreten zu lassen. Ziel des Spiels ist es, wie beim herkömmlichen Bingo, alle Wörter in einer Zeile, Spalte oder Diagonale zu markieren und die übermäßige Verwendung von Schlagwörter, die keinen Inhalt besitzen, zu kritisieren. Außerdem verwendet das Spiel eine GUI-Bibliothek, die das Spiel übersichtlicher darstellt.

Das Projekt wurde im Rahmen einer Portfolioprüfung im Modul Betriebssysteme und Rechnernetze an der University of Applied Sciences in Frankfurt durchgeführt. Ziel war es, die grundlegenden Konzepte der Interprozesskommunikation und der Python-Programmierung zu erlernen und vertiefen. Die folgende Dokumentation gibt einen Überblick über die Architektur des Spiels, die verwendeten Mittel und Bibliotheken, sowie die Herausforderungen und Lösungen, die während der Entwicklung auftraten.

1 Die Anforderungen des Spiels

In erster Linie soll das Spiel funktionsfähig sein, d.h. ausführbar sein und keine Fehler in der Nutzung beinhalten. Außerdem soll es folgende Funktionen besitzen:

- Auswahl zufälliger Wörter aus einer benutzerdefinierten Wörterliste.
- Einfügen der Wörter in eine Matrix variabler Größe.
- Die Mitte des Feldes (falls vorhanden) ist ein Joker.

- Anzahl und Namen der Spieler sind wählbar.
- Kommandozeilenanwendung mit verständlich kommentiertem Code.
- Jeder Spieler spielt in einem eigenen Prozess auf demselben Computer.
- Spieler können Wörter markieren und Fehler rückgängig machen.
- Grafische Ausgabe erfolgt durch eine GUI-Bibliothek in der Shell.
- Erkennung und sinnvolle Behandlung fehlerhafter Eingaben.
- Sieg eines Spielers wird für alle Teilnehmer erkennbar angezeigt.
- Erstellung von Logdateien für jeden Spieler.
- Ein Spieler eröffnet die Partie, andere können beitreten.

Die Anforderungen wurden aus diesem **Dokument**¹ übernommen.

2 Die Aufteilung der Aufgaben

Zu Beginn des Projekts wurden die Aufgaben klar aufgeteilt. Bei der Umsetzung dieser klaren Aufteilung entstanden jedoch Probleme. Da die Aufgaben aufeinander aufbauten, war es herausfordernd, nur seinen Teil zu erledigen. Zum Beispiel war es schwierig, eine gute GUI zu entwickeln, während die Funktionen für das Spiel nicht fertig waren. Ebenso konnte man keine Interprozesskommunikation einbauen, solange es noch kein Fundament gab. Um dieses Problem zu lösen, wurde beschlossen, dass jeder versuchte, seinen Teil zu erledigen, aber allen anderen Gruppenmitgliedern auch half. So wurde aus der klaren Einteilung ein grober Leitfaden, der die Vorgehensweise definierte.

3 Benötigte Bibliotheken bzw. Imports

Damit das Spiel funktionsfähig ist, benötigt man zusätzliche Installationen und Bibliotheken (bzw. Imports), diese sind:

- Python² (idealerweise 3.10.12)
- Ubuntu (WSL³)
- pip⁴ zur Installation von Paketen (*sudo apt install python3-pip*)
- rich-Bibliothek⁵ (*python3 -m pip install rich*)
- prompt-toolkit-Bibliothek⁶ (*python3 -m pip install rich*)

¹ https://www.christianbaun.de/BSRN24/Skript/bsrn_SS2024_portfoliopruefung_teil_1_alternative_3.pdf

² <https://www.python.org/downloads/>

³ <https://learn.microsoft.com/en-us/windows/wsl/install>

⁴ <https://pypi.org/project/pip/>

⁵ <https://github.com/Textualize/rich>

⁶ <https://github.com/prompt-toolkit/python-prompt-toolkit>

4 Bibliotheken und Imports

Um unnötige Imports zu vermeiden, nutzen wir als externe Bibliotheken 'rich' und 'prompt-toolkit'. Eine ausführliche Erklärung aller Imports:

- **os**: Bietet eine Schnittstelle zu Betriebssystemfunktionen wie das Arbeiten mit Dateisystemen und Umgebungsvariablen.
- **sys**: Ermöglicht den Zugriff auf systembezogene Parameter und Funktionen, einschließlich des Beendens des Programms und des Arbeitens mit der Standard-Eingabe und -Ausgabe.
- **random**: Stellt Funktionen zur Verfügung, um Zufallszahlen zu generieren und zufällige Auswahlvorgänge durchzuführen.
- **time**: Ermöglicht den Zugriff auf zeitbezogene Funktionen wie das Messen der Zeit und das Erzeugen von Pausen.
- **rich**: Wird verwendet, um die Konsole zu steuern und Inhalte wie Tabellen und dekorative Panels farbig anzuzeigen.
- **prompt_toolkit**: Ermöglicht das Erstellen und Verwalten von interaktiven Benutzereingaben sowie das Erstellen von Dialogen und die Formatierung von Texteingaben.
- **datetime**: Stellt Klassen zur Verfügung, um mit Datum und Uhrzeit zu arbeiten.

