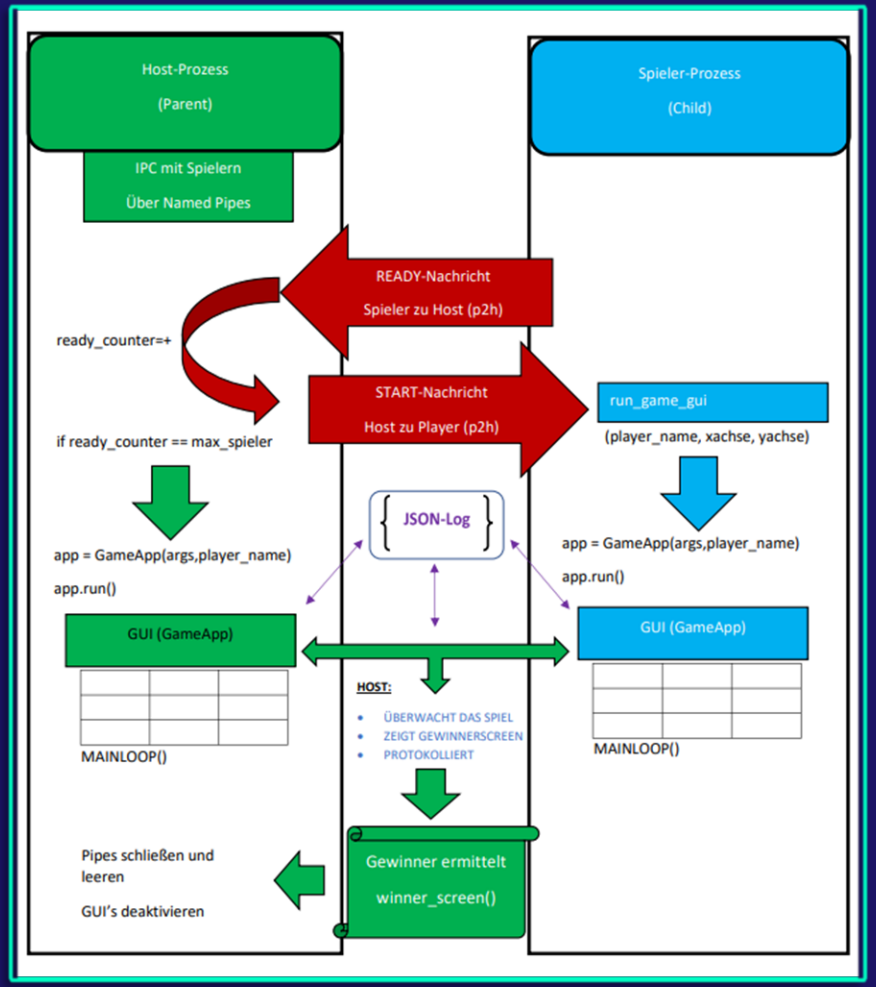
BSRN Buzzword-Bingo

Khaled Badrash, Mohamed Muheisen

Spiellogik

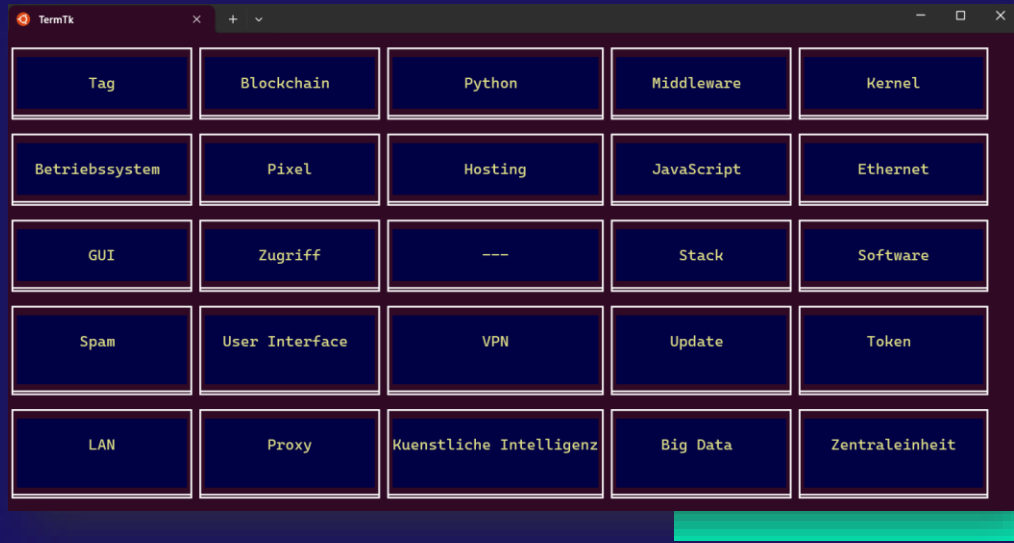
1. Initialisierung
2. Host: Kommunikation READY- und START-Nachrichten
3. Spieler: joinen , p2h, h2p
4. Initialisiert die GUI
5. JSON-Logging
6. Spielüberwachung
7. Gewinnerermittlung
