Hochschule für Technik
und Wirtschaft Berlin

University of Applied Sciences

Projektbeschreibung
“Super Superhirn”

B42 Software Engineering 2-1.Zug-1.Gruppe
Wintersemester 23/24

Dozent: Prof. Dr. Stephan Salinger

Autoren (Gruppe 15):

Quang Quan ‘Cris’ Tran	584900
Arsene Rayane Wanda Tchatat	585008
Hoai Linh Pham	584555
Daniyal Khan	582859

Inhaltsverzeichnis

1. Einleitung

Das "Mastermind"-Projekt, an dem wir gearbeitet haben, erforderte umfangreiche Analysen, Entwürfe und Implementierungen, um unser Ziel zu erreichen, eine Softwarelösung von höchster technischer Qualität zu entwickeln. Das Spiel "Mastermind", bei dem ein Spieler in die Rolle eines Programmierers schlüpft und versucht, einen geheimen Farbcode zu entschlüsseln, war das Spiel, mit dessen Entwicklung wir beauftragt wurden. Man kann gegen den Computer als Programmierer oder als Rater spielen, und man hat auch die Möglichkeit, als Zuschauer teilzunehmen, wobei der Computer gegen einen Server antritt. Die vielen Optionen, die wir in das Spiel eingebaut haben, erlauben es Ihnen, all diese Dinge zu tun. In den folgenden Abschnitten dieser Projektbeschreibung werden wir uns eingehender mit der Analyse, dem Design und den Implementierungsstrategien befassen. Insbesondere werden wir den ausgewählten Softwareprozess und die Qualitätssicherungsmaßnahmen untersuchen, die eingeführt wurden. Abschließend bewerten wir das fertige Produkt, reflektieren die Projektdurchführung und zeigen auf, in welchen Bereichen das Projekt noch effektiver hätte durchgeführt werden können.Wir haben eine umfassende Analyse durchgeführt, um die Anforderungen zu ermitteln und eine solide Grundlage für das Design zu schaffen. Auf der Grundlage dieser Analyse stellten wir sicher, dass unser Softwaredesign und die Implementierung den Qualitätsstandards und Best Practices entsprachen. Durch die Implementierung eines methodischen Qualitätssicherungsprozesses waren wir in der Lage, alle im Code vorhandenen Fehler zu überprüfen und zu korrigieren. Die Qualitätssicherung bildete die Grundlage unseres Projekts. Dazu gehörten die strikte Einhaltung von Codierungsstandards, automatisierte Tests und regelmäßige Codeüberprüfungen.

2. Analyse

Die Analysephase sollte sich über die gesamte Projektlaufzeit erstrecken. Der Grund dafür war, dass immer die Möglichkeit bestand, neue Anforderungen einzubeziehen. Es stellte sich heraus, dass dies nicht der Fall war. Ein beträchtlicher Teil der Analyse wurde jedoch in den ersten zwei Wochen des Projekts entwickelt. Man könnte sagen, dass dies eine bemerkenswerte Leistung war. Alle Informationen, die in diesem Abschnitt enthalten waren, wurden während des gesamten Projekts auf dem neuesten Stand gehalten und bei Unstimmigkeiten geändert. Es gab zwei wichtige Ereignisse, die während der Phase stattfanden und die die Phase ausmachten. Zum einen das Gespräch mit dem Kunden, bei dem die Anforderungen gesammelt wurden, und zum anderen die Präsentation der Analyseergebnisse eine Woche später. Diese beiden Ereignisse fanden statt. Beide Ereignisse haben stattgefunden.

2.1. Anforderungen

Funktionale Anforderungen:

- Die Standardregeln von Super Superhirn gelten.
- Die Zahlen 1-8 sollen als Farben gelten.
- Zeichen mit farbigem Hintergrund
- Der Nutzer hat im Ratemodus kein Zeitlimit.
- Im Codiermodus muss die Farbe manuell gesetzt werden.
- Im Codiermodus sollen unehrliche Antworten zulässig sein.
- Farbcode soll im Ratemodus zufällig generiert werden.
- Das Feedback als Codierer muss manuell erfolgen.
- Am Ende soll die korrekte Kombination dargestellt werden.
- Das gesamte Spielbrett wird nach jedem Zug aktualisiert und angezeigt
- Spielbar als Codierer oder Rater
- Als Codierer soll der Spieler bei jedem Rat feedback geben.
- Robustheit gegen Falscheingaben.
- Nach einer Runde kann ein neues Spiel gestartet werden.
- Richtigkeit des Spiels muss garantiert sein.
- Gegen fehlerhafte User-Eingaben gewappnet sein.

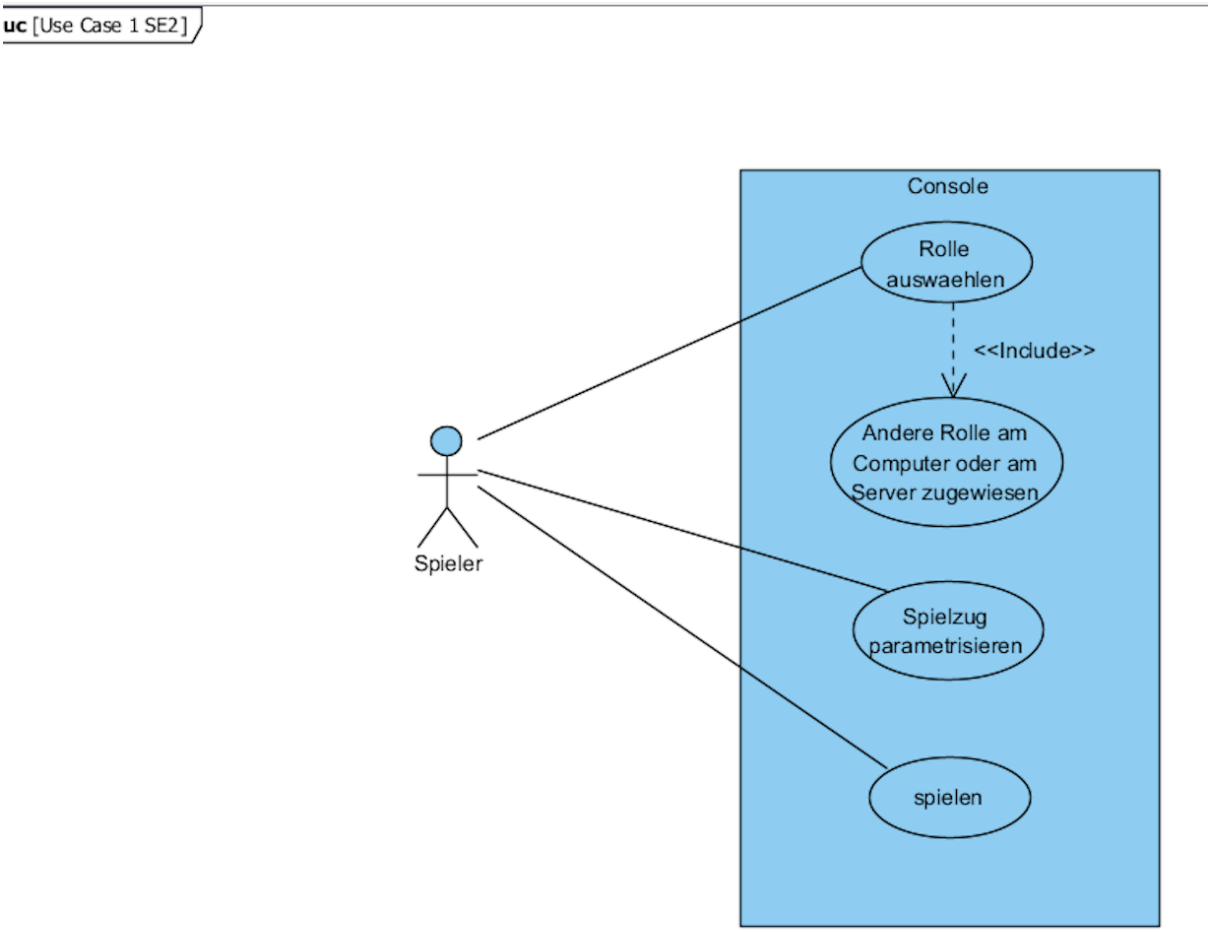
Nicht funktionale Anforderungen:

- Kommerziell nutzbar
- Zielgruppe ab 6 Jahren (Jugendschutz)
- Programm soll auf Deutsch sein
- Auf allen gängigen Betriebssysteme funktionieren
- Python 3
- OOP

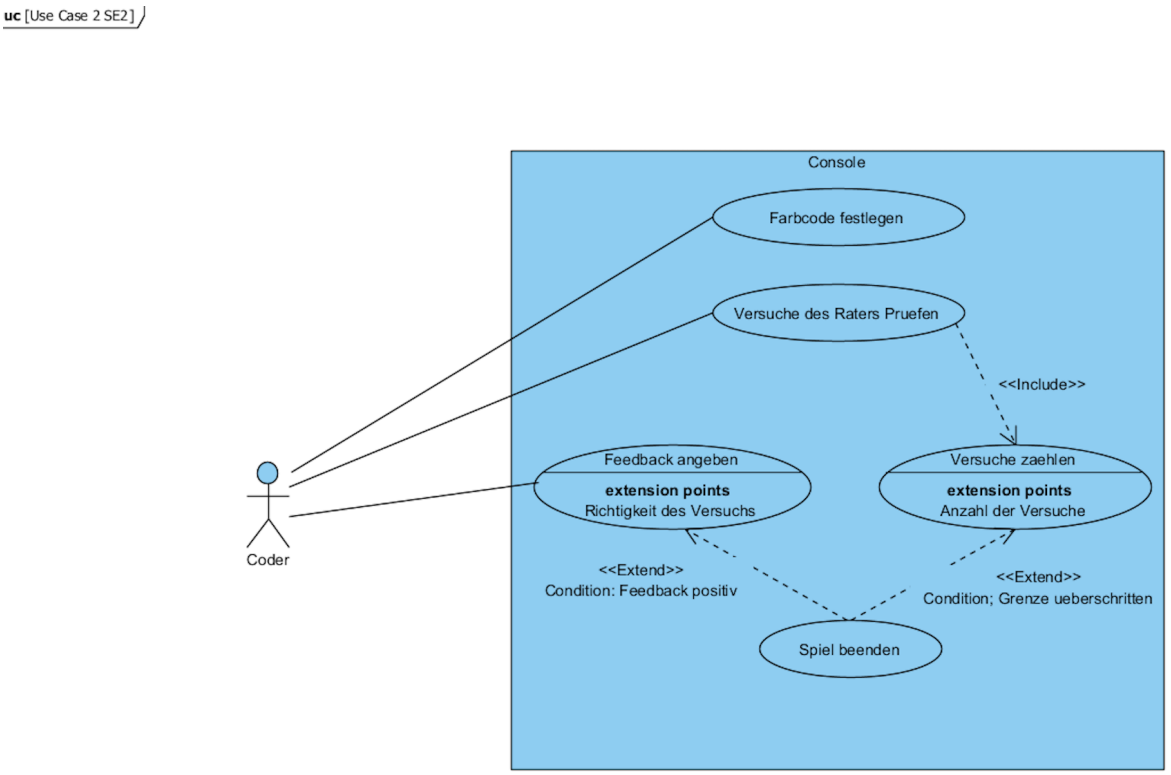
- Ersichtliche Bedienbarkeit / Intuitivität
- Im Ratermodus soll der Bot schnellstmöglich antworten.
- Muss auf der Konsole laufen.
- Möglichkeit für zukünftige Erweiterung/Änderungen.
- Dokumentation

2.2. Use Case Diagramme

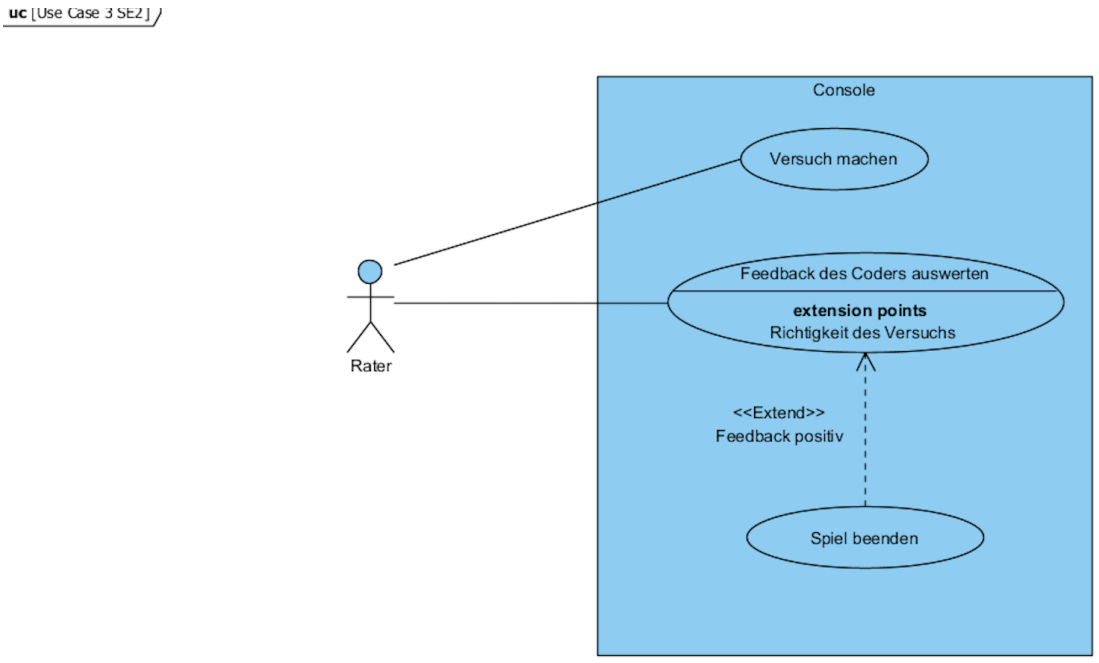
Start-game:



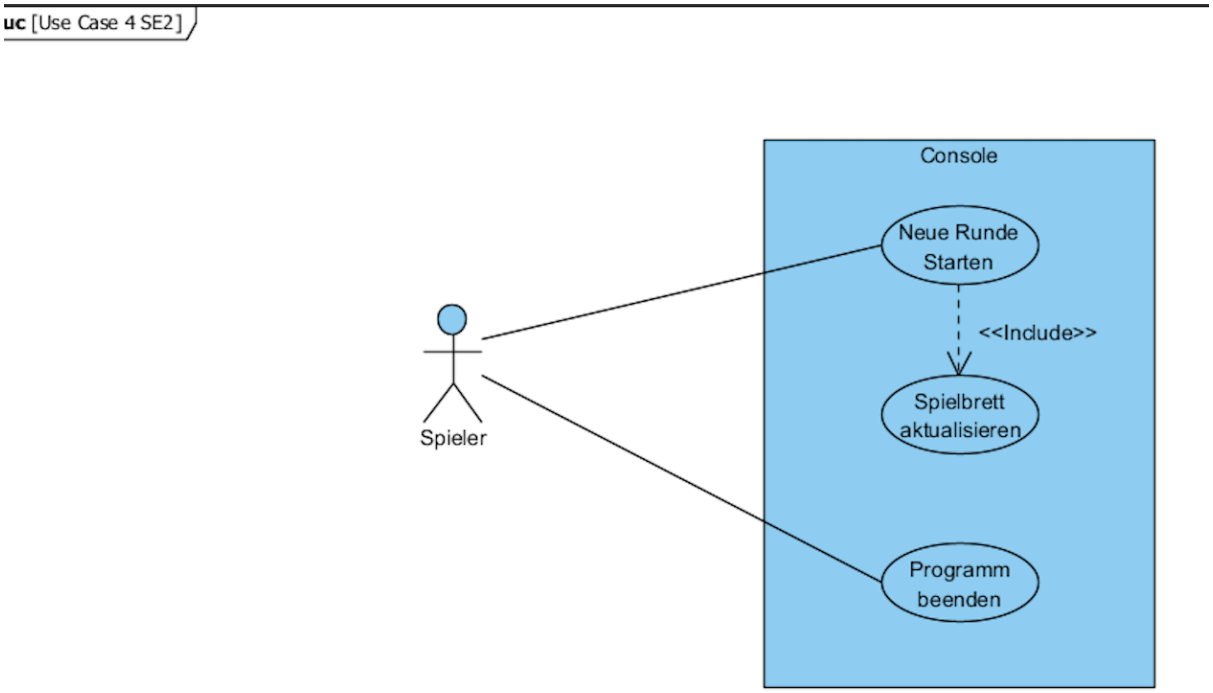
Coder:



Guesser:



End-Game:



2.3. Uses Case Tabellen

Kodierer

- Farbcode festlegen

Name	Farbcode festlegen
Primärakteur	Kodierer
Anwendungsbereich	Partie
Niveau	
Stakeholder und Interessen	Der Kodierer möchte den zu erratenden Code festlegen
Vorbedingungen	Spiel gestartet und Rollenauswahl
Zusicherung im Erfolgsfall	Rateprozess wird gestartet
Auslöser	Rollenerteilung
Haupterfolgsszenario	1- Kodierer legt einen Code fest 2- Festgelegter Farbcode wird zur Gewinnbedingung 3- Rater ist jetzt dran
Erweiterungen	

- Versuche des Raters prüfen

Name	Versuche des Raters prüfen
Primärakteur	Kodierer
Anwendungsbereich	Partie
Niveau	
Stakeholder und Interessen	Der Kodierer möchte den Versuch des Raters untersuchen
Vorbedingungen	Rater hat einen Versuch erteilt
Zusicherung im Erfolgsfall	- Kodierer kann Feedback angeben - Versuchsanzahl wird erhöht
Auslöser	Versuch des Raters vorhanden
Haupterfolgsszenario	1- Kodierer vergleicht seinen festgelegten Code mit dem Versuch des Raters 2- Er kann ein Feedback angeben
Erweiterungen	

- Versuche zählen

Name	Versuche zaehlen
Primärakteur	Spiel
Anwendungsbereich	Partie
Niveau	
Stakeholder und Interessen	Das Spiel zählt die Versuche des Raters ab
Vorbedingungen	Versuch vom Rater erteilt und vom Kodierer geprüft
Zusicherung im Erfolgsfall	Varianten: - Anzahl der Versuche auf dem Bildschirm erhöht - Spiel wird beendet und aufgelöst, weil Versuche aufgebraucht
Auslöser	Prüfung des Versuchs vom Rater
Haupterfolgsszenario	1- Das Spiel erhöht die Anzahl der Versuche um eins 2- Die aktuelle Anzahl der Versuche wird angezeigt oder das Spiel wird beendet
Erweiterungen	

- Feedback geben

Name	Feedback angeben
Primärakteur	Kodierer
Anwendungsbereich	Partie
Niveau	
Stakeholder und Interessen	Der Kodierer gibt einen Feedback zu dem vom Rater erteilten Versuch
Vorbedingungen	Versuch des Raters geprüft
Zusicherung im Erfolgsfall	Varianten: 1- Neuer Rateprozess wird gestartet 2- Spiel wird beendet (Rater hat den Farbcode richtig erraten)
Auslöser	Prüfung des Versuchs
Haupterfolgsszenario	1- Der Kodierer bewertet den Versuch des Raters anhand einer Farbkodierung 2- Rater ist wieder dran und kann einen neuen Versuch machen oder das Spiel wird beendet, weil sein Versuch richtig war
Erweiterungen	

- Spiel beenden

Name	Spiel beenden
Primärakteur	Programm
Anwendungsbereich	Partie
Niveau	
Stakeholder und Interessen	Das Programm beendet den Spielzug, wenn bestimmte Bedingungen eintreffen
Vorbedingungen	Kodierer hat gewonnen oder verloren
Zusicherung im Erfolgsfall	Varianten: 1- Das Spiel kommt am Ende 2- Der Spieler kann eine neue Runde starten
Auslöser	Prüfung des Versuchs oder Angabe des Feedbacks

Haupterfolgsszenario	Der Spielzug ist am Ende, weder der Kodierer noch der Rater können noch Farbcodierungen zusenden.
Erweiterung	

Rater

- Versuch machen

Name	Versuch machen
Primärakteur	Rater
Anwendungsbereich	Partie
Niveau	
Stakeholder und Interessen	Der Rater schickt dem Kodierer seinen Farbcode zur Auswertung
Vorbedingungen	Spiel gestartet und Rolle des Raters ausgewählt
Zusicherung im Erfolgsfall	Varianten: 1- Der Rater ist nicht mehr dran 2- Der Rater bekommt ein Feedback vom Kodierer
Auslöser	Spielzug gestartet
Haupterfolgsszenario	1- Der Rater macht sich Überlegungen über den Geheimcode des Kodierers und fertigt einen Farbcode an 2- Er bestätigt seinen Versuch, der dann zum Kodierer geschickt wird
Erweiterungen	

- Feedback des Kodierers auswerten

Name	Feedback des Kodierers auswerten
Primärakteur	Rater
Anwendungsbereich	Partie
Niveau	
Stakeholder und Interessen	Der Rater untersucht das Feedback, das er bekommen hat, um seinen Farbcode zu verfeinern
Vorbedingungen	Farbcode geschickt und ausgewertet
Zusicherung im Erfolgsfall	Varianten: 1- Farbcode angepasst und neuen Versuch gemacht 2- Spiel wird beendet (Rater hat den Geheimcode richtig erraten)
Auslöser	Auswertung des Versuchs
Haupterfolgsszenario	1- Der Rater bekommt ein Feedback vom Kodierer für seinen Versuch 2- Der Rater passt seinen Farbcode entsprechend an und macht einen neuen Versuch 3- Oder das Spiel wird beendet, weil der Rater eine gute Auswertung bekommen hat
Erweiterungen	

- Spiel beenden

Name	Spiel beenden
Primärakteur	Programm
Anwendungsbereich	Partie
Niveau	
Stakeholder und Interessen	Das Programm beendet den Spielzug, wenn bestimmte Bedingungen eintreffen
Vorbedingungen	Gute Auswertung vom Kodierer bekommen oder Versuche aufgebraucht

Zusicherung im Erfolgsfall	Varianten: 1- Das Spiel kommt am Ende 2- Der Spieler kann eine neue Runde starten
Auslöser	Auswertung des Versuchs
Haupterfolgsszenario	Der Spielzug ist am Ende, weder der Kodierer noch der Rater können noch Farbkodierungen zusenden.
Erweiterungen	

Start

- Rolle auswählen

Name	Rolle auswählen
Primärakteur	Spieler
Anwendungsbereich	Programm
Niveau	
Stakeholder und Interessen	Der Spieler soll die Rolle, die er übernehmen will auswählen
Vorbedingungen	Programm funktioniert
Zusicherung im Erfolgsfall	Die andere Rolle wird am Computer oder am Server zugewiesen
Auslöser	Programm gestartet
Haupterfolgsszenario	Der Spieler startet das Programm und wählt eine Rolle aus
Erweiterungen	

- Andere Rolle am Computer oder am Server zugewiesen

Name	Andere Rolle am Computer oder am Server zugewiesen
Primärakteur	Programm
Anwendungsbereich	Partie
Niveau	
Stakeholder und Interessen	Das Programm gibt die andere Rolle an dem Computer oder dem Server
Vorbedingungen	Der Spieler hat eine Rolle ausgewählt
Zusicherung im Erfolgsfall	Der Spieler kann den Spielzug parametrisieren
Auslöser	Rolle vom Spieler ausgewählt

Haupterfolgsszenario	Das Programm weist die übrige Rolle dem Computer oder dem Server zu
Erweiterungen	

- Spielzug parametrisieren

Name	Spielzug parametrisieren
Primärakteur	Spieler
Anwendungsbereich	Partie
Niveau	
Stakeholder und Interessen	Der Spieler legt die Eigenschaften des Spielzuges fest
Vorbedingungen	Der Spieler hat eine Rolle ausgewählt und die andere Rolle wurde schon zugewiesen
Zusicherung im Erfolgsfall	Varianten: 1- Die Eigenschaften des Spielzuges sind festgelegt 2- Der Spielzug startet
Auslöser	Spieler wählt eine Rolle aus
Haupterfolgsszenario	Der Spieler legt eine Länge für den Farbcode und die Anzahl der zu nutzenden Farben fest
Erweiterungen	