5 Funktionen und Methoden

In den nächsten Abschnitten werden die Funktionen und Methoden des Spiels gezeigt und erklärt. Aus Platz- und Visualisierungsgründen, werden nur, aus unserer Sicht, die wichtigsten Methoden erklärt. Ggf. werden Methoden auch verkürzt dargestellt oder so verändert, dass diese kürzer sind aber die Funktion genau die gleiche bleibt.

5.1 Initialisierung der Wörterliste

Die Wörter werden von einer Wörterliste geladen. Der Nutzer kann die Wörterliste selbst gestalten und bei jedem Spiel diese ändern bzw. eine neue Liste verwenden.

```
1 def initialize_file(filename):  
2     global buzzwords_list  
3     try:  
4         with open(filename) as file: # Öffnet die Datei  
5             reader = file.readlines() # Liest die Datei ein
```

```

6         buzzwords_list = [i.strip() for i in reader] #
           Entfernt Zeilenumbrüche und speichert Wörter
           in Liste
7         random.shuffle(buzzwords_list) # Mischt die Liste,
           um zufällige Auswahl zu gewährleisten
8     except FileNotFoundError:
9         console.print(f"Fehler: Datei '{filename}' nicht
           gefunden.", style="bold red")
10        return False
11    return True

```

Code-Ausschnitt 1.1. Laden der Wörterliste

Der Nutzer kann durch die Funktion `get_filename()` die Wörterliste selbst wählen, indem er den Namen der Wörterliste eingibt. Dies geschieht durch folgenden Code:

```

1 def get_filename():
2     [...]
3     filename = session.prompt(HTML("Geben Sie den <
           ansigreen>Dateinamen</ansigreen> ein, aus dem die
           Wörter gezogen werden sollen: ")) # Prompt zur
           Eingabe des Dateinamens
4     if initialize_file(filename):
5         return filename

```

Code-Ausschnitt 1.2. Auswahl der Wörterliste

Bei Entwicklung der Wörterlisten entstanden keine Probleme. Um das Spiel einfacher zu gestalten haben, gibt es bei der vorgegebenen Wörterliste nur kurze englische Wörter. Dies vermeidet viele Probleme, wie den Umgang mit deutschen Umlauten oder das Strecken der Wörterfelder

5.2 Verwaltung von Spielern

Bevor das Spiel beginnt, wird abgefragt, wie viele Spieler am Spiel teilnehmen. Dies geschieht durch die Funktion `get_player_count()`, die ebenso, wie die `get_filename()`-Funktion, eine Prompt zur Eingabe nutzt.

```

1 def get_player_count():
2     [...]
3     playercount = int(session.prompt(HTML("Geben Sie
           die <ansicyan>Spieleranzahl</ansicyan> ein: "
           )))
4     return playercount
5     except ValueError:
6         console.print("Fehler: Die Eingabe muss eine
           ganze Zahl sein.", style="bold red")

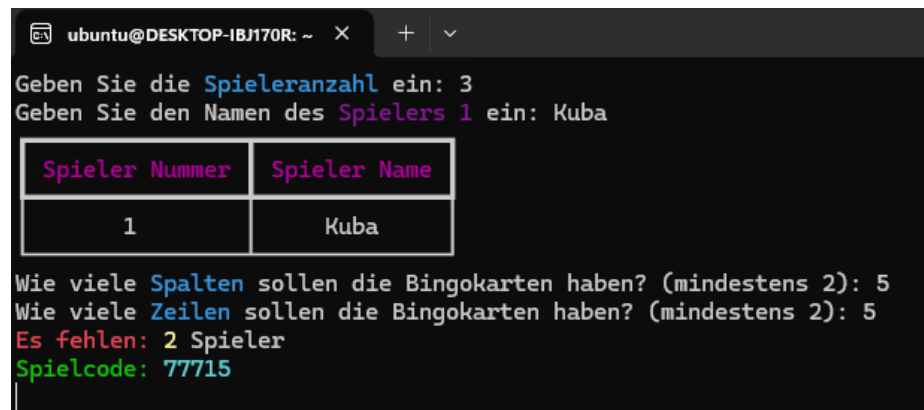
```

Code-Ausschnitt 1.3. Spieleranzahl Bestimmung

Alle Abfragen (Wörterliste, Spieleranzahl, Spielernamen, Feldgröße) erfolgen nach dem gleichen Prinzip. Durch eine Aufforderung muss der Spieler die Werte bestimmen, bevor das Spiel starten kann. Die Besonderheit bei der def *get_player_names()*-Funktion ist, dass diese den Wert von *playercount* übernimmt.