8. Spielende



Grafische Benutzeroberfläche

Bibliothek: PyTermTK

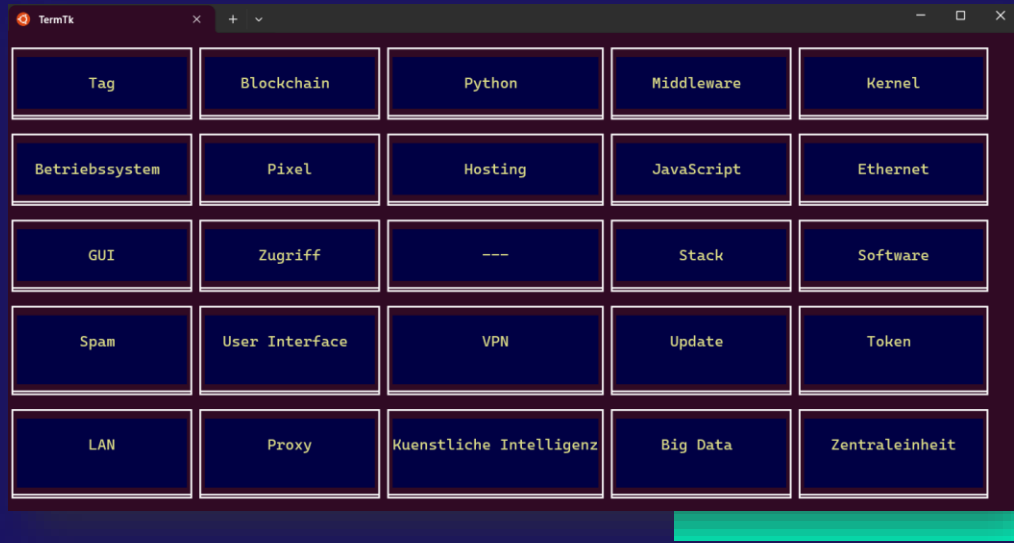
- Bingo-Felder als Buttons
- Markierung der Felder
- Prüfung Bingo
- Gewinner-Fenster



Grafische Benutzeroberfläche

Bingo-Felder als Buttons

```
button = ttk.TTkButton(parent=self.root, text=wort, border=True, pos=(i, j))
```

