Werkstück A

Alternative 3

(Buzzword-Bingo)

im Modul

„Betriebssysteme und Rechnernetze“

im Studiengang

„Wirtschaftsinformatik“

Vorgelegt von:

Bandura, Niclas (1512674)

Barea Walz, Miguel (1512616)

Dillemuth, Jan (1505007)

Fleischer, Maurice (1523322)

Turkmani, Niamatullah (1511114)

im Sommer-Semester 2024

am Fachbereich 2

der Frankfurt University of Applied Sciences

Erstprüfer: **Prof. Dr. Christian Baun**

Abgabedatum: **27.06.2024**

Inhaltsverzeichnis

1. Einführung

Diese Dokumentation beschreibt die Entwicklung und Funktionsweise des Buzzword-Bingo-Spiels, das mit der Textual-Bibliothek in Python erstellt wurde. Das Buzzword-Bingo-Spiel ist eine Kommandozeilenanwendung mit einer grafischen Oberfläche. Ziel dieses Projekts war es, ein interaktives Bingo-Spiel zu schaffen, bei dem zwei oder mehr Spieler auf einem Computer gegeneinander antreten können. Außerdem soll jeder Spieler ein eigener Prozess sein und über eine Interprozesskommunikation kommunizieren.

Das Spiel endet, sobald ein Spieler eine vollständige Zeile, Spalte oder Diagonale markiert und das Bingo bestätigt hat. Die Koordination zwischen den Spielern wird durch die Nutzung von Shared Memory aus der POSIX-Bibliothek ermöglicht, was die Interprozesskommunikation gewährleistet. Die Benutzeroberfläche wurde mit der Textual-Bibliothek gestaltet und bietet eine übersichtliche und intuitive Darstellung der Bingo-Karten und Spielaktionen. Darüber hinaus sorgen Logger dafür, dass alle wichtigen Ereignisse und Aktionen im Spiel nachvollziehbar aufgezeichnet werden.

Während der Entwicklung standen wir vor verschiedenen Herausforderungen, darunter die Implementierung einer robusten IPC und die Sicherstellung einer reibungslosen Benutzererfahrung. Das Projekt bot uns die Möglichkeit, verschiedene Konzepte der Softwareentwicklung praktisch anzuwenden und unser Wissen in Bereichen wie Konsolen-GUI-Design und Interprozesskommunikation zu vertiefen. Insgesamt ist das Buzzword-Bingo-Spiel ein Beispiel für die Kombination von Spaß und technischer Komplexität in einer einfachen, benutzerfreundlichen Anwendung.

2. Anforderungsanalyse

2.1. Funktionale Anforderungsanalyse

- **Spielermodus:** Das Buzzword-Bingo-Spiel soll eine Kommandozeilenanwendung sein die mindestens zwei Spieler unterstützen, die auf einem Computer gegeneinander antreten können.
- **Bingo-Karten:** Jeder Spieler erhält eine eigene Bingo-Karte, die basierend auf benutzerdefinierten Parametern wie Kartengröße und Wortdatei generiert wird.
- **Spielende:** Das Spiel endet, sobald ein Spieler eine vollständige Zeile, Spalte oder Diagonale markiert und das Bingo bestätigt hat.
- **Interprozesskommunikation:** Jeder Spieler wird als eigenständiger Prozess dargestellt. Die Prozesse kommunizieren über Shared Memory aus der POSIX-Bibliothek.

2.2. Nicht-funktionale Anforderungen

- **Robustheit:** Das Spiel muss robust gegenüber falschen Eingaben sein, um eine fehlerfreie Spielerfahrung zu gewährleisten.
- **Leistungsfähigkeit:** Die Kommunikation zwischen den verschiedenen Prozessen muss effizient und zuverlässig sein, um eine reibungslose Spielerfahrung sicherzustellen.
- **Plattformabhängigkeit:** Aktuell funktioniert das Spiel nur auf Linux-Systemen aufgrund der Nutzung von POSIX-IPC.

- **Benutzeroberfläche:** Die Benutzeroberfläche soll eine klare und intuitive Darstellung der Bingo-Karten und Spielaktionen bieten, um möglichst benutzerfreundlich zu sein.

3. Architektur und Design

Das Buzzword-Bingo-Spiel besteht aus einem Spieler, der ein Spiel erstellt (Host) und weiteren Spielern. Der Host initiiert das Spiel, legt den Spielnamen, die Wörterdatei und die Spielfeldgröße fest. Spieler können sich jederzeit dem Spiel anschließen und interagieren über Shared Memory miteinander.

Die Benutzeroberfläche wird mit der Textual-Bibliothek gestaltet und bietet eine strukturierte Darstellung der Bingo-Karten und Spielaktionen in einer Konsolen-GUI. Die GUI umfasst ein Grid-Layout mit Labels und Buttons, die die Benutzerinteraktion erleichtern.

Zusätzlich sorgen ein IPC-Logger und ein Spiel-Logger dafür, dass alle wichtigen Ereignisse und Aktionen im Spiel nachvollziehbar aufgezeichnet werden. Der IPC-Logger speichert, die geschehen des Shared Memory in einer eigenen Datei und der Spiel Logger speichert jede Interaktion des Spiels auch in einer eigenen Datei

Während der Entwicklung lag der Fokus auf einer robusten IPC und einer benutzerfreundlichen GUI, um eine reibungslose und intuitive Spielerfahrung zu gewährleisten. In den folgenden Kapiteln werden die einzelnen Komponenten und deren Funktionsweise detailliert beschrieben.

4. Installation

Um als Team problemlos am Buzzword-Bingo-Spiel zu arbeiten, nutzen wir Anaconda, eine Open-Source-Distribution für Python und R. Anaconda vereinfacht die Installation und Verwaltung von Python sowie zahlreichen Bibliotheken, ohne dass Abhängigkeitskonflikte auftreten.