5.3 Die Generierung des Spielfelds

Bevor die Karten generiert werden, muss der Spieler zunächst die Feldgröße bestimmen. Dies passiert durch zwei Funktionen *get_dimensionx* und *get_dimensiony*, die die Breite und die Länge des Felds definieren. Wie bereits beschrieben, übernimmt das Spiel die Eingabe des Spielers durch Prompts.



```
ubuntu@DESKTOP-IBJ170R: ~ X + v
Geben Sie die Spieleranzahl ein: 3
Geben Sie den Namen des Spielers 1 ein: Kuba



| Spieler Nummer | Spieler Name |
|----------------|--------------|
| 1              | Kuba         |



Wie viele Spalten sollen die Bingokarten haben? (mindestens 2): 5
Wie viele Zeilen sollen die Bingokarten haben? (mindestens 2): 5
Es fehlen: 2 Spieler
Spielcode: 77715
```

Abbildung 1. Bildschirmaufnahme des Spiels, nachdem alle Parameter eingegeben wurden

Nachdem alle Spieler mit dem Spielcode beigetreten sind, beginnt das Spiel, indem die Funktion *generate_bingo_cards()* die Karten generiert. Der vollständige Code mit Kommentaren befindet sich im CA 1.7, weswegen auf eine ausführliche Analyse hier verzichtet und stattdessen auf Probleme und Lösungen eingegangen wird.

1. Zu wenige Bingokarten in der Wörterliste führten zum Absturz des Spiels.
 - (a) Überprüfung, ob Wörterliste größer als $y \cdot x$ ist, falls nicht, konkrete Fehlermeldung.

2. Wörter kamen doppelt in dem Spielfeld eines Spielers vor.
 - (a) Verwendung der Variablen 'used_words', damit dann ein neues Wort genutzt wird.
3. Joker befand sich nicht in der Mitte, war bei nicht korrekter Feldgröße (gerade Zahlen) vorhanden oder wurde zur Feldgröße hinzugefügt, anstatt das mittige Wort zu ersetzen.
 - (a) Korrekte Berechnung der Mitte mit '//' (abrunden), anstatt '/' (aufrunden).
 - (b) Lösung der anderen Probleme erfolgte durch Implementierung einer if-Anweisung, bei der der Joker nur hinzugefügt wird, wenn die Feldbreite und Feldlänge nicht durch 2 teilbar ist.

5.4 Die Visualisierung

Eine passende GUI-Bibliothek zu finden, war umständlich. Da eine der Anforderung eine Kommandozeilenanwendung forderte, fielen Optionen wie tkinter⁷ weg. Eine Alternative war pyTermTk⁸. Die Implementierung erschien aber zu aufwendig und komplex, weswegen rich⁹ zum Einsatz kam.

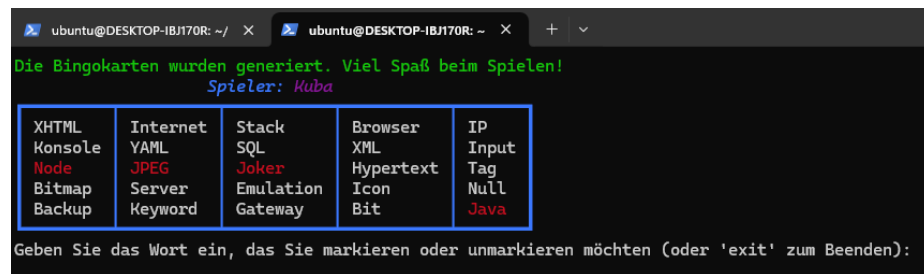


Abbildung 2. Das Layout des Spiels

```

1 def display_bingo_cards(playernamelist, matrixlist,
2   marked_words): # Funktion, die die Bingokarten anzeigt
3   print("\033c", end="", flush=True) # Löscht die Konsole
4
5   console.print("Die Bingokarten wurden generiert. Viel Spaß
6     beim Spielen!", style="bold green")
7   for i, matrix in enumerate(matrixlist): # i ist der
8     Index, matrix ist die Bingokarte