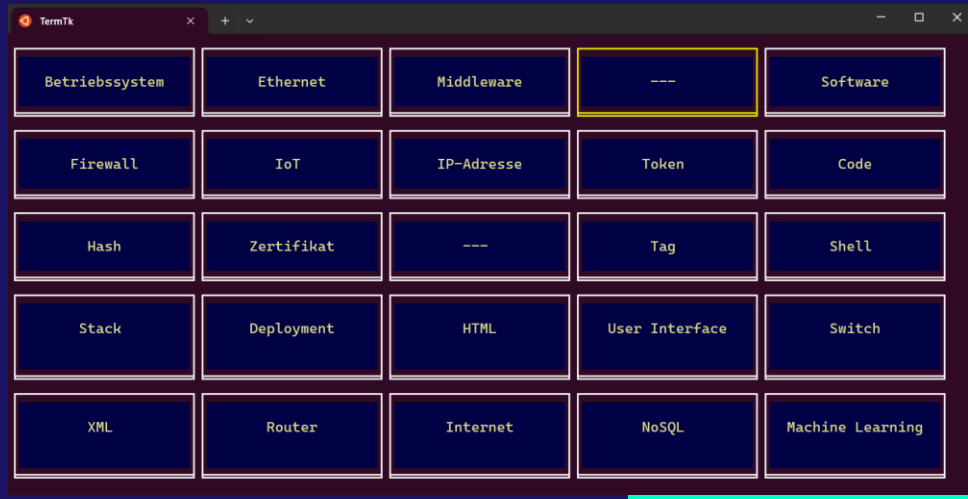


Grafische Benutzeroberfläche

Markierung der Felder (def button_click)

```
def button_click(self, button, x_wert, y_wert):
```

```
else:
    original_text = button.text()
    button.setText("---")
    auswahl_zeitpunkt = datetime.now().strftime('%d-%m-%Y %H:%M:%S Uhr')
    self.log_data_json(original_text, x_wert, y_wert, auswahl_zeitpunkt)
```



Grafische Benutzeroberfläche

Bingo-Felder als Buttons

```
for i in range(self.args.xachse):
    for j in range(self.args.yachse):

        if i == self.args.xachse // 2 and j == self.args.yachse // 2:
            button = ttk.TTkButton(parent=self.root, text='---', border=True, pos=(i, j))
            grid_layout.addWidget(button, i, j)
            self.log_joker('---', i, j, datetime.now().strftime('%d-%m-%Y %H:%M:%S Uhr'))
        else:
            wort = self.woerter[i * self.args.yachse + j]
            button = ttk.TTkButton(parent=self.root, text=wort, border=True, pos=(i, j))
            self.original_texts[button] = button.text()
            button.clicked.connect(lambda btn=button, x=i, y=j: self.button_click(btn, x, y))
            grid_layout.addWidget(button, i, j)
```

Layout (3X3)

i = 0, j = 0	i = 0, j = 1	i = 0, j = 2
i = 1, j = 0	i = 1, j = 1	i = 1, j = 2
i = 2, j = 0	i = 2, j = 1	i = 2, j = 2