Mit Anaconda können wir einfach eine spezielle Umgebung für das Buzzword-Bingo-Spiel erstellen. Diese isoliert alle benötigten Abhängigkeiten von anderen Python-Installationen auf unserem System. Innerhalb dieser Umgebung installieren wir die erforderlichen Bibliotheken wie die Textual-Bibliothek und POSIX IPC.

Durch die Verwendung von Anaconda stellen wir sicher, dass alle Teammitglieder eine konsistente Entwicklungsumgebung nutzen, was die Zusammenarbeit erleichtert und mögliche Konflikte minimiert. Dies gewährleistet eine zuverlässige Plattform zur Verwaltung von Bibliotheken und Paketen, was die Implementierung des Spiels konsistent und stabil macht.

4.1. GitHub

GitHub wurde als zentrale Plattform für die Verwaltung und Zusammenarbeit an unserem Buzzword-Bingo-Projekt genutzt. Das Repository auf GitHub ermöglichte es uns, unseren Code zu speichern, zu versionieren und gemeinsam daran zu arbeiten. Während der Entwicklung traten gelegentlich Herausforderungen auf, wie zum Beispiel Konflikte bei der Zusammenführung von Änderungen. Diese wurden durch koordinierte Teamarbeit und die Anwendung von Git-Befehlen zur Konfliktlösung erfolgreich gelöst.

Direkten Zugang zu unserem Projektcode - Link zum GitHub-Repository: https://github.com/FreezingHyper/BuzzwordBingo

4.2. Hauptspiel

4.2.1. Vorstellung

Die Bingo-App wurde mit der Textual-Bibliothek in Python entwickelt. Die textbasierte Anwendung bietet eine grafische Benutzeroberfläche (GUI), die es Spielern ermöglicht, ein Bingo-Spiel zu erstellen, beizutreten, Bingo-Felder anzuklicken, Wörter zu generieren und ein Bingo zu bestätigen.

4.2.2. Hauptkomponenten der Anwendung

4.2.2.1. Benutzeroberfläche

Die Konsolen-GUI der Anwendung umfasst verschiedene interaktive Elemente wie Buttons und Labels. Die Hauptaufgabe der GUI ist es, den Spielern eine intuitive und visuell ansprechende Möglichkeit zu bieten, das Bingo-Spiel zu spielen.

- **Header:** Zeigt Information zu Spielnamen, die IPC_ID und die SpielerID an.
- **Labels:** Anzeigen von Texten, z.B. aktuelles Wort, Fehlermeldungen, Bingo-Status.
- **Buttons:** Interaktive Elemente, die vom Benutzer gedrückt werden können, z.B. zur Generierung eines Wortes oder zur Bestätigung eines Bingos.
- **Static:** Grid-Container für die Bingo-Felder.

4.2.2.2. Erstellung der Bingo-Felder

Die Bingo-Felder werden als Buttons in einem Grid dargestellt. Die Größe des Grids (3x3 bis 7x7) wird zu Beginn des Spiels festgelegt. Bei Spielfeldern mit ungerader Größe (5x5 oder 7x7) ist das mittlere Feld ein Joker-Feld und wird automatisch durchgestrichen und deaktiviert. Außerdem ist der Status eines Feldes immer farbig gekennzeichnet. Ist es nicht gedrückt worden so leuchtet es Grün. Ist es gedrückt worden leuchtet es Rot und das Wort in ihm ist durchgestrichen. Eine Hover-Funktion ist auch eingebaut, bei der das Feld dann grau aufleuchtet.

4.2.2.3. Überprüfung von Bingo

Die App überprüft, ob ein Bingo vorliegt. Ein Bingo ist eine vollständige horizontale, vertikale oder diagonale Linie von durchgestrichenen Feldern. Wenn ein Bingo erkannt wird, wird der entsprechende Bingo-Button zur Bestätigung eines Bingos erkenntlich markiert (Gelb) und der Spieler kann das Bingo bestätigen. Daraufhin wird in den Shared Memory geschrieben, dass der Spieler gewonnen hat.

4.2.2.4. Event-Handler

Event-Handler sind Methoden, die auf Benutzeraktionen reagieren. In der Bingo-App gibt es verschiedene Event-Handler, die auf Klicks von Buttons reagieren und entsprechende Aktionen ausführen.

- **Wort streichen:** Ändert den Status des gedrückten Buttons und überprüft, ob ein Bingo vorliegt.
- **Zufallswort generieren:** Wählt ein zufälliges Wort aus der Liste der Buzzwords und zeigt es an.

- **Bingo bestätigen:** Überprüft, ob ein Bingo vorliegt und aktualisiert den Gewinn-Status.
- **Beenden:** Beendet das Spiel und schließt die Anwendung.
- **Spiel abbrechen:** Beendet das Spiel und gibt den Speicher frei.

4.2.2.5. Aktualisierungen in regelmäßigen Intervallen

Um die Interprozesskommunikation (IPC) zu realisieren, haben wir uns für die Nutzung von POSIX genauer Shared Memory entschieden (siehe Kapitel 4.3) und setzen dabei auf aktives Warten. Diese Entscheidung stellt zwar nicht die ideale Lösung dar, ist jedoch in unserem Fall aus folgenden Gründen notwendig:

- **Häufige Aktualisierungsintervalle:** Um das Spiel flüssig zu gestalten und die Zusammenarbeit mehrerer Prozesse über Shared Memory zu ermöglichen, ist eine häufige Abfrage der Daten notwendig. Diese Abfrage könnte durch Sleep-Zustände reduziert werden, indem der sendende Prozess ein Weck-Signal an die anderen Prozesse sendet. Um dies zu realisieren, wird eine Liste mit allen beteiligten Prozess IDs benötigt. Durch die Möglichkeit des nachträglichen Beitretens in eine Runde, muss diese Liste dynamisch gepflegt werden. Als Konsequenz muss der Inhalt dieser Liste kontinuierlich gelesen werden, um sicherzustellen, dass alle Prozesse geweckt werden. Somit würde sich das dauernde Aktualisieren lediglich verschieben und nicht beseitigen. Zusätzlich wäre ein größerer Verwaltungsaufwand mit der alternativen Umsetzung verbunden, was den Entschluss stützt, die erste Form des kontinuierlichen Aktualisierens zu wählen.
- **Vermeidung von Prozess-Freeze:** Außerdem ermöglicht das aktive Warten, dass das Einfrieren der Prozesse aufgrund von Wartezuständen vermieden wird. Dadurch wird eine kontinuierliche und flüssige Darstellung der Informationen in der GUI sichergestellt und die konstante Reaktionsfähigkeit der Prozesse gewährleistet.