```

⁷ <https://docs.python.org/3/library/tkinter.html>

⁸ <https://github.com/ceccopierangiolieugenio/pyTermTk>

⁹ <https://github.com/Textualize/rich>

```

6
7     table = Table(show_header=False, box=HEAVY_EDGE,
8                   border_style="bold blue",
9                   title=f"[bold blue]Spieler:[/bold blue]
10                      [magenta]{playernamelist[i]}[/
11                      magenta]")
12
13     for _ in range(len(matrix[0])): # Spaltenanzahl
14         table.add_column()
15
16     for row in matrix: # row ist eine Zeile der
17         Bingokarte
18         table.add_row(*[f"[red]{cell}[/red]" if cell in
19                        marked_words or cell == "Joker" else str(cell)
20                        for cell in row]) # Ausgabe der Bingokarte
21
22     console.print(table)

```

Code-Ausschnitt 1.4. Umsetzung der Tabelle in Abbildung 2.

Das Anzeigen des Spielfeldes und der Bingokarte erfolgt durch die Funktion *display_bingo_cards()*. Für jeden Spieler entsteht eine Tabelle, gefüllt mit den entsprechenden Karten. Markierte Wörter und Joker erscheinen farblich hervorgehoben. Die fertigen Tabellen erscheinen schließlich in der Konsole. Die äußere Schleife iteriert über die Spieler und ihre Karten, während die inneren Schleifen die Spalten und Zeilen jeder Karte durchlaufen, um die Tabellen zu füllen.

5.5 Die Logdatei

Das Spiel speichert Ereignisse in individuellen Logdateien für jeden Spieler. Jede Logdatei dokumentiert das Starten des Spiels, sämtliche Aktionen und das Spielende. Dies ermöglicht eine einfache Nachverfolgung und Fehleranalyse. Die Erstellung und Verwaltung der Logdateien erfolgt durch die Funktionen *create_log_file()* und *log_event()*. Die Funktion *create_log_file()* generiert eine neue Logdatei mit einem Zeitstempel im Namen, während *log_event()* Ereignisse mit Zeitstempel in diese Datei schreibt.

```

1 def create_log_file(pid):
2     timestamp = datetime.now().strftime("%Y-%m-%d-%H-%M-%S")
3     filename = f"{timestamp}-bingo-Spieler{pid}.txt"
4     log_files[0] = open(filename, 'w')
5     log_files[0].write(f"{timestamp} Start des Spiels\n")
6     return log_files[0]
7
8 def log_event(event):
9     timestamp = datetime.now().strftime("%Y-%m-%d-%H-%M-%S")
10    log_files[0].write(f"{timestamp} {event}\n")

```

Code-Ausschnitt 1.5. Erstellung und Verwaltung der Logdatei

5.6 Kommunikation über Pipes

Das Spiel nutzt Pipes zur Interprozesskommunikation, um Daten zwischen den einzelnen Spielerprozessen auszutauschen. Pipes ermöglichen eine effiziente und direkte Datenübertragung, die für das synchrone Spiel unerlässlich ist. Die Funktionen *send()* und *empfang()* verwalten das Senden und Empfangen von Nachrichten. *sendfile()* und *empfangfile()* übernehmen die Übertragung und den Empfang von Spieldaten.

Die Funktion *send()* öffnet eine Pipe zum Schreiben und sendet eine Nachricht. Die Funktion *empfang()* liest Nachrichten von den Pipes der Gegenspieler und speichert diese in einer Liste. *sendfile()* überträgt Spieldaten an die Pipes der Gegner, während *empfangfile()* die Daten von der eigenen Pipe empfängt und in einer Liste speichert.

```
1 def send(pipe, pipm):  
2     fifo = os.open(str(pipe), os.O_WRONLY) # Pipe öffnen zum  
        Schreiben  
3     s = f"{pipm}\n".encode()  
4     os.write(fifo, s) # Pipe-Nachricht schreiben  
5     os.close(fifo) # Pipe schließen
```

Code-Ausschnitt 1.6. Nachricht an Pipe senden

6 Schlusswort

Diese Dokumentation beschreibt die Entwicklung eines Buzzword-Bingo-Spiels, welches in Python programmiert und über die Kommandozeile ausgeführt wird. Ziel des Projekts war es, die grundlegenden Konzepte der Interprozesskommunikation und der Python-Programmierung zu erlernen und praktisch umzusetzen.

