Portfolioprüfung – Werkstück A – Alternative 3

Dokumentation des Bingo-Spiels in Python

Batuhan Erdogan,
Marvin Ehmler,
Marc-Lauri Franke,
Ramiz Babayev,
Milad ?

Frankfurt University of Applied Sciences

Betriebssysteme und Rechnernetze

Christian Baun

XX.0X.2024, Frankfurt am Main

Inhaltsverzeichnis

1. Einleitung

Dieses Dokument beschreibt ein Bingo-Spiel in Python, das für zwei Spieler konzipiert wurde. Das Programm verwendet die asciimatics-Bibliothek zur Darstellung der Bingokarten im Terminal und die multiprocessing-Bibliothek zur Handhabung der parallelen Spielerprozesse. Es bietet eine einfache Möglichkeit, ein interaktives Bingo-Spiel für zwei Spieler zu spielen und die Ergebnisse in Log-Dateien zu speichern.

2. Projektübersicht

Das Projekt besteht aus mehreren Funktionen zur Generierung und Anzeige von Bingokarten, Überprüfung von Gewinnbedingungen sowie zur Handhabung der Spielerinteraktionen. Die Hauptfunktion “host_game” koordiniert das Spiel, während “player_game” die Logik für jeden Spielerprozess enthält.

3. Implementierungsdetails

Das Programm nutzt mehrere Python-Bibliotheken:
- “random” zur Zufallsauswahl und Durchmischung von Wörtern,
- “os” zur Dateiverwaltung,
- “datetime” zur Zeitstempelung von Log-Einträgen,
- “multiprocessing” zur parallelen Ausführung der Spielerprozesse,
- “asciimatics” zur Darstellung der Spieloberfläche im Terminal,
- “sys” zur Steuerung des Programmablaufs.

4. Funktionen und ihre Beschreibungen

4.1 generate_bingo_card

```
def generate_bingo_card(words, xaxis, yaxis):
    random.shuffle(words)
    card = [words[i * xaxis:(i + 1) * xaxis] for i in range(yaxis)]
    return card
```

Diese Funktion mischt die übergebene Liste von Wörtern und teilt sie in eine Bingokarte mit den Abmessungsparametern x(-Achse) und y(-Achse) auf.

4.2 display_bingo_cards

```
def display_bingo_cards(screen, card1, marked1, card2, marked2):
    screen.clear()
    screen.print_at("Spieler 1", 0, 0, colour=7)
    screen.print_at("Spieler 2", 40, 0, colour=7)

    for y, row in enumerate(card1):
```

```
        for x, word in enumerate(row):
            if marked1[y][x]:
                word = '\u0336'.join(word) + '\u0336'
                screen.print_at(word, x * 10, y * 2 + 2, colour=2)
            else:
                screen.print_at(word, x * 10, y * 2 + 2)

    for y, row in enumerate(card2):
        for x, word in enumerate(row):
            if marked2[y][x]:
                word = '\u0336'.join(word) + '\u0336'
                screen.print_at(word, x * 10 + 40, y * 2 + 2, colour=2)
            else:
                screen.print_at(word, x * 10 + 40, y * 2 + 2)

    screen.refresh()
```

Diese Funktion zeigt die Bingokarten für beide Spieler an und streicht markierte Wörter durch. Sie aktualisiert die Ausgabe auf dem Bildschirm und zeigt die gestrichenen Wörter mit der Farbe Grün an.

4.3 check_winner

```
def check_winner(marked):
    n = len(marked)
    for row in marked:
        if all(row):
            return True
    for col in range(n):
        if all(row[col] for row in marked):
            return True
    if all(marked[i][i] for i in range(n)) or all(marked[i][n-i-1] for i in range(n)):
        return True
    return False
```

Diese Funktion überprüft, ob eine der Gewinnbedingungen erfüllt ist. Dabei wird geprüft, ob eine komplette Reihe, eine Spalte oder eine Diagonale vorliegt. Liegt ein Gewinn vor, wird der Parameter *True* zurückgegeben.