- Spielen

Name	Spielen
Primärakteur	Spieler, Programm
Anwendungsbereich	Partie
Niveau	
Stakeholder und Interessen	Der Spieler und das Programm interagieren zusammen im Rahmen eines Spielzuges
Vorbedingungen	Spielzug parametrisiert
Zusicherung im Erfolgsfall	Der Spieler kann in der ausgewählten Rolle spielen

Auslöser	Spielzug parametrisiert
Haupterfolgsszenario	Der Spieler interagiert mit dem Programm und kann Farbcodes senden und empfangen
Erweiterungen	

End

- Neue Runde starten

Name	Neue Runde starten
Primärakteur	Spieler
Anwendungsbereich	Partie
Niveau	
Stakeholder und Interessen	Der Spieler will eine neue Runde spielen
Vorbedingungen	Spielzug beendet
Zusicherung im Erfolgsfall	Ein neuer Spielzug wird gestartet
Auslöser	Ende eines Spielzuges
Haupterfolgsszenario	Der Spieler tritt in einen neuen Spielzug ein
Erweiterungen	

- Spielbrett aktualisieren

Name	Spielbrett aktualisieren
Primärakteur	Programm
Anwendungsbereich	Partie
Niveau	
Stakeholder und Interessen	Das Programm löscht alle Daten des alten Spielzuges und erstellt ein neues Spielbrett
Vorbedingungen	Neue Runde gestartet
Zusicherung im Erfolgsfall	Das Spielbrett ist erneut aufgebaut
Auslöser	Spieler, der einen neuen Spielzug machen will
Haupterfolgsszenario	Die Daten des alten Spielbretts werden entfernt und ein neues aufgebaut
Erweiterungen	

- Programm beenden

Name	Programm beenden
Primärakteur	Spieler
Anwendungsbereich	Programm
Niveau	
Stakeholder und Interessen	Der Spieler schließt das Programm
Vorbedingungen	Spielzug beendet
Zusicherung im Erfolgsfall	Varianten: 1- Das Programm schließt 2- Kein Spielzug mehr kann gestartet werden

Auslöser	Spieler, der das Programm nicht mehr nutzen will
Haupterfolgsszenario	Der Spieler hat das Programm beendet
Erweiterungen	

3. Design

Nach der Analysephase haben wir einen Überblick über verschiedene Aspekte des zu entwickelnden Systems erhalten können, was wir jetzt in der Designphase als Formen bzw. Diagramme darstellen sollten. Dieser Schritt war zweifellos der wichtigste, weil darauf die nachfolgende Implementierungsphase und die eigentliche Konkretisierung unseres Projektes basieren sollten.

3.1. Architekturstil

Wir haben uns für eine Kombination aus mehreren Architekturstilen entschieden, um sowohl die funktionalen Anforderungen als auch die nicht-funktionalen Anforderungen wie Wartbarkeit, Erweiterbarkeit und Testbarkeit zu erfüllen. Die gewählten Architekturstile spiegeln unsere Bestrebungen wider, eine modulare, erweiterbare und robuste Anwendung zu schaffen.

Haupt-Architekturstile:

- **Layered (Schichten-) Architektur**

Unser System ist in verschiedene Schichten unterteilt, die für spezifische Aufgabenbereiche verantwortlich sind. Beispielsweise gibt es klare Trennungen zwischen der Datenlogik (wie Spielregeln und Zustand), der Benutzerschnittstelle und der Netzwerkkommunikation. Diese Trennung ermöglicht eine unabhängige Entwicklung und Wartung der einzelnen Schichten.

- **Client-Server-Architektur:**

Insbesondere für die Netzwerkfunktionalität haben wir einen Client-Server-Ansatz gewählt. Der CoderNetworkClient agiert als Client, der Anfragen an einen Server sendet und Antworten empfängt. Diese Struktur ermöglicht eine klare Trennung von Netzwerkoperationen und Spiellogik.

- **Modulbasierte Architektur:**

Das gesamte System ist in verschiedene Module unterteilt, z.B. Board, Guesser, CoderBaseClient und seine Ableitungen sowie verschiedene UI-Komponenten. Jedes Modul ist für eine spezifische Funktionalität verantwortlich, was die Wiederverwendbarkeit fördert und die Komplexität reduziert.

- **Event-Driven Architecture (EDA):**

Obwohl unser aktueller Fokus auf einer Konsolenanwendung liegt, wurde die Architektur mit Blick auf eine mögliche zukünftige Erweiterung um eine Event-Driven GUI entworfen. Dieser Stil wäre besonders relevant, wenn die Benutzerschnittstelle interaktiver und reaktiver gestaltet wird.

Die Architektur von "Super Supermind" ist das Ergebnis einer sorgfältigen Planung und des Bestrebens, ein System zu schaffen, das nicht nur die aktuellen Anforderungen erfüllt, sondern auch für zukünftige Erweiterungen und Änderungen gut gerüstet ist. Durch die Kombination mehrerer Architekturstile konnten wir eine robuste, flexible und wartbare Anwendung entwickeln. Diese Architektur wird uns auch in zukünftigen Entwicklungsphasen, insbesondere bei der Integration einer GUI oder der Erweiterung der Netzwerkfähigkeiten, zugutekommen.

3.2. Ablauf

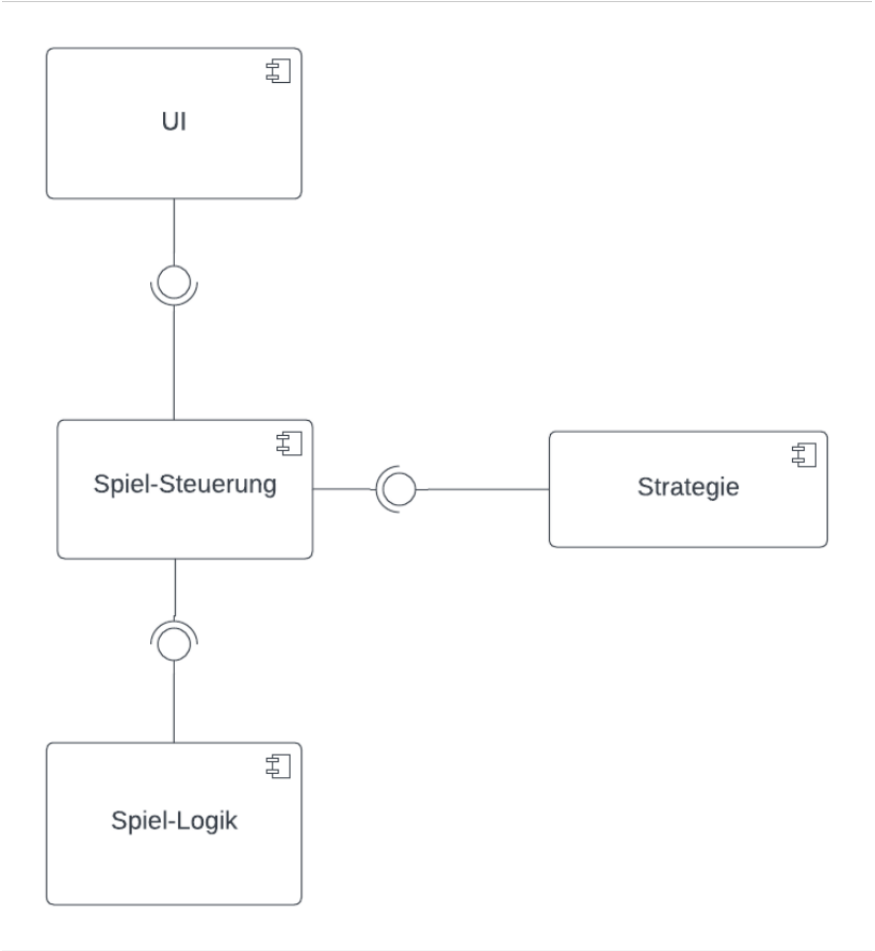
Zum Beginn war es für uns noch ein fremdes Feld, deswegen sollten wir im Laufe unseres Projektes diese Phase mehrmals iterieren. Dank der funktionalen Anforderungen, die wir während der Analysephase haben sammeln können, war es uns klar, dass der zukünftige Nutzer des Programms zwei Rollen übernehmen können sollte (entweder Kodierer oder Rater). Damit konnten wir schon unsere ersten Diagramme erstellen, nämlich zwei Sequenzdiagramme jeweils in der Rolle des Kodierers und des Raters. Dabei hatten wir uns zusammen erstmal Überlegungen über die verschiedenen Aktionen, die der Spieler in den jeweiligen Rollen durchführen sollte, und das System in zwei miteinander kommunizierenden Subsysteme repräsentiert (User Interface und Spiellogik). Danach konnten wir anhand der Prinzipien der Systemzerlegung die wichtigsten Komponenten unseres Systems identifizieren und für jede Komponente eine Reihe von Klassen zuordnen. So entstanden unsere Komponenten-, Klassen- und Zustandsdiagramme.