Das Spiel erlaubt es den Nutzern, eigene Wörterlisten zu verwenden, die Spielfeldgröße zu bestimmen und mehrere Spieler gleichzeitig gegeneinander antreten zu lassen. Eine benutzerfreundliche GUI-Bibliothek, *rich*, wurde eingesetzt, um eine ansprechende visuelle Darstellung in der Kommandozeile zu ermöglichen. Die Funktion *display_bingo_cards()* visualisiert die Spielkarten farbenfroh und übersichtlich, was die Benutzererfahrung erheblich verbessert.

Die Implementierung der Logdateien durch *create_log_file()* und *log_event()* ermöglicht eine umfassende Nachverfolgung und Fehleranalyse des Spielverlaufs.

Jede Aktion der Spieler wird detailliert in individuellen Logdateien dokumentiert, was besonders nützlich für Debugging-Zwecke und die Auswertung des Spielverhaltens ist.

Ein zentraler Aspekt des Spiels ist die Interprozesskommunikation mittels Pipes. Funktionen wie *send()* und *empfang()* gewährleisten einen reibungslosen Datenaustausch zwischen den Spielerprozessen. Trotz der robusten Implementierung gibt es Potenzial für weitere Optimierungen, insbesondere in Bezug auf die Effizienz und die Geschwindigkeit der Kommunikation. Die aktuelle Implementierung kann bei größeren Spielerzahlen zu Verzögerungen führen, weshalb ein zukünftiger Ansatz die Nutzung von Message Queues oder Sockets in Betracht ziehen könnte.

Zusammenfassend lässt sich sagen, dass das Projekt erfolgreich die Ziele der Portfolioprüfung erfüllt hat. Die entwickelten Funktionen arbeiten zuverlässig und bieten eine solide Basis für zukünftige Erweiterungen und Verbesserungen. Zu den möglichen Erweiterungen gehören die Implementierung einer grafischen Benutzeroberfläche, die Integration von Netzwerkfähigkeit für das Spielen über verschiedene Geräte hinweg und die Optimierung der Interprozesskommunikation. Diese Verbesserungen könnten die Benutzerfreundlichkeit weiter erhöhen und das Spiel auf ein neues Niveau heben.

7 Anhang

```
1  def generate_bingo_cards(playernamelist, xsize, ysize):
2      # Funktion, die die Bingokarten generiert
3      matrixlist = [] # Liste der Bingokarten
4      middle_x = xsize // 2 # Mitte der x-Achse zur Bestimmung
5                          # des Jokers
6      middle_y = ysize // 2 # Mitte der y-Achse zur Bestimmung
7                          # des Jokers
8
9      for k in playernamelist: # Schleife, die die Bingokarten
10                             # für jeden Spieler generiert
11                             if len(buzzwords_list) < xsize * ysize: # Überprü-
12                                 # fung, ob genug Buzzwörter vorhanden sind
13                                 raise ValueError(
14                                     "Nicht genug Buzzwords, um die Bingokarten zu
15                                     füllen") # Fehlermeldung, wenn nicht
16                                     # genug Buzzwörter vorhanden sind
17
18     used_words = set() # Set für verwendete Wörter
19     matrix = [] # Liste für die Bingokarte
20     for l in range(ysize): # Zeilen
21         b = [] # Liste für die Zeile
22         for j in range(xsize): # Spalten
23             if xsize % 2 != 0 and ysize % 2 != 0 and l ==
24                 middle_y and j == middle_x: # Joker in
```

```

    der Mitte, nur wenn xsize und ysize
    ungerade sind
16     b.append("Joker") # Fügt den Joker in
    die Mitte der Bingokarte ein
17     else:
18         while True:
19             random_word = buzzwords_list.pop(0)
    # Nimmt das erste Element aus der
    Liste und entfernt es, damit
    kein Wort doppelt vorkommt
20             if random_word not in used_words: #
    Überprüfung, ob das Wort bereits
    verwendet wurde
21                 used_words.add(random_word) # Fü
    gt das Wort zu den
    verwendeten Wörtern hinzu
22                 b.append(random_word) # Fügt das
    zufällige Wort in die
    Bingokarte ein
23                 break # Beendet die Schleife,
    wenn ein neues Wort gefunden
    wurde
24             buzzwords_list.append(random_word) # Fü
    ge das Wort zurück zur Liste, damit
    es nicht verloren geht
25         matrix.append(b) # Fügt die Zeile der Bingokarte
    in die Bingokarte ein
26         matrixlist.append(matrix) # Fügt die
    Bingokartenmatrix einer Person in die
    Bingokartenliste ein
27     return matrixlist # Gibt die Liste der Bingokarten zurü
    ck
```

Code-Ausschnitt 1.7. Generierung der Bingokarten