Zusammengefasst haben wir uns aus den oben genannten Gründen für aktives Warten entschieden, um sicherzustellen, dass alle Prozesse die erforderlichen Daten haben, um effektiv zu arbeiten.

4.2.3. Ergebnis

Die GUI der Bingo-App bietet eine interaktive und visuell ansprechende Möglichkeit, Bingo zu spielen. Durch die Verwendung der Textual-Bibliothek wird eine moderne und responsive Benutzeroberfläche ermöglicht, die einfach zu bedienen ist. Die strukturierte Aufteilung in verschiedene Komponenten erleichtert die Wartung und Erweiterung der Anwendung.

4.3. Interprozesskommunikation (IPC)

4.3.1. Vorstellung

Die Klasse "SpielIPC" ermöglicht die Interprozesskommunikation zwischen verschiedenen Spielprozessen über einen Shared Memory. Dies erlaubt eine effiziente Datenübertragung und Synchronisation zwischen den Spielinstanzen. Die Klasse ermöglicht dem Bingo Spiel:

- Die Verbindung zu trennen
- Den Speicher freizugeben
- Ein Bingo zu senden / lesen
- Den Spielstart zu senden / lesen
- Den Dateipfad zu senden / lesen
- Die Spielgröße zu senden / lesen
- Gerufene Wörter zu senden / lesen

4.3.2. Speicherdesign

Um die IPC zu realisieren, wurde ein Shared Memory verwendet. Dabei wird eine Liste mit Daten serialisiert und in das Shared Memory geschrieben. Die verschiedenen Positionen haben folgende Bedeutungen:

Position 0: Bingo Status

Position 1: Dateipfad

Position 2: Spielfeld Größe

Position 3: Wörter

Position 4: Start-Status

Position 5: IPC-ID

4.3.3. Umsetzung

Um die richtigen Informationen jederzeit aus dem Shared Memory lesen zu können, wird die serialisierte Liste herausgelesen. Anschließend wird diese deserialisiert und die benötigte Information aus der entsprechenden Position abgerufen. Wenn eine neue Information gespeichert werden soll, dann wird eine neue Liste erstellt. An die entsprechende Position wird die neue Information gespeichert und die anderen Positionen mit bereits gespeicherten Informationen überschrieben. Anschließend wird diese geupdatete Liste in den Shared Memory geschrieben. Dabei überschreibt die neue Liste, die Alte.

4.3.4. Herausforderungen und Probleme bei der Umsetzung

Die gespeicherten Daten im Shared Memory müssen immer aktuell sein. Wenn die neuen Daten z.B. in Form von Bytestrings an den bisherigen Inhalt angefügt werden, so ist es schwierig immer die richtigen Informationen zu lesen. Die Lösung war das Speichern in ein Listen-Objekt, welches serialisiert wird und das Alte überschreibt.

Beim Programmieren wird oft kein Wert auf der Konsole ausgegeben, weil vieles in den Speicher geschrieben und gelesen wird, allerdings keinen Output für den User bringt. Das machte das Testen der Methoden umständlich.

4.3.5. Ausführliche Erklärung

Eine umfassendere Darstellung der Umsetzung der IPC sowie eine genaue Erklärung aller Methoden, die verwendet werden, findet sich im Anhang (siehe Anhang).

4.4. Logger

4.4.1. Player Logger

4.4.1.1. Vorstellung

Der Player-Logger ist ein zentraler Bestandteil für die Dokumentation der Aktivitäten eines Spielers im Spiel. Er zeichnet den Zeitpunkt des Beitritts, die Aktionen auf dem Spielfeld und den Ausgang des Spiels auf. Für jeden Spieler wird anhand der Prozess-ID eine individuelle Log-Datei erstellt, die im entsprechenden Verzeichnis gespeichert wird.

4.4.1.2. Verwendete Module

Für die Implementierung des Player-Loggers wurden die Bibliotheken datetime und os verwendet. datetime dient zur Nutzung von Zeitstempeln in Log-Ereignissen, während os das Erstellen und Zugreifen auf Dateien ermöglicht.

4.4.1.3. Implementierung und Funktionen

Beim Erstellen eines Player-Logger-Objekts wird überprüft, ob ein Verzeichnis für die Log-Dateien existiert; falls nicht, wird es erstellt. Anschließend wird ein einzigartiger Dateiname generiert, der das aktuelle Datum und die Spieler-ID enthält. Diese Datei wird zum späteren Loggen verwendet. Der Player-Logger verfügt über mehrere Methoden zur Protokollierung von Spielereignissen. Diese Methoden erfassen den Zeitpunkt, zu dem ein Spieler dem Spiel beitritt, das Ende des Spiels, durchgestrichene und demarkierte Wörter, die Möglichkeit eines „Bingo“ und das Spielergebnis. Die Protokollierung erfolgt präzise, indem die aktuelle Zeit für jeden Eintrag erfasst wird, was eine detaillierte Nachverfolgung und Analyse des Spielverlaufs ermöglicht.

4.4.2. IPC-Logger

4.4.2.1. Vorstellung

Der IPC-Logger dokumentiert alle Ereignisse der Interprozesskommunikation (IPC) im Spiel. Er ist dem Player-Logger ähnlich, unterscheidet sich jedoch im methodischen Aufbau. Der IPC-Logger vereinfacht die Erstellung neuer Log-Ereignisse durch eine Hauptmethode, die spezifische Ereignisnachrichten anhängt.