Grafische Benutzeroberfläche

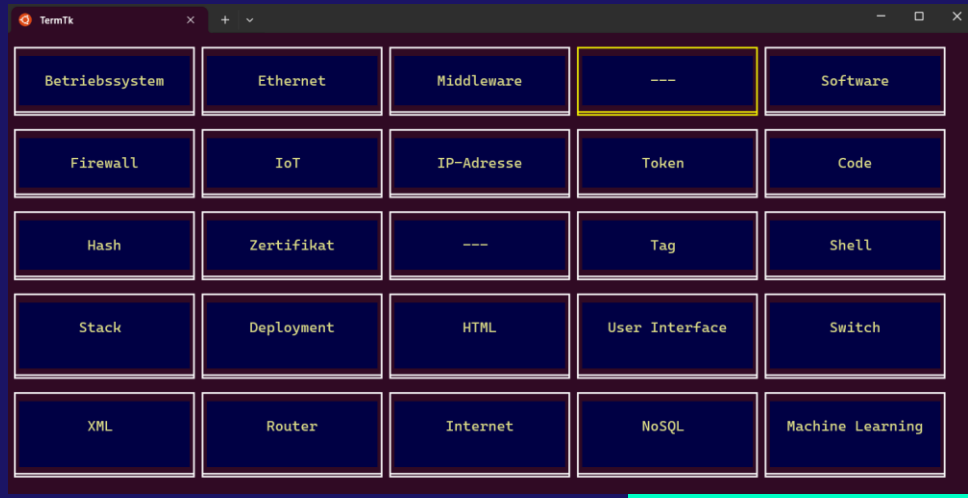
Markierung der Felder (def button_click)

```
def button_click(self, button, x_wert, y_wert):

if button.text() == "---":
    log_index = next((index for (index, d) in enumerate(logs) if d.get("x_wert") ==
x_wert and d.get("y_wert") == y_wert), None)

    if log_index is not None and log_index == len(logs) - 1:
        logs.pop(log_index)
        button.setText(self.original_texts[button])
        write_json_log(logs)

else:
    original_text = button.text()
    button.setText("---")
    auswahl_zeitpunkt = datetime.now().strftime('%d-%m-%Y %H:%M:%S Uhr')
    self.log_data_json(original_text, x_wert, y_wert, auswahl_zeitpunkt)
```



Grafische Benutzeroberfläche

Prüfung Bingo

```
def pruefe_bingo(max_feld, logs):
    # Funktion:
    marked_positions = [(log.get('x_wert'), log.get('y_wert')) for log in logs
                        if log.get('button_text') == '---' or log.get('JOKER') == '---']

    for i in range(max_feld):
        if all((i, j) in marked_positions for j in range(max_feld)):
            return True

    for j in range(max_feld):
        if all((i, j) in marked_positions for i in range(max_feld)):
            return True

    if all((i, i) in marked_positions for i in range(max_feld)):
        return True

    if all((i, max_feld - 1 - i) in marked_positions for i in range(max_feld)):
        return True

    return False
```

Layout (3X3)

i = 0, j = 0	i = 0, j = 1	i = 0, j = 2
i = 1, j = 0	i = 1, j = 1	i = 1, j = 2
i = 2, j = 0	i = 2, j = 1	i = 2, j = 2

Named Pipes

- Vorteile
  - unabhängige Kommunikation Host ←→ Spielern.
  - Persistent:
    - erleichtert die Fehlerbehebung.

```
KhaledBadrash *
def setup_pipes(): # Funktion: Einrichtung der Pipes für die IPC
    host_to_players_path = '/tmp/host_to_players' # Pfad für die Pipe h2p
    players_to_host_path = '/tmp/players_to_host'  # Pfad für die Pipe p2h

    if os.path.exists(host_to_players_path):
        os.remove(host_to_players_path) # Entferne vorhandene Pipe, falls s
    if os.path.exists(players_to_host_path):
        os.remove(players_to_host_path) # Same[Fehlerbehebung]

    os.mkfifo(host_to_players_path, mode=0o666)  #Erstelle FIFO-Pipe: h2p
    os.mkfifo(players_to_host_path, mode=0o666)  #Erstelle FIFO-Pipe: p2h
    #mode-Parameter definiert die Zugriffsberechtigungen:
                                #0o666 --> 6 für Lesen und Schreiben
    #                                      4 für Lesen
    #                                      2 für Schreiben
```