Nach dieser ersten Iteration sind wir in der Implementierungsphase gelandet, wo wir nach Implementierung einiger Klassen schon erste Mängel unserer Systemarchitektur hervorheben konnten. Diese Fehler konnten darauf zurückzuführen sein, dass keiner von uns vorher schon richtig mit der Programmiersprache Python zu tun hatte, was auch eine richtige Herausforderung für uns in diesem Projekt war, denn es war schwierig, sich ohne diese Grundkenntnisse vorzustellen, was überhaupt möglich oder nicht hinsichtlich unseres Projektes ist. Infolgedessen haben wir unser Design untersucht, um diese Fehler zu beheben, was wiederum dieses Mal einfacher für uns war, da wir nach dieser ersten Implementierung schon einen Überblick darüber hatten, welche Funktionen die verschiedenen Teile unseres Systems anbieten können. Die Grobarchitektur unseres Systems blieb an mehreren Stellen unverändert, die meisten Veränderungen erfolgten in der Feinarchitektur, wobei wir einige Klassen durch andere ersetzten und das Zusammenspiel zwischen einander angepasst haben.
Entgegen aller Erwartungen wurden wir anschließend mit neuen Anforderungen des Kunden konfrontiert, nämlich die Möglichkeit, dass das Spiel im Ratermodus mit einem externen Server, der die Rolle des Kodierers übernimmt, spielbar wird. Dies zwang uns dazu, unser

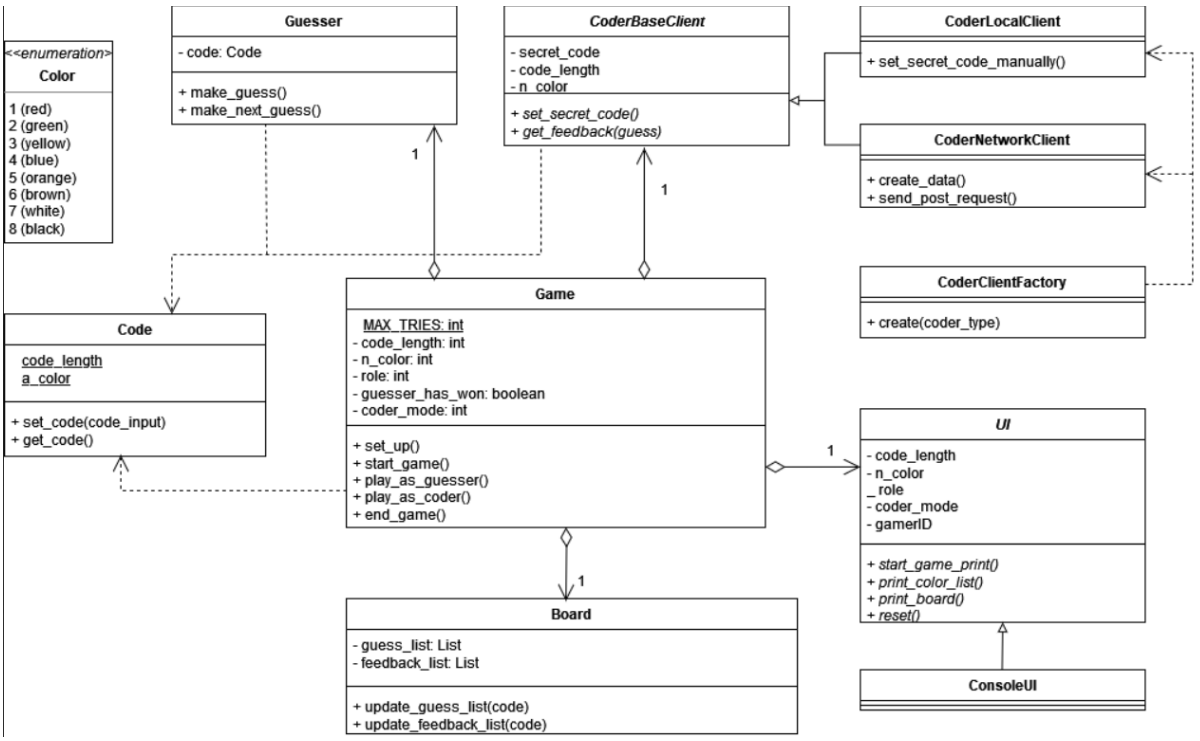
Design nochmal anzupassen bzw. eine neue Komponente zu integrieren und die dazugehörigen Klassen auszudenken.

Endlich haben wir es geschafft, ein Design, das zu den Anforderungen unseres Programms passt, zu beschaffen. Im laufe dieser Phase der Entwicklung haben wir es gelernt, dass die am Anfang geltenden Anforderungen, sich während des Entwicklungsprozesses ändern können, deswegen haben wir uns, um solche Situationen leichter zu beheben und vielleicht später auftretende Probleme lösen zu können, für ein Modul basiertes Design entschieden. Wobei alle Komponenten unseres Programms in geeigneten Modulen gruppiert sind, so dass man auf eine Komponente oder eine bestimmte Funktion unseres Programms schneller und effizienter zugreifen kann.

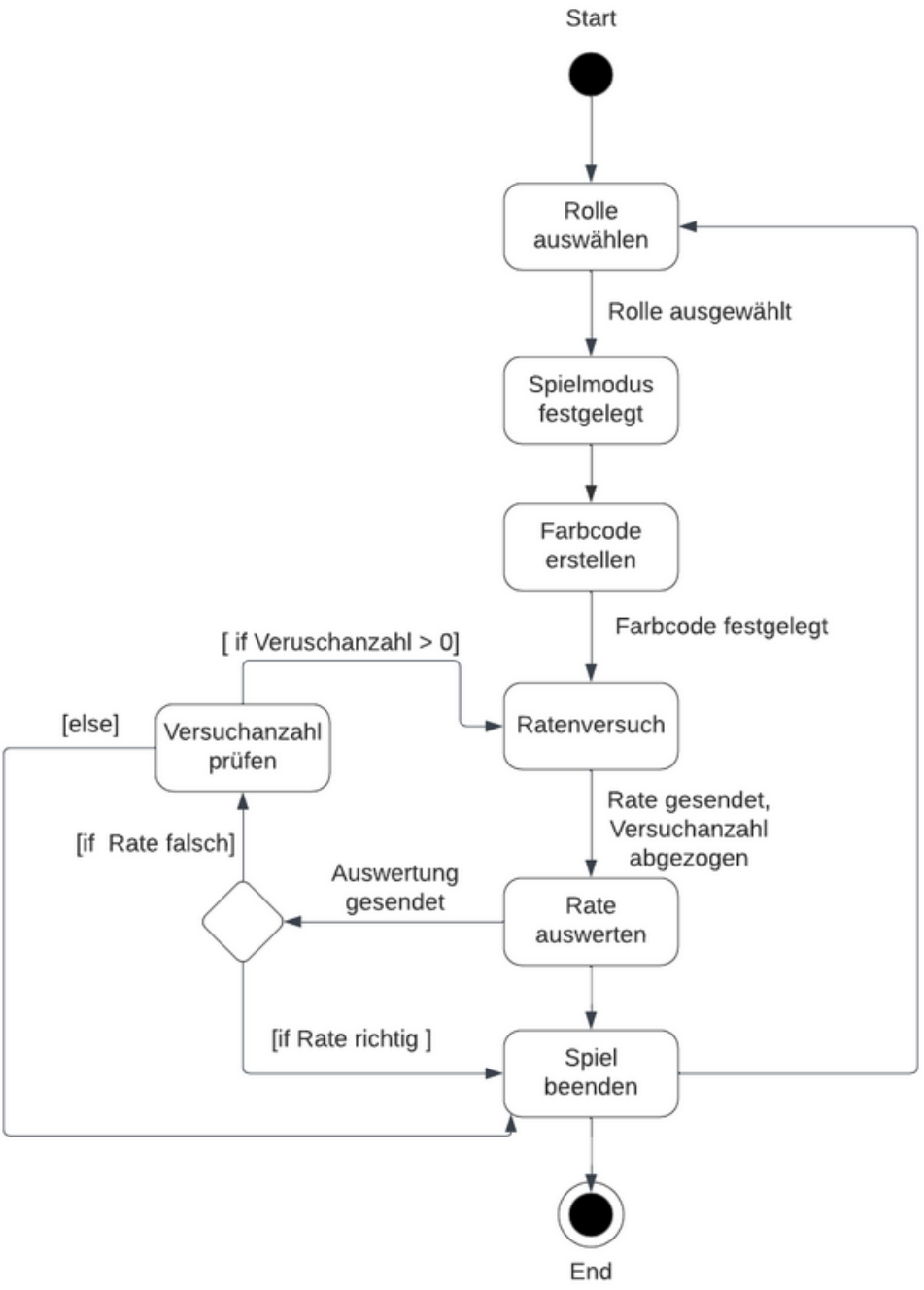
Diagramme:



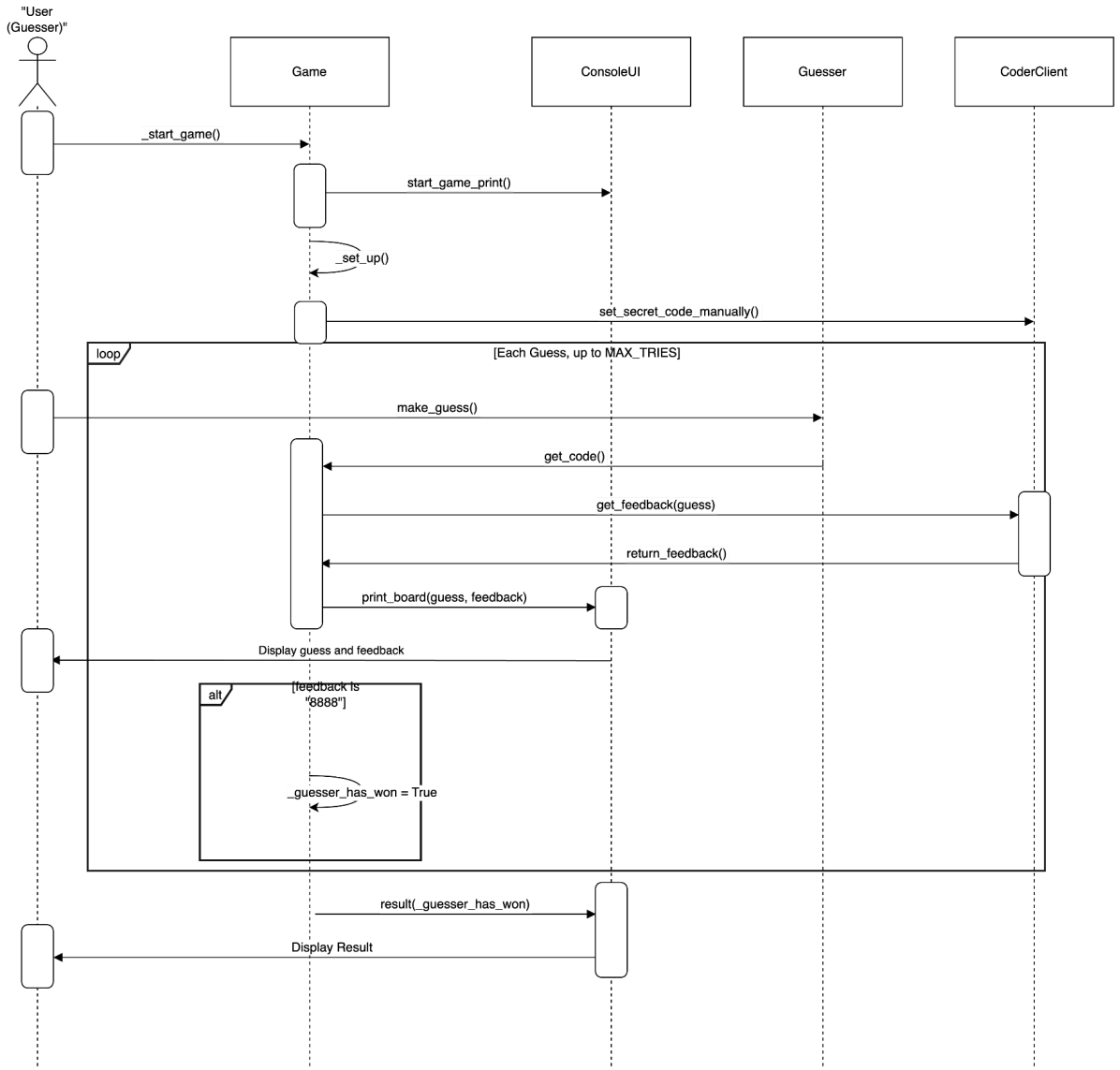
Komponentendiagramm



Klassendiagramm

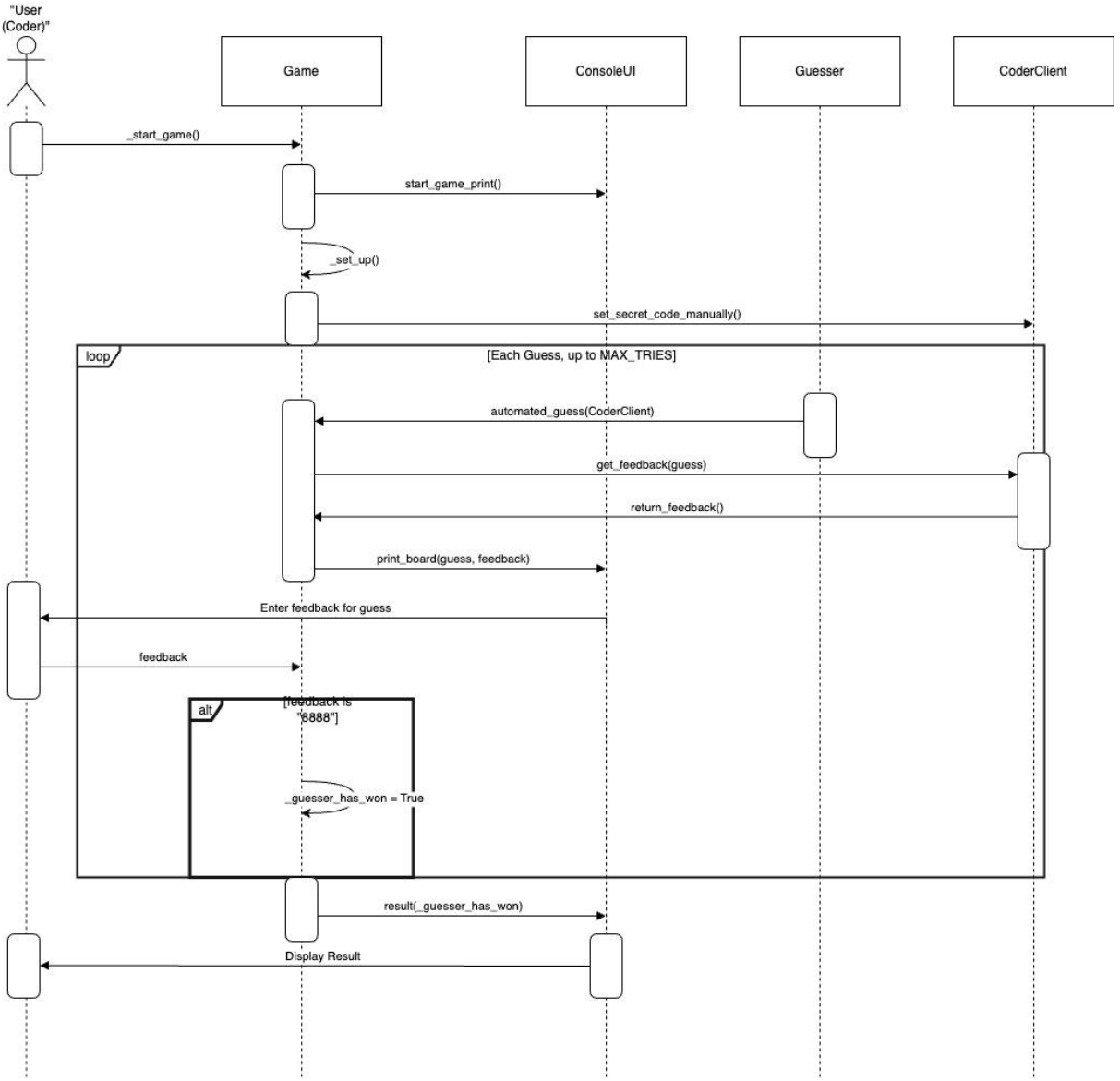


Zustandsdiagramm



User (Guesser) vs Coder (System)

Sequenzdiagramm 1



User (Coder) vs Guesser (System)

Sequenzdiagramm 2

4. Implementierung

Im Rahmen unseres Projekts haben wir eine objektorientierte Herangehensweise, die uns Flexibilität, Wiederverwendbarkeit des Codes und klare Trennung der verschiedenen Softwarekomponenten ermöglicht. Unser Ziel war es, eine robuste und erweiterbare Anwendung zu schaffen, die sowohl auf der Konsole funktioniert und, falls erwünscht, später auch mit einer grafischen Benutzeroberfläche (GUI) ergänzt werden kann.

4.1. Design Patterns

MVC (Model-View-Controller): Zur Trennung der Geschäftslogik von der Benutzerschnittstelle haben wir das MVC-Muster implementiert. Dies ermöglichte es uns, die Spiellogik (Model), die Benutzerinteraktion (Controller) und die Darstellung (View) unabhängig voneinander zu entwickeln und zu testen. Die Board- und Game-Klassen repräsentieren das Model, während UI und ihre Ableitungen (ConsoleUI) als View fungieren. Der Controller wird durch die Interaktion zwischen Game und UI realisiert.

Factory Method: Für die Erstellung von Coder-Objekten haben wir das Factory Method-Muster angewendet. Die CoderClientFactory entscheidet dynamisch, ob ein CoderLocalClient oder ein CoderNetworkClient basierend auf Benutzereingaben instanziiert wird, was die Erstellung von Coder-Objekten vereinfacht und zukünftige Erweiterungen erleichtert.

4.2. Robustheit

Um die Robustheit unseres Spiels zu gewährleisten, haben wir mehrere Maßnahmen ergriffen:

Fehlerbehandlung: Umfassende Fehlerbehandlung wurde implementiert, insbesondere in den Klassen CoderLocalClient und CoderNetworkClient, um mit unvorhersehbaren Eingaben oder Netzwerkfehlern umzugehen.

Eingabevalidierung: Sowohl in der UI-Klasse als auch in den Coder-Klassen wurden Validierungen für Benutzereingaben eingeführt, um sicherzustellen, dass nur korrekte und sinnvolle Daten verarbeitet werden.

Testbarkeit: Durch die klare Trennung von Logik und Darstellung haben wir sichergestellt, dass wichtige Teile des Systems unabhängig voneinander getestet werden können.

4.3. Erweiterbarkeit

Das System wurde mit Blick auf zukünftige Erweiterungen konzipiert:

Modulare Struktur: Durch die Verwendung von Klassen und die klare Trennung der verschiedenen Funktionen ist es einfach, neue Features hinzuzufügen oder bestehende zu ändern, ohne andere Teile des Systems zu beeinträchtigen.

Vorbereitung auf GUI-Integration: Die abstrakte UI-Klasse und die Implementierung des MVC-Musters legen den Grundstein für die spätere Integration einer grafischen Benutzeroberfläche.

Anpassungsfähigkeit: Die Implementierung des Factory- und Strategy-Musters ermöglicht es uns, das System leicht an veränderte Anforderungen anzupassen, wie zum Beispiel die Integration neuer Coder-Typen oder Feedback-Algorithmen.

4.4. Integration neuer Anforderungen