4.4.2.2. Verwendete Module

Auch beim IPC-Logger werden die Bibliotheken datetime und os genutzt. Diese ermöglichen die Erfassung von Zeitstempeln und den Dateizugriff sowie das Erstellen von Verzeichnissen und Dateien.

4.4.2.3. Implementierung Funktion

Bei der Initialisierung eines IPC-Logger-Objekts werden der Spielname und die Spiel-ID übergeben. Diese Informationen dienen zur Generierung eines eindeutigen Dateinamens, der im Verzeichnis "ipc_logs" gespeichert wird. Sollte dieses Verzeichnis nicht existieren, wird es erstellt. Der Logger besitzt im Gegensatz zum Player-Logger eine allgemeine Log-Methode, die das aktuelle Datum und die Uhrzeit erfasst und eine Nachricht in die Log-Datei schreibt. Diese Methode bildet die Grundlage für die Protokollierung von IPC-Ereignissen, wie die Erstellung des Spiels, das Hinzufügen neuer Wörter zur Wortliste, das Rufen von „Bingo“, das Setzen von Dateipfaden und Spielfeldgrößen, das Lesen von Daten aus dem Shared Memory sowie das Löschen des Spiels und das Trennen der Verbindung.

4.4.3. Herausforderungen und Probleme

Beide Logger-Systeme standen öfters vor Problemen und Herausforderungen. Beim Player-Logger war die Erstellung von Dateien mit individuellem Dateipfaden und variablen Inhalten komplex und fehleranfällig. Oft führten kleine Fehler dazu, dass das gesamte Testprogramm nicht funktionierte. Beim IPC-Logger stellte der gleichzeitige Zugriff mehrerer Prozesse auf eine Log-Datei ein Problem dar. Dies wurde durch die Einführung einer dedizierten ID gelöst, die das Spiel eindeutig identifiziert und den Zugriff mehrerer Prozesse ermöglicht. Darüber hinaus war es eine Herausforderung zu bestimmen, welche Ereignisse für die spätere Analyse überhaupt relevant sind, was ständige Anpassungen der Log-Methoden erforderte.

5. Bekannte Probleme

Es gibt zwei bekannte Probleme, die bei der Nutzung des Buzzword-Bingo-Spiels berücksichtigt werden müssen. Erstens funktioniert das Spiel aktuell nur auf Linux-Systemen. Dies liegt daran, dass unserer IPC unter der POSIX-Bibliothek nur auf Linux-Systemen läuft. Es ist wichtig, diese Einschränkung bei der Nutzung des Spiels zu berücksichtigen, da das Spiel momentan auf anderen Betriebssystemen nicht läuft.

Zweitens muss bei der Verwendung einer eigenen Buzzwords-Datei die Formatierung stimmen. Die Wörter müssen durch Zeilenumbrüche (Enter) getrennt sein, um korrekt eingelesen und auf den Bingo-Karten verteilt zu werden. Eine falsche Formatierung kann dazu führen, dass die Wörter nicht richtig erkannt werden und das Spiel nicht korrekt startet.

6. Start des Programms

Um das Programm zu starten wird ein Aufruf der Game.py über die Konsole benötigt. Dabei muss der Start mit dem Befehl „python“ erfolgen. Wenn das Programm in der Konsole gestartet ist, kommt es zur Auswahl des Spielnamen, mit welchem sich verbunden werden soll. Existiert ein Spiel bereits kann dieses betreten werden. Andernfalls erfragt das Programm die Parameter, die es zum Erstellen eines Spiels braucht.

7. Nutzung des Programms

Die Benutzeroberfläche des Buzzword-Bingo-Spiels ist in mehrere Bereiche unterteilt. Im oberen Bereich befindet sich der Header. Darunter sind zwei Buttons, die nur für den Host sichtbar sind: "Wort generieren" und "Spiel abbrechen". Die Bingo-Karte in der Mitte des Bildschirms ist ein Gitter aus Feldern mit Buzzwords. Direkt über ihr befindet sich immer das aktuell gerufene Wort. Im unteren Bereich gibt es zwei Buttons: "Bingo bestätigen" und "Quit". Ganz unten befindet sich dann eine Liste der schon gerufenen Buzzwords. Spieler die neu dazugekommen sind, können so auch schon gerufene Wörter auswählen. Mit den genannten Komponenten lässts sich das Spiel problemlos bedienen. Hier ein Beispiel Ablauf:

- **Spiel starten:** Der Spieler wählt entweder ein bestehendes Spiel aus oder erstellt ein neues Spiel.
- **Buzzword-Datei einlesen:** Der Spieler kann entweder die voreingestellte Datei verwenden oder einen neuen Dateipfad angeben.
- **Bingo-Felder generieren:** Die App generiert ein Bingo-Grid basierend auf der angegebenen Größe und füllt es mit den Wörtern aus der Datei.
- **Wort generieren:** Ein zufälliges Wort wird aus der Liste der Buzzwords ausgewählt und angezeigt. Nur der Host sieht den "Wort generieren Button" und kann ihn betätigen.
- **Felder streichen:** Spieler klicken auf die Felder, um sie durchzustreichen oder zurückzusetzen.
- **Bingo überprüfen:** Die App überprüft automatisch, ob ein Bingo vorliegt.
- **Bingo bestätigen:** Der Spieler kann ein Bingo bestätigen, wenn alle Bedingungen erfüllt sind.
- **Spiel beenden:** Das Spiel kann jederzeit beendet oder abgebrochen werden. Ist ein Bingo bestätigt worden, so wird es automatisch beendet.

Wenn das Programm ohne das Eintretens eines Bingos beendet werden soll, so ist darauf zu achten, dass der Host-Prozess die Option „Spiel abbrechen“ wählt, um Ressourcen freizugeben.