Prozessstruktur und IPC 1/3

- Host-Prozess Initialisierung
  - setup_pipes()
- Warten auf READY-Nachrichten
- Verbindungsstatus der Spieler
- START-Nachrichten Senden

```python
def handle_host_connections(args, conn):
    # Funktion zur Handhabung der Host-Verbindungen
    print(f"Host-Prozess gestartet mit PID: {os.getpid()}")
    anz_spieler = args.max_spieler
    print(f"Warte auf {anz_spieler} Spieler...")
    setup_pipes()  #Richte die Pipes ein
    connected_players = 0

    with open('/tmp/host_to_players', 'w') as h2p, open('/tmp/players_to_host', 'r') as p2h:
        while connected_players < anz_spieler:
            print("Warten auf READY-Nachricht von einem Spieler...")
            line = p2h.readline().strip()  # Lese eine Zeile von den Spielern
            if line == 'READY':
                connected_players += 1
                print(f"Spieler {connected_players} verbunden.")
                print(f"Warte auf {anz_spieler - connected_players} Spieler...")

        print("Alle Spieler sind verbunden. Das Spiel beginnt!")
        for _ in range(anz_spieler):
            h2p.write(f'START {args.xachse} {args.yachse}\n')  # Sende START-Nachricht an die Spieler
            h2p.flush()  #schreibt alle gepufferten Daten sofort in die Pipe --> keine Verzögerung
            print(f"START-Nachricht an {connected_players} Spieler gesendet.")

        conn.send(True)  #Sende Nachricht an den Hauptprozess, dass das Spiel gestartet werden kann
```

Prozessstruktur und IPC 2/3

- Spieler-Prozess Initialisierung
- Pipe-Verbindungen Öffnen
  - Exeption
- Spielerbereitmeldung

```
def player_process(player_name):
    print(f"Spieler-Prozess gestartet mit PID: {os.getpid()}")
    #hilfreich um später zu sehen, ob es tatsächliche Prozesse sind

    try:
        h2p = open('/tmp/host_to_players', 'r')
        p2h = open('/tmp/players_to_host', 'w')
    except Exception as e:
        print(f"Fehler beim Öffnen der Pipes: {e}")
        return

    print(f"Spieler {player_name} ist bereit.")
```

Prozessstruktur und IPC 3/3

- Warten auf START-Nachricht
- Nachricht empfangen
- Koordinaten extrahieren
  - X / Y aus „START“
- Startet die Spiel-GUI

```
def player_process(player_name):
        p2h.close()
        return

    while True:
        print("Warten auf START-Nachricht vom Host...")
        try:
            start_message = h2p.readline().strip()  #Lese START-Nachricht vom Host
            if start_message:
                print(f"Nachricht vom Host empfangen: {start_message}")
                if start_message.startswith('START'):
                    _, xachse, yachse = start_message.split()
                    xachse = int(xachse)
                    yachse = int(yachse)
                    print(f"DEBUG: Spiel beginnt für {player_name} mit den Koordinaten ({xachse}, {yachse})")
                    run_game_gui(player_name, xachse, yachse)  # Starte die Spiel-GUI
                    break
            else:
                print("Keine Nachricht empfangen. Warten auf Nachricht...")
                time.sleep(1)
        except Exception as e:
            print(f"Fehler beim Lesen der START-Nachricht: {e}")
            break
```