In unserem Projekt haben wir erfolgreich zwei neue Anforderungen integriert: die Möglichkeit, die Länge des Farbcodes auf 4 oder 5 Stellen anzupassen und die Auswahl der Farben zwischen 2 und 8 zu variieren, sowie die Einbindung des Coders über ein Netzwerk. Diese Integrationen demonstrieren die Effektivität unserer vorbereiteten Architektur und unseres Entwicklungsansatzes hinsichtlich Erweiterbarkeit und Anpassungsfähigkeit.

Variable Farblänge und Farbauswahl:

a. Die Anpassung der Farblänge und die Auswahlmöglichkeiten zwischen 2 und 8 Farben wurden durch die flexible Architektur unserer Code-Klasse ermöglicht. Wir haben Funktionen implementiert, die es dem Benutzer ermöglichen, die Länge des Codes und die Anzahl der verwendbaren Farben festzulegen. Diese Funktionen wurden nahtlos in das bestehende System eingebunden, was die Benutzererfahrung erweitert, ohne die Kernlogik zu beeinträchtigen.

Netzwerkbasierte Coder-Integration:

b. Die zweite wesentliche Erweiterung betraf die Netzwerkanbindung des Coders. Durch die Verwendung eines Client-Server-Modells und die Implementierung des CoderNetworkClient konnten wir dem Benutzer die Option bieten, den Coder entweder lokal oder über ein Netzwerk zu nutzen. Dies wurde erreicht, ohne die grundlegende Spiellogik oder die Benutzeroberfläche zu verändern, was die Flexibilität des Spiels unterstreicht.

Die erfolgreiche Integration dieser neuen Anforderungen verdeutlicht die Wichtigkeit und Wirksamkeit unserer anfänglichen Designentscheidungen. Durch die vorbereitete modulare Struktur und die Anwendung von Design Patterns konnten wir flexibel und effizient auf Änderungen reagieren.

Die Implementierung unseres Projekts "Super Supermind" basiert auf soliden objektorientierten Prinzipien und bewährten Entwurfsmustern. Wir haben großen Wert auf Robustheit, Testbarkeit und Erweiterbarkeit gelegt, um eine zuverlässige und zukunftssichere Anwendung zu gewährleisten. Unser Code reflektiert Best Practices in Software-Design und -Architektur, was eine solide Basis für zukünftige Entwicklungen und Verbesserungen bildet

4.5. Anwendung von Knuth's Algorithmus

In unserem Projekt haben wir Knuth's Algorithmus, einen zentralen Bestandteil der Spiellogik, effektiv implementiert. Dieser Algorithmus, bekannt für seine Fähigkeit, das Mastermind-Spiel effizient zu lösen, spielt eine entscheidende Rolle in der Guesser-Klasse unseres Projekts.

- **Initialisierung und Hypothesenbildung**: Zunächst generiert der Guesser eine Liste aller möglichen Codes, die auf der Grundlage der gegebenen Code-Länge und der verfügbaren Farben erstellt wird. Diese umfassende Liste repräsentiert den gesamten Lösungsraum zu Beginn des Spiels.

- **Erster Rateversuch**: Der Algorithmus startet mit einem anfänglichen, oft zufälligen Rateversuch. Dieser initiale Schritt ist der Ausgangspunkt für die iterative Analyse und Einengung der möglichen Lösungen.

- **Feedback-Verarbeitung**: Nach jedem Rateversuch bewertet der Algorithmus das Feedback, das angibt, wie viele Stifte in der richtigen Position und Farbe sind. Diese Informationen sind entscheidend, um den Lösungsraum schrittweise einzugrenzen.