8. Zeitliche Planung

Für die Entwicklung des Buzzword-Bingo-Spiels haben wir eine strukturierte Planung und zeitliche Aufteilung vorgenommen. Dabei haben wir die Plattform Notion intensiv genutzt, um unsere Aufgaben zu koordinieren und den Projektfortschritt zu verfolgen. Die Aufgaben wurden fair und strukturiert an die Teammitglieder verteilt, wodurch jeder effektiv in seinem Bereich arbeiten konnte, ohne auf andere warten zu müssen. Dies ermöglichte eine reibungslose Entwicklung der GUI sowie der Implementierung der IPC-Funktionalitäten und der Logger. Durch diese strukturierte Planung und Aufgabenverteilung konnten wir die Entwicklungszeit optimieren und Engpässe vermeiden, was zu einem erfolgreichen Abschluss des Buzzword-Bingo-Projekts führte.

Status	Titel	Beschreibung	Zuständigkeit	Erledigen am:
Erledigt	Einarbeiten in benötigte Bibliotheken und grobe Planung		Niclas Miguel Nima Jan Maurice	15. April 2024 → 15. Mai 2024
Erledigt	Projektskizze		Niclas Miguel Nima Jan Maurice	30. April 2024 → 6. Mai 2024
Erledigt	Einteilung der Zuständigkeiten		Niclas Nima Miguel Jan Maurice	16. Mai 2024
Erledigt	Anaconda	Base einrichten	Miguel Niclas Nima Jan Maurice	16. Mai 2024
Erledigt	IPC	Detaillierte Methodenangaben in IPC Beispielbenutzung	Niclas	17. Mai 2024 → 22. Mai 2024
Erledigt	Logger	Detaillierte Methodenangaben, wenn man To-Do öffnet	Maurice	17. Mai 2024 → 22. Mai 2024
Erledigt	BingoGame Main	Entwicklung der Methoden. Entwicklung versch. Konsolen-GUIs mit unterschiedlichen Bibliotheken.	Nima Miguel Jan	17. Mai 2024 → 22. Mai 2024
Erledigt	Zwischenbesprechnung	Wenn Teile bereits fertig, Kapazitäten umverteilen	Niclas Miguel Nima Jan Maurice	23. Mai 2024
Erledigt	Projektschnipsel Zusammensetzen		Nima	24. Mai 2024 → 29. Mai 2024
Erledigt	Bug Fixing		Niclas Miguel Nima Jan Maurice	30. Mai 2024 → 6. Juni 2024
In Bearbeitung	Dokumentation & Präsi		Niclas Miguel Nima Jan Maurice	6. Juni 2024 → 25. Juni 2024
Nicht begonnen	Re-Read ÖFFNEN		Niclas Miguel Nima Jan Maurice	24. Juni 2024 → 26. Juni 2024
Nicht begonnen	Puffer			27. Juni 2024 → 29. Juni 2024
Nicht begonnen	Projekt Abgabe		Niclas Miguel Nima Jan Maurice	30. Juni 2024

Figure 1: Zeitplan in Notion

Die zeitliche Planung wurde unter Berücksichtigung der Projektphasen und der Verfügbarkeit der Teammitglieder durchgeführt. Regelmäßige Meetings wurden abgehalten, um den Fortschritt zu besprechen, Probleme zu lösen und neue Ziele festzulegen. Dadurch konnten wir sicherstellen, dass der Zeitplan eingehalten und das Projekt termingerecht abgeschlossen wurde. Während des Entwicklungsprozesses traten Herausforderungen auf, die wir durch flexible Zeitplanung und effektive Kommunikation bewältigen konnten. Notion erwies sich erneut als nützliches Tool zur Dokumentation von Zeitplanänderungen und zur Planung alternativer Lösungen, falls erforderlich.

9. Fazit

Das Buzzword-Bingo-Projekt war eine wertvolle Gelegenheit, verschiedene Softwareentwicklungskonzepte und -technologien anzuwenden. Trotz einiger bekannter Einschränkungen wurde erfolgreich ein funktionsfähiges Spiel entwickelt, das innerhalb des geplanten Zeitrahmens abgeschlossen wurde. Das Spiel vereint technische Komplexität mit Benutzerfreundlichkeit:

Entwickelt in Python als Kommandozeilenanwendung mit grafischer Oberfläche, ermöglicht es Spielern interaktiv Bingo auf einem Computer zu spielen. Jeder Spieler agiert als eigenständiger Prozess, der effizient über Shared Memory kommuniziert. Die robuste Implementierung der Interprozesskommunikation und die intuitive Benutzeroberfläche gewährleisten eine reibungslose Spielerfahrung. Die Verwendung von Anaconda zur Umgebungsverwaltung erleichtert die Installation und Wartung erforderlicher Bibliotheken. GitHub diente als zentrale Plattform für die gemeinsame Entwicklung.

Die Spielarchitektur des Buzzword-Bingo-Spiels wird durch ein gut strukturiertes GUI-Design, nachvollziehbare Logging-Mechanismen und die effektive Nutzung von IPC unterstützt. Herausforderungen wie die Handhabung von IPC und die Optimierung des GUI wurden erfolgreich durch praktische Anwendung und Teamarbeit bewältigt. Insgesamt bietet das Buzzword-Bingo-Spiel eine gelungene Kombination aus Technologie und Unterhaltung.