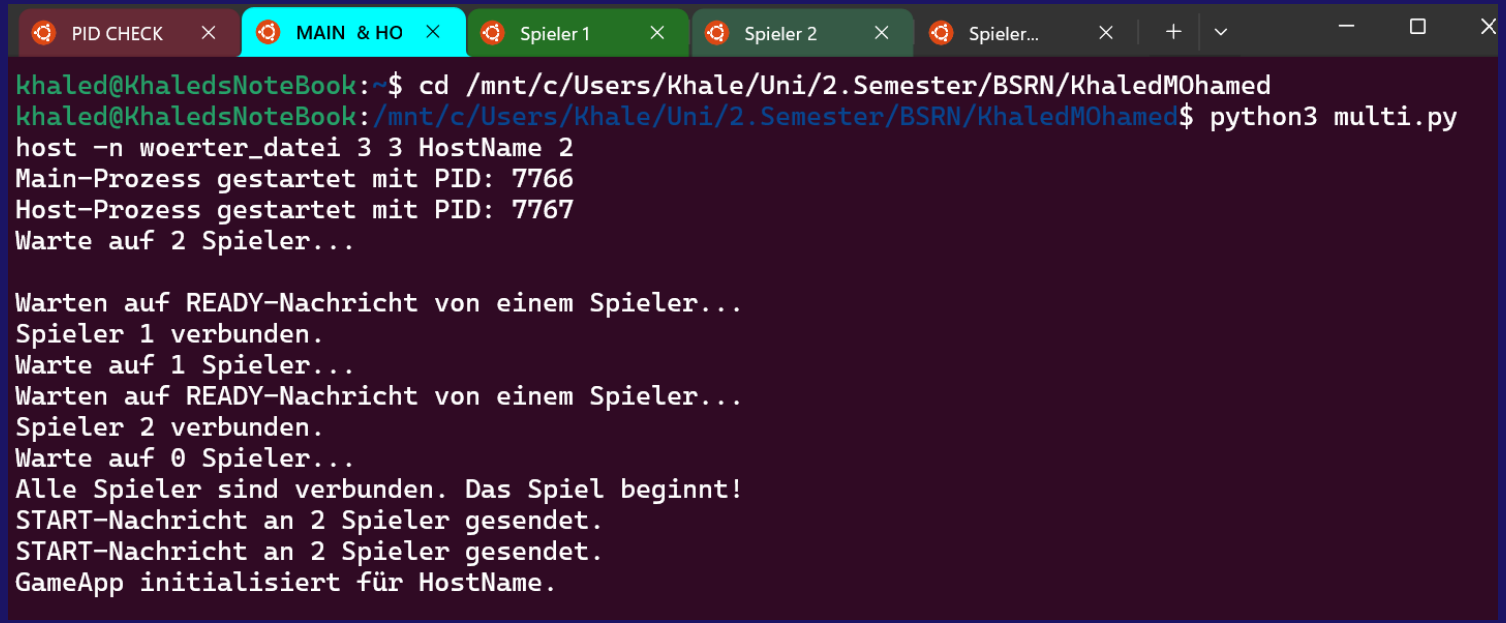
Multiprocessing

- MP → unabhängigen Prozessen
- Pipe-Paar erstellen
- Verbindungsprozess erstellen
  - Argumente übergeben
- start_method()
  - ‚fork' → Multithreading zu vermeiden
- Prozess starten

```
parent_conn, child_conn = Pipe()  # Erstelle ein Pipe-Paar: ermöglicht bidirektionale Kommunikation zwischen Prozessen
connection_process = Process(target=handle_host_connections, args=(
args, child_conn))  # Erstelle einen neuen Prozess durch die Klasse Process aus dem MP Modul
multiprocessing.set_start_method('fork', force=True)  #jetzt wird geforkt, kein Multithreading
#Setzt den Startmodus für Prozesse auf "fork" um threading zu vermeiden
connection_process.start()  #Starte den Verbindungsprozess
```