4.4 player_game

```
def player_game(conn, player_id, card, xaxis, yaxis, log_file):
    marked = [[False] * xaxis for _ in range(yaxis)]

    def mark_word(word):
        for y, row in enumerate(card):
            for x, card_word in enumerate(row):
```

```
            if card_word == word:
                marked[y][x] = True
                log_action(log_file, f"Mark {card[y][x]} at ({x},{y})")
                return x, y
    return None, None

def log_action(log_file, action):
    with open(log_file, 'a') as f:
        f.write(f"{datetime.now()} {action}\n")

conn.send((player_id, "ready"))

while True:
    msg = conn.recv()
    if msg == "quit":
        conn.send((player_id, "quit"))
        break
    elif isinstance(msg, str):
        x, y = mark_word(msg)
        conn.send((player_id, "mark", x, y, msg))
        if check_winner(marked):
            conn.send((player_id, "win", x, y, msg))
            break

conn.close()
```

Diese Funktion führt die Spielaktivitäten für einen Spieler aus. Sie nutzt dabei die Funktion *mark_word*, um Wörter auf der Karte zu markieren, *log_action* dokumentiert markierte Wörter im Logs-Verzeichnis, überprüft auf Gewinn und sendet Nachrichten mit Hilfe von *conn.send* über eine Pipe-Verbindung zurück zum Host-Prozess, ob der Spieler das Spiel verlassen hat, ein Wort markiert hat oder gewonnen hat.

4.5 host_game

```
def host_game(screen, words, xaxis, yaxis, log_dir):
    card1 = generate_bingo_card(words, xaxis, yaxis)
    card2 = generate_bingo_card(words, xaxis, yaxis)
    parent_conn1, child_conn1 = Pipe()
    parent_conn2, child_conn2 = Pipe()
```

Diese Funktion ist das Herzstück des Spiels. Hierbei wird mit der Funktion *generate_bingo_card* die beiden Bingokarten, während mit Hilfe von *Pipe()* die Kommunikation zwischen den Prozessen gewährleistet wird.

```
    player_log1 = os.path.join(log_dir,
f"{datetime.now().strftime('%Y-%m-%d-%H-%M-%S')}-bingo-Player1.txt")
```

```
    player_log2 = os.path.join(log_dir,
f"{datetime.now().strftime('%Y-%m-%d-%H-%M-%S')}-bingo-Player2.txt")

    with open(player_log1, 'w') as f:
        f.write(f"{datetime.now()} Start des Spiels\n")
        f.write(f"{datetime.now()} Größe des Spielfelds: ({xaxis},{yaxis})\n")

    with open(player_log2, 'w') as f:
        f.write(f"{datetime.now()} Start des Spiels\n")
        f.write(f"{datetime.now()} Größe des Spielfelds: ({xaxis},{yaxis})\n")
```

Diese Zeilen Code erstellen je eine Log-Datei pro Spieler. In diesen wird die Startzeit des Spiels, sowie die Größe des Spielfelds eingetragen.

```
    marked1 = [[False] * xaxis for _ in range(yaxis)]
    marked2 = [[False] * xaxis for _ in range(yaxis)]
```

Diese beiden Zeilen erstellen zwei zweidimensionale Tabellen für die beiden Bingokarten der Spieler. Diese werden verwendet, um zu verfolgen, welche Wörter markiert wurden. Beispiel 2x2:
Spieler 1 Spieler 2
[False] [False] [False] [False]
[False] [False] [False] [True]
In diesem Beispiel wurde das Wort bei den Koordinaten x=2 und y=2 des Spielers 2 gezogen.

```
    p1 = Process(target=player_game, args=(child_conn1, 1, card1, xaxis, yaxis,
player_log1))
    p2 = Process(target=player_game, args=(child_conn2, 2, card2, xaxis, yaxis,
player_log2))
    p1.start()
    p2.start()
```