10. Anhang

10.1. Ausführliche Erklärung der Interprozesskommunikation

10.1.1. Vorstellung

Die Klasse „SpielIPC“ ermöglicht die Interprozesskommunikation zwischen verschiedenen Spielprozessen über einen Shared Memory. Dies erlaubt eine effiziente Datenübertragung und Synchronisation zwischen den Spielinstanzen. Die Klasse ermöglicht dem Bingo Spiel:

- Die Verbindung zu trennen
- Den Speicher freizugeben
- Ein Bingo zu senden / lesen
- Den Spielstart zu senden / lesen
- Den Dateipfad zu senden / lesen
- Die Spielgröße zu senden / lesen
- Gerufene Wörter zu senden / lesen

10.1.2. Verwendete Module

- Um die IPC zu realisieren, wurden folgende Bibliotheken verwendet:
- Posix_ipc – Ermöglicht die Erstellung eines Shared Memory in Python
- Mmap – Dient der Speicherabbildung und ermöglicht den direkten Zugriff auf den Shared Memory
- Pickle – Ermöglicht das Serialisieren von Objekten in Bytefolgen, welche im Shared Memory gespeichert werden können
- Random – Erstellung einer Zufallszahl für die IPC-ID
- IPC_Logger – Eigen erstelltes Modul zur Protokollierung von IPC-Ereignissen

10.1.3. Speicherdesign

Um die IPC zu realisieren, wurde ein Shared Memory verwendet. Dabei wird eine Liste mit Daten serialisiert und in das Shared Memory geschrieben. Die verschiedenen Positionen haben folgende Bedeutungen:

Position 0: Bingo Status

Position 1: Dateipfad

Position 2: Spielfeld Größe

Position 3: Wörter

Position 4: Start-Status

Position 5: IPC-ID

Die Listen, welche während der IPC-Benutzung im Shared Memory als Bytefolgen geschrieben werden, überschreiben alte Bytes. Solange die Bytefolgen ausschließlich

länger werden, führt dies zu keinen Problemen. Wäre die neu geschriebene Bytefolge kürzer als die zuvor Gespeicherte, so würde nur der erste Teil überschrieben werden. Die restlichen Bytes bleiben bestehen und die Bytefolge kann nicht mehr deserialisiert werden. Es ist sichergestellt, dass keine Informationen aus der Liste gelöscht werden und lediglich neue hinzukommen. Somit wird auch die Bytefolge ausschließlich länger und die alte immer vollständig überschrieben

10.1.4. Umsetzung

10.1.4.1. Initialisierung

10.1.4.1.1. Konstruktor

```
class SpielIPC:
    def __init__(self, SpielName,ProzessID):
        self.SpielName = SpielName
        self.ProzessID = ProzessID
        self.Connection = self._connect()
        self.IPCLogger=IPCLogger(SpielName,self.getIPC_ID())
```

Der Konstruktor wird mit 2 Argumenten aufgerufen: den Spielnamen, welchen der Spieler im Bingo Spiel selbst wählt und die Prozess ID des Spiels, welche für jeden Prozess eindeutig ist. Die Shared Memory Verbindung wird über die _connect Methode (sieht nachfolgend) hergestellt und gespeichert. Außerdem wird ein IPC Logger erstellt, welcher spezifisch für den übergebenden Spielnamen und die generierte IPC ID (siehe nachfolgend) ist.

10.1.4.1.2. Verbindung zum Shared Memory (_connect)

```
def _connect(self):
        shared_memory = posix_ipc.SharedMemory
        (name="BuzzWordBingo_"+self.SpielName, size=1024,
        flags=posix_ipc.O_CREAT)
```

Die _connect Methode wird einmalig im Konstruktor aufgerufen. Dank der Flag wird ein neues Shared Memory erstellt, wenn es noch keinen Bereich mit dem Namen gibt, ansonsten wird eine Verbindung hergestellt. Der Name setzt sich aus einem fixen Präfix und dem gewählten Spielnamen des Benutzers zusammen. Es wird ein Shared Memory mit einer Speichergröße von 1024 Bytes verwendet.

Zur Speicherung der Daten wird eine serialisierte Liste verwendet, welche aus dem Shared Memory gelesen und hineingeschrieben wird. Sollte das Memory neu erstellt worden sein, so ist der Inhalt leer, was später zu Problemen beim Lesen führen kann. Daher muss sichergestellt werden, dass immer eine Liste vorhanden ist. Sollten noch keine Daten gespeichert wurden sein, so ist diese eben leer.

```
        try:
            shared_memory_mmap = mmap.mmap
            (shared_memory.fd, shared_memory.size,
            mmap.MAP_SHARED, mmap.PROT_READ)
            listeAlsCode =
shared_memory_mmap.read(shared_memory.size)
            shared_memory_mmap.close()
```

```
            speicherListe = pickle.loads(listeAlsCode)
```

Es wird ein mmap Objekt erstellt, welches mithilfe des Dateideskriptors (shared_memory.fd) auf den Speicherbereich zugreift. Außerdem wird die Größe des Shared Memory übergeben. Mit mmap.MAP_Shared wird sichergestellt, dass mehrere Prozesse den Shared Memory gleichzeitig benutzen können und das PROT_READ Argument gibt die Leserechte an das mmap Objekt. Anschließend werden die Bytes aus dem Shared Memory gelesen und in „listeAlsCode“ gespeichert. Die mmap-Verbindung wird geschlossen. Kann die Pickle Bibliothek diese Bytefolge deserialisieren, so ist bereits eine Liste gespeichert und keine weiteren Aktionen müssen erfolgen.

```
        except pickle.UnpicklingError:
            listeAlsCode = pickle.dumps
            (["", "", "", "","",str(random.randint(0,1000000))])
            shared_memory_mmap = mmap.mmap
            (shared_memory.fd, shared_memory.size,
             mmap.MAP_SHARED, mmap.PROT_WRITE)
            shared_memory_mmap.write(listeAlsCode)
            shared_memory_mmap.close()

        return shared_memory
```

Ist die Bytefolge allerdings leer, so entsteht ein pickle.UnpicklingError und es muss sichergestellt werden, dass eine neue Liste gespeichert wird. Eine Liste wird erstellt, und mithilfe von Pickle serialisiert und als Bytefolge in „listeAlsCode“ gespeichert. Diese Liste hat 6 Positionen, wobei lediglich die letzte initial mit einem Wert belegt wird.