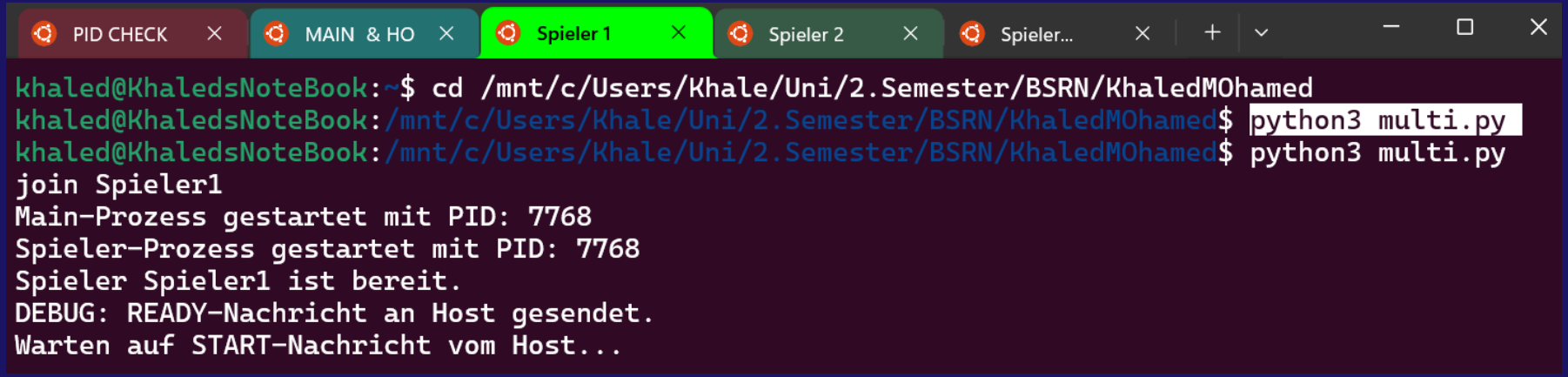
Analyse der Prozessstruktur

- Main Prozess
- Host Prozess



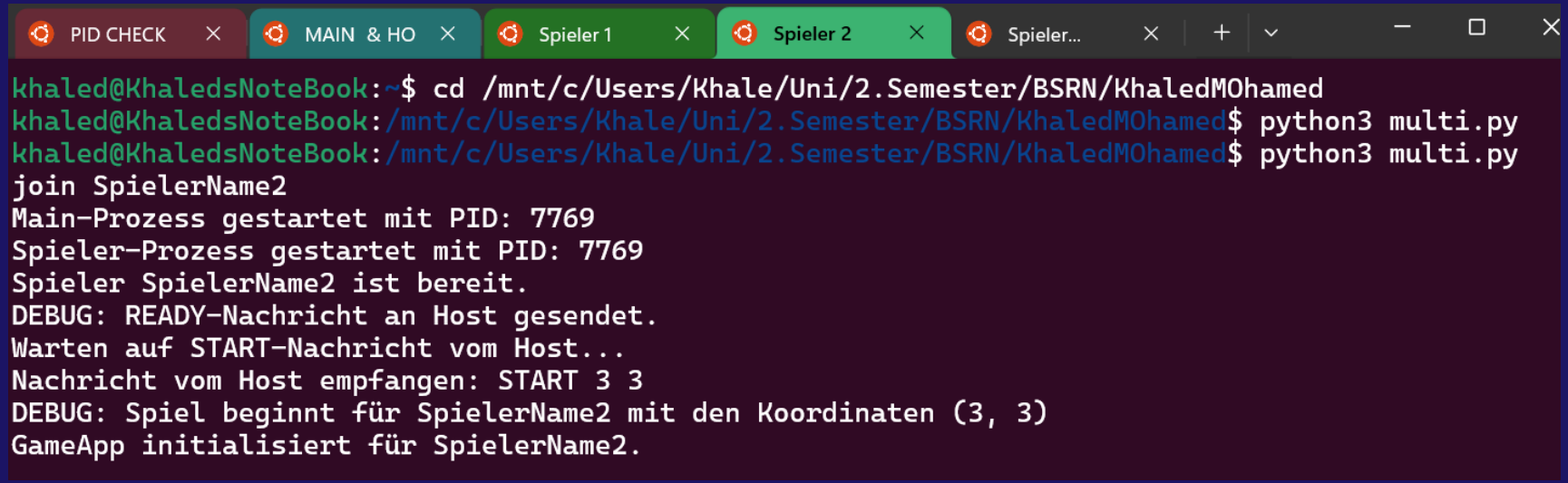
Analyse der Prozessstruktur

- Spieler Prozess 1



Analyse der Prozessstruktur

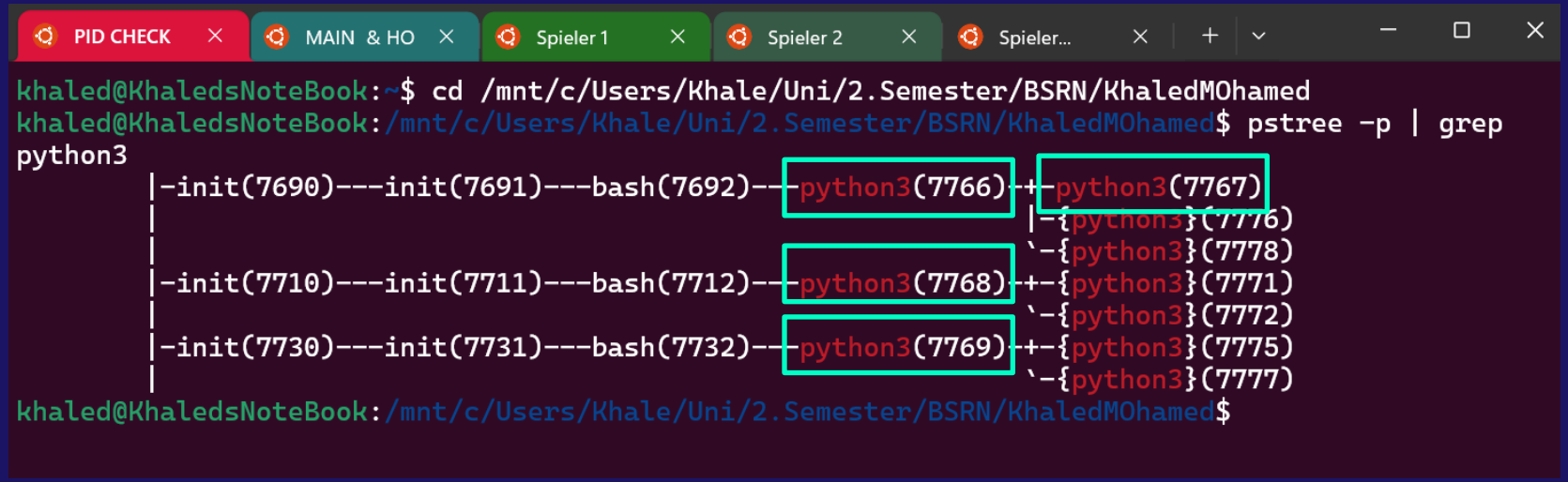
- Spieler Prozess 2



Analyse der Prozessstruktur

- Main → 7766
- Host → 7767
- Spieler 1 → PID 7768
- Spieler 2 → PID: 7769

- pstree -p | grep python3
  - Hierarchie
  - Unabhängigkeit der Prozesse



Analyse der Prozessstruktur

- Main → 7766
- Host → 7767
- Spieler 1 → PID 7768
- Spieler 2 → PID: 7769

- ps aux
  - Liste aller laufenden Prozesse
  - CPU- und Speichernutzung

```
khaled     7752  0.0  0.1   6212  5200 pts/5    Ss+  21:04   0:00 -bash
khaled     7766  0.2  0.5 170784 19720 pts/1    Tl   21:10   0:00 python3 multi.py host -n woer
khaled     7767  0.0  0.0      0     0 pts/1    Z    21:10   0:00 [python3] <defunct>
khaled     7768  0.2  0.4 170732 19256 pts/3    Sl+  21:11   0:00 python3 multi.py join Spieler
khaled     7769  0.2  0.4 170732 19068 pts/4    Tl   21:12   0:00 python3 multi.py join Spieler
khaled     7781  0.0  0.0   7480  3164 pts/2    R+   21:14   0:00 ps aux
khaled@KhaledsNoteBook:/mnt/c/Users/Khale/Uni/2. Semester/BSRN/KhaledMOhamed$
```

Live Spiel vorführung


print("Wir bedanken uns\nfür Ihre\nAufmerksamkeit!")
Wir bedanken uns für Ihre Aufmerksamkeit!
print("Khaled Badrash")
print("Mohamed Muheisen")