Hierbei werden zwei unabhängig voneinander Prozesse (*p1* und *p2*) gestartet und die Argumente *child_connX, X, cardX, xaxis, yaxis, player_logX* übergeben.

```
    try:
        ready_players = 0
        while ready_players < 2:
            if parent_conn1.poll():
                msg = parent_conn1.recv()
                if msg[1] == "ready":
                    ready_players += 1
            if parent_conn2.poll():
                msg = parent_conn2.recv()
                if msg[1] == "ready":
                    ready_players += 1
```

```
    display_bingo_cards(screen, card1, marked1, card2, marked2)

    while True:
        event = screen.get_event()
        if event and isinstance(event, KeyboardEvent):
            if event.key_code == ord('q'):
                parent_conn1.send("quit")
                parent_conn2.send("quit")
                break
            elif event.key_code == ord('m'):
                word = random.choice(words)
                screen.clear()
                display_bingo_cards(screen, card1, marked1, card2, marked2)
                screen.print_at(f"Gezogenes Wort: {word}", 0, screen.height - 2,
colour=Screen.COLOUR_YELLOW)
                screen.print_at("Bitte F drücken um fortzufahren.", 0, screen.height - 1,
colour=Screen.COLOUR_WHITE)
                screen.refresh()
                parent_conn1.send(word)
                parent_conn2.send(word)

        if parent_conn1.poll():
            msg = parent_conn1.recv()
            if msg[1] == "quit":
                screen.print_at("Spieler 1 hat das Spiel verlassen.", 0, screen.height - 3,
colour=Screen.COLOUR_RED)
                screen.refresh()
                break
            elif msg[1] == "mark":
                _, _, x, y, word = msg
                if x is not None and y is not None:
                    marked1[y][x] = True
            elif msg[1] == "win":
                screen.print_at(f"Spieler 1 gewinnt mit dem Wort: {msg[4]}!", 0, screen.height -
4, colour=Screen.COLOUR_GREEN)
                screen.refresh()
                time.sleep(3.0)
                screen.clear()
                screen.refresh()
                print("Der Sieger ist Spieler 1!")
                time.sleep(5.0)
                host_game(screen, words, xaxis, yaxis, log_dir)


        if parent_conn2.poll():
            msg = parent_conn2.recv()
            if msg[1] == "quit":
```

```
                screen.print_at("Spieler 2 hat das Spiel verlassen.", 0, screen.height - 3,
colour=Screen.COLOUR_RED)
                screen.refresh()
                break
            elif msg[1] == "mark":
                _, _, x, y, word = msg
                if x is not None and y is not None:
                    marked2[y][x] = True
            elif msg[1] == "win":
                screen.print_at(f"Spieler 2 gewinnt mit dem Wort: {msg[4]}!", 0, screen.height -
4, colour=Screen.COLOUR_GREEN)
                screen.refresh()
                time.sleep(3.0)
                screen.clear()
                screen.refresh()
                print("Der Sieger ist Spieler 2!")
                time.sleep(5.0)
                host_game(screen, words, xaxis, yaxis, log_dir)


        if event and isinstance(event, KeyboardEvent):
            if event.key_code == ord('f'):
                display_bingo_cards(screen, card1, marked1, card2, marked2)

    finally:
        p1.join()
        p2.join()
        parent_conn1.close()
        child_conn1.close()
        parent_conn2.close()
        child_conn2.close()
```

5. Hauptprogramm

```python
def main(screen, xaxis, yaxis, words_file, log_dir):
    try:
        with open(words_file, 'r') as f:
            words = [line.strip() for line in f.readlines()]

        if len(words) < xaxis * yaxis * 2:
            print("Nicht genügend Wörter in der Datei.")
            exit(1)

    except Exception as e:
        print(f"Fehler: {e}")
```

```
    while True:
        host_game(screen, words, xaxis, yaxis, log_dir)


if __name__ == "__main__":
    try:
        xaxis = int(input("Bitte eine Zahl für die x-Achse eingeben: "))
        yaxis = int(input("Bitte eine Zahl für die y-Achse eingeben: "))
    except ValueError:
        print("Bitte eine gültige Zahl eingeben.")
        exit(1)

    words_file = input("Bitte den Pfad zur Wortdatei eingeben: ")
    log_dir = input("Bitte das Verzeichnis für die Logs eingeben: ")

    if not os.path.exists(log_dir):
        print("Das angegebene Verzeichnis für die Logs existiert nicht.")
        exit(1)

    Screen.wrapper(main, arguments=[xaxis, yaxis, words_file, log_dir])
```

Das Hauptprogramm fragt den Benutzer nach den Dimensionen der Bingokarten sowie nach dem Pfad zur Wortdatei und dem Verzeichnis für die Log-Dateien. Es startet dann das Spiel mit diesen Parametern.

6. Benutzerschnittstelle

Das Spiel wird im Terminal ausgeführt und verwendet die `asciimatics`-Bibliothek zur Darstellung. Die Steuerung erfolgt über Tastatureingaben:
- `q`: Beendet das Spiel.
- `m`: Zieht ein neues Wort und markiert es auf den Karten.
- `f`: Aktualisiert die Anzeige der Bingokarten.

7. Anmerkungen und zukünftige Erweiterungen

Mögliche Erweiterungen dieses Programms könnten beinhalten:
- Unterstützung für mehr als zwei Spieler.
- Erweiterung der Benutzeroberfläche um mehr Interaktionsmöglichkeiten.
- Verbesserung der Anzeige und Bedienung für verschiedene Bildschirmgrößen.
- Netzwerkunterstützung für ein Spiel über mehrere Geräte hinweg.

8. Fazit

Dieses Bingo-Spiel in Python demonstriert die Verwendung mehrerer Bibliotheken zur Erstellung eines interaktiven und parallel ausgeführten Spiels. Es bietet eine solide Grundlage für weitere Entwicklungen und Anpassungen, um den Spielspaß und die Funktionalität zu erweitern.