Die IPC-ID (Position 6) wird bei der Erstellung des Shared Memory als zufällige Zahl zwischen 0-1000000 erstellt und bleibt bis zum Freigeben des Speichers gleich. Mithilfe dieser ID ist es dem IPC Logger möglich, auch bei unterschiedlichen Prozessen in die gleiche Log Datei zu schreiben.

Nach dem Speichern der Liste in der Variablen wird diese in den Shared Memory geschrieben. Zum vorherigen Lesen unterscheiden sich nur die Zugriffsrechte, welche nun mit PROT_WRITE Schreibrechte sind. Nach dem Schreiben wird die Verbindung getrennt und der Shared Memory zurückgegeben.

10.1.4.2. Grundmethoden zum lessen und schreiben

Nachfolgende Methoden dienen nur der klasseninternen Benutzung und ermöglichen das Abrufen der aktuell gespeicherten Liste sowie das Schreiben das einer Neuen.

10.1.4.2.1. Lesen (_read)

```
def _read(self):
        shared_memory = self.Connection
        shared_memory_mmap = mmap.mmap
        (shared_memory.fd, shared_memory.size,
         mmap.MAP_SHARED, mmap.PROT_READ)
        listeAlsCode = shared_memory_mmap.read(shared_memory.size)
```

```
        speicherListe = pickle.loads(listeAlsCode)
        shared_memory_mmap.close()
        return speicherListe
```

Die initial gespeicherte Shared Memory Connection wird in eine andere Variable gespeichert (dient der übersichtlichen Darstellung). Anschließend wird wie in 1.2. der Speicher über eine mmap-Verbindung gelesen, deserialisiert, die Verbindung geschlossen und Liste zurückgegeben.

10.1.4.2.2. Schreiben (_write)

```
def _write(self,speicherListe):
        listeAlsCode = pickle.dumps(speicherListe)
        shared_memory = self.Connection
        shared_memory_mmap = mmap.mmap
        (shared_memory.fd, shared_memory.size,
         mmap.MAP_SHARED, mmap.PROT_WRITE)
        shared_memory_mmap.write(listeAlsCode)
        shared_memory_mmap.close()
```

Dieser Methode wird eine Liste übergeben, welche im Shared Memory gespeichert werden soll. Dazu wird die Liste wie in 1.2. serialisiert und gespeichert.

10.1.4.3. Methoden zum externen Aufrufen

Diese Methoden werden aktiv im Bingo Spiel benutzt und greifen größtenteils auf die Grundmethoden zu.

10.1.4.3.1. Methode: CheckIfBingo()

```
def checkIfBingo(self):
        speicherListe = self._read()
        self.IPCLogger.logCheckIfBingo(self.ProzessID)
        if speicherListe[0] == "Bingo":
            return True
        else:
            return False
```

Die Liste wird mithilfe von self._read() aus dem Shared Memory gelesen. Steht an der Position 0 „Bingo“ wird True zurückgegeben, andernfalls False. Außerdem wird eine Log-Nachricht geschrieben.

10.1.4.3.2. Methode: getDateipfad()

```
  def getDateipfad(self):
        speicherListe = self._read()
        self.IPCLogger.logGetDateipfad(self.ProzessID)
        return speicherListe[1]
```

Die Liste wird mithilfe von self._read() aus dem Shared Memory gelesen. Die Position 1 wird zurückgeben und eine Log-Nachricht zurückgegeben.

10.1.4.3.3. Methode: getGroesse()

```
def getGroesse(self):
        speicherListe = self._read()
```

```
        self.IPCLogger.logGetGröße(self.ProzessID)
        return speicherListe[2]
```

Die Liste wird mithilfe von self._read() aus dem Shared Memory gelesen. Die Position 2 wird zurückgeben und eine Log-Nachricht zurückgegeben.

10.1.4.3.4. Wörter lessen

10.1.4.3.4.1. Methode: getWortString()

```
def getWortString(self):
        speicherListe = self._read()
        self.IPCLogger.logReadWortliste(self.ProzessID)
        return speicherListe[3]
```

Die Liste wird mithilfe von self._read() aus dem Shared Memory gelesen. Die Position 3 wird zurückgeben und eine Log-Nachricht zurückgegeben.

10.1.4.3.4.2. Methode: _getWortListe()

Diese Methode könnte einen externen Nutzen haben, wurde allerdings in dieser Umsetzung des Bingo Spiels nicht benötigt. Allerdings wird diese von der nachfolgen Methode benutzt.

```
def _getWortListe(self):
        wortString = self.getWortString()
        wortListe = wortString.split(";")
        return wortListe
```

Der gespeicherte Wort-String wird mithilfe von self._getWortString() abgerufen und in einem List Objekt gespeichert, indem der String an jedem „;“ geteilt wird. Anschließend wird diese Liste zurückgegeben.

10.1.4.3.4.3. Methode: getLastWort()

```
def getLastWort(self):
        wortListe = self._getWortListe()
        self.IPCLogger.logReadLastWord(self.ProzessID)
        return wortListe[-1]
```

Die Wortliste wird mit self._getWortListe() aufgerufen und der letzte und damit neuste Eintrag zurückgegeben, sowie eine Log-Nachricht zurückgegeben.

10.1.4.3.5. Methode: getIPC_ID

```
def getIPC_ID(self):
speicherListe = self._read()
return speicherListe[5]
```

Position 5 wird aus dem Speicher zurückgegeben.

10.1.4.3.6. Methode: bingo()

```
def bingo(self):
        position0 = "Bingo"
        position1 = self.getDateipfad()
        position2 = self.getGroesse()
```

```
        position3 = self.getWortString()
        if self.checkIfStarted():
            position4 = "started"
        else:
            position4 = ""
        position5 = self.getIPC_ID()
        speicherListe = [position0, position1,
        position2,position3,position4,position5]
        self._write(speicherListe)
        self.IPCLogger.logBingo(self.ProzessID)
```

Die Methode erstellt eine neue Liste, welche an Position 0 „Bingo“ schreibt. Die anderen Positionen werden durch die zuvor beschriebenen Methoden gelesen und an die gleiche Stelle gespeichert. Bei Position 4 handelt es sich um einen booleschen Rückgabewert, welcher je nach Wert zurück in seine Speicherform gebracht werden muss. Diese neue Liste überschreibt die alte Liste im Speicher und eine Log-Nachricht wird geschrieben.

10.1.4.3.7. Methoden: setDateipfad (Dateipfad) & setGroesse(Größe)

```
def setDateipfad(self,dateipfad):
        if self.checkIfBingo():
            position0 = "Bingo"
        else:
            position0 = ""
        position1 = dateipfad
        position2 = self.getGroesse()
        position3 = self.getWortString()
        if self.checkIfStarted():
            position4 = "started"
        else:
            position4 = ""
        position5 = self.getIPC_ID()
        speicherListe = [position0, position1,
        position2,position3,position4,position5]
        self._write(speicherListe)
        self.IPCLogger.logSetDateipfad(self.ProzessID,dateipfad)
```

```
def setGroesse(self,groesse):
        if self.checkIfBingo():
            position0 = "Bingo"
        else:
            position0 = ""
        position1 = self.getDateipfad()
        position2 = groesse
        position3 = self.getWortString()
        if self.checkIfStarted():
            position4 = "started"
        else:
            position4 = ""
```

```
        position5 = self.getIPC_ID()
        speicherListe = [position0, position1,
        position2,position3,position4,position5]
        self._write(speicherListe)
        self.IPCLogger.logSetGröße(self.ProzessID,groesse)
```

Diese Methoden funktionieren ähnlich wie die Bingo Methode. Hier wird jedoch der Dateipfad bzw. die Größe als Argument übergeben und an die entsprechende Position gespeichert. Die anderen Werte werden aus der bisherigen Liste übernommen. Position 0 und 4 sind boolesche Werte, welche umgewandelt werden müssen.

10.1.4.3.8. Methode: addWord(Wort)

```
def addWord(self,wort):
        if self.checkIfBingo():
            position0 = "Bingo"
        else:
            position0 = ""
        position1 = self.getDateipfad()
        position2 = self.getGroesse()
        if(len(self.getWortString()) > 0):
            wort = ";" + wort
        else:
            wort = wort
        position3 = self.getWortString() + wort
        if self.checkIfStarted():
            position4 = "started"
        else:
            position4 = ""
        position5 = self.getIPC_ID()
        speicherListe = [position0, position1,
        position2,position3,position4,position5]
        self._write(speicherListe)
        self.IPCLogger.logAddWord(self.ProzessID,wort)
```

Der Aufbau bleibt gleich zu den 3 vorherigen Methoden. Bei dieser wird das gerufene Wort als Argument übergeben und nur Position 3 verändert. In diesem Fall wird allerdings überprüft, ob der bisher gespeicherte Wortstring leer ist. Ist das der Fall, wird das übergebende Wort unverändert gelassen. Andernfalls wird vor das Wort ein „;“ geschrieben, um später das Splitten in eine Liste zu ermöglichen. Anschließend wird der Wortstring ergänzt und in Position 3 gespeichert. Anschließend wird die Liste wieder in den Shared Memory geschrieben.

10.1.4.3.9. Methode: startGame()

```
def startGame(self):
        if self.checkIfBingo():
            position0 = "Bingo"
        else:
            position0 = ""
        position1 = self.getDateipfad()
```

```
        position2 = self.getGroesse()
        position3 = self.getWortString()
        position4 = "started"
        position5 = self.getIPC_ID()
        speicherListe = [position0, position1,
position2,position3,position4,position5]
        self._write(speicherListe)
        self.IPCLogger.logGameCreation(self.ProzessID)
```

Der Aufbau verändert sich nicht im Vergleich zu den vorherigen Methoden. Hier wird die Position 4 mit „started“ überschrieben und die Liste gespeichert.

10.1.4.3.10. Methode: speicherFreigeben()

```
def speicherFreigeben(self):
        shared_memory = self.Connection
        try:
            shared_memory.unlink()
            self.IPCLogger.logGameDeletion(self.ProzessID)
        except posix_ipc.ExistentialError:
            pass
        finally:
            self.verbindungTrennen()
```

Zum Ende des Bingo-Spiels muss der Shared Memory wieder freigegeben werden, dazu dient diese Methode. Hier wird mit dem .unlink() Aufruf der Speicher freigegeben. Da während des Spiels mehrere Prozesse den Shared Memory benutzen und am Ende freigeben, kann es passieren, dass die Methode aufgerufen wird, allerdings ein anderer Prozess diesen bereits freigegeben hat. Dadurch wird ein posix_ipc.ExistentialError ausgelöst, welcher abgefangen und ignoriert werden kann, da der Speicher bereits freigeben ist. So schreibt auch nur der Prozess, welcher erfolgreich den Speicher freigibt, eine Log-Nachricht. Abschließend muss immer die Verbindung getrennt werden (siehe nachfolgend)

10.1.4.3.11. Methode: verbindungTrennen()

```
def verbindungTrennen(self):
        self.Connection.close_fd()
        self.IPCLogger.logVerbindungTrennen(self.ProzessID)
        return None
```

Diese Methode schließt den Dateideskriptor und damit die Verbindung zum Speicherbereich. Außerdem wird eine Log-Nachricht geschrieben.

10.1.4.4. Herausforderungen und Probleme bei der Umsetzung

Die gespeicherten Daten im Shared Memory mussten immer aktuell sein. Wenn die neuen Daten z.B. in Form von Bytestrings an den bisherigen Inhalt angefügt werden, so ist es schwierig immer die richtigen Informationen zu lesen. Die Lösung war das Speichern in ein Listen-Objekt, welches serialisiert wird und das Alte überschreibt.

Beim Programmieren wird oft kein Wert auf der Konsole ausgegeben, weil vieles in den Speicher geschrieben und gelesen wird, allerdings keinen Output für den User bringt. Das machte das Testen der Methoden umständlich.