- **Eliminierungsprozess**: Unter Verwendung des erhaltenen Feedbacks entfernt der Algorithmus alle Codes aus der Liste der möglichen Lösungen, die nicht dasselbe Feedback erzeugen würden, wenn sie der geheime Code wären.

- **Auswahl des nächsten Rateversuchs**: Basierend auf der aktualisierten Liste möglicher Codes wählt der Algorithmus den nächsten Rateversuch aus. Diese Entscheidung wird unter Anwendung des Minimax-Prinzips getroffen, um den effizientesten nächsten Schritt zu bestimmen.

Die Implementierung von Knuth's Algorithmus ermöglicht es dem Guesser, den geheimen Code mit einer minimierten Anzahl von Versuchen zu erraten, und trägt somit wesentlich zur Intelligenz und Effizienz des Spiels bei. Diese Implementierung unterstreicht unsere Fähigkeit, fortgeschrittene Algorithmen effektiv in realen Anwendungen einzusetzen und bietet eine solide Grundlage für weitere Optimierungen und Erweiterungen im Spiel.

5. Qualitätssicherung

5.1. Konstruktive Qualitätssicherung

Wir haben uns nach Anforderungen des Projekts die Frage für den Kunden gestellt. Jeder musste seine eigenen Fragen überlegen, damit alle Unklarheiten geklärt werden, um mit der Analysephase zu beginnen. Außerdem wissen wir nach dem Interview mehr über die Kundenwünsche, um die späteren Änderungen zu vermeiden.

Wir haben das Agile-Scrum Modell als Prozessmodell benutzt. Es wurde entwickelt, um Teams dabei zu helfen, sich auf die schnelle Anpassung an sich ändernde Anforderungen während des Entwicklungsprozesses zu konzentrieren. Dazu haben wir die Teamarbeit-Managing App “Trello” verwendet. Es gibt Scrum, Backlogs, Sprint-Reviews, usw. Dort kann jedes Mitglied seine eigenen Aufgaben sehen und wir können auch kontrollieren, welche Arbeit von wem erledigt wird, welche Arbeit noch im Prozess ist und welche Arbeit abgeschlossen wurde. Außerdem verwenden wir auch Pair Programming, damit wird sichergestellt, dass jedes Teammitglied alle Teile des Projektes kennengelernt und die Fehler früher erkannt werden.

Am Anfang war es etwas schwierig, einen Termin für ein Treffen zu vereinbaren, da jeder seinen eigenen Arbeits- und Lernplan hatte. Wir setzten uns jedoch zusammen und ordneten jeweils unsere Zeitpläne, damit wir zusammenarbeiten konnten. Die Zusammenarbeit befindet sich dienstags, mittwochs und sonntags. Normalerweise haben wir uns an der Hochschule und sonntags haben wir um Discord zusammengearbeitet. Danach haben wir auch die Arbeit geteilt, die man alleine machen muss, weil die Zusammenarbeit Zeit nicht reicht. Jeder hat seinen Teil zu Hause gemacht und die anderen in den nächsten Treffen erklärt, damit alle die Implementierungen und Logik verstehen können.

Nach den Präsentationen der Arbeitsergebnisse werden wir ein Sprint-Review durchführen, um zu überprüfen, was wir erledigt haben und welche Probleme wir noch haben.

Als Qualitätsmodell haben wir die ISO 9126 gewählt. Das bedeutet, wir kümmern uns um die Funktionalität, Zuverlässigkeit, Benutzbarkeit, Effizienz und Übertragbarkeit. Die Funktionalität soll die Angemessenheit, Genauigkeit und auch die Interoperabilität abdecken. Andererseits besteht die Zuverlässigkeit sowohl aus der Reife als auch der Fehlertoleranz und Wiederherstellbarkeit. Um die Nutzung zu erleichtern , soll das Programm verständlich, erlernbar und bedienbar sein. Wir müssen auch auf das Zeitverhalten und die Ressourceneffizienz aufpassen. Darüber hinaus soll das Spiel änderbar sein. Das heißt, dass man die Möglichkeit ohne unerwartete Auswirkungen haben soll, die Fehler zu identifizieren und die Änderungen in der Software oder neue Anforderungen von den Kunden vorzunehmen. Mit dem Ziel der Übertragbarkeit soll es auch die Anforderungen an die Anpassbarkeit, Installierbarkeit und Konformität erfüllen.

Bei der Robustheit haben wir es implementiert, damit das Spiel bei Fehlerhaften Eingaben nicht abstürzt, sondern man eine Hinweis bekommt und die Eingabe nochmal machen kann. Wenn der Spieler als Kodierer spielen möchte, wird der Bot nach jeder Feedback sein Rate verbessern. Aus dieser Funktion hat der Bot auch die Möglichkeit zu gewinnen. Unser Spiel

ist gut mit anderen Systemen oder Plattformen interagiert und widerstandsfähig gegenüber Änderungen in der Umgebung ist.

Um die Wartbarkeit zu erreichen, haben wir den Code in gut definierte Module und Komponenten aufgeteilt, um Änderungen oder Upgrades an bestimmten Teilen des Systems zu ermöglichen, ohne das gesamte System zu beeinträchtigen. Neben den Kommentaren und der Projektbeschreibung haben wir auch die klaren und umfassenden Dokumentationen, damit die Entwicklern in der Zukunft die effiziente Fehlerbehebung und Wartung ermöglichen.

Gemäß den Richtlinien von PEP8 halten wir uns an die Lesbarkeit, Sichtbarkeit und Namenskonventionen des Codes. Alle Methoden und Klassen haben wir auf Englisch implementiert. Und unsere Dokumentation ist auf Deutsch.

5.2. Analytische Qualitätssicherung

Tests haben wir Unit-Tests , um sicherzustellen, dass einzelne Komponenten in Ordnung funktionieren. Außerdem haben wir auch Integrationstests und Funktionstests.

Beim Python ist es schwierig, das ganze Spiel mit UI und auch Spiellogik zu testen, weil wir noch die Methode haben, die Random benutzen.

Für die Tests benutzen wir Mock als Eingabe. Es wird geprüft, ob alle Eingaben gespeichert werden oder wenn man die falsche Eingabe eingibt, wird das Spiel nicht abbrechen, sondern fordern wir den Spieler nochmal einzugeben auf. Wir testen das Spiel auch mit verschiedenen Szenarien, um sicherzustellen, dass es unter unterschiedlichen Bedingungen robust funktioniert.

Wir testen nicht nur die Spiellogik, sondern auch die Benutzeroberfläche des Spiels. Außerdem haben wir auch die Tests geschrieben, um zu prüfen, ob ein neues Spiel erstellt wird, welche alle Eingabe, Code, Versuchsanzahl erneut werden, wenn man noch mal spielen möchte.

6. Fazit

Beurteilung des Ergebnisses

Das Ergebnis unseres Projekts "Super Supermind" erfüllt die grundlegenden Anforderungen und Ziele, die wir uns zu Beginn gesetzt hatten. Wir haben ein funktionsfähiges und robustes Spiel entwickelt, das sowohl in einer Konsolenumgebung als auch mit potenzieller Erweiterung für eine grafische Benutzeroberfläche (GUI) funktioniert. Durch die Anwendung von Design Patterns wie MVC, Factory Method und Strategy Pattern haben wir eine solide Architektur und eine klare Trennung der Verantwortlichkeiten erreicht, was die Wartbarkeit und Erweiterbarkeit des Projekts erleichtert.

Die Implementierung zeichnet sich durch hohe Robustheit aus, insbesondere durch umfassende Fehlerbehandlung und Eingabevalidierung, was die Zuverlässigkeit des Spiels gewährleistet. Die Vorbereitung auf die GUI-Integration stellt sicher, dass das Projekt für zukünftige Anforderungen und Erweiterungen gut aufgestellt ist.

Beurteilung des Prozesses

Der Entwicklungsprozess verlief größtenteils effizient und effektiv. Die Anwendung agiler Methoden ermöglichte es uns, flexibel auf Änderungen zu reagieren und kontinuierlich Verbesserungen vorzunehmen. Die klare Rollenverteilung innerhalb des Teams und regelmäßige Meetings trugen zu einer guten Kommunikation und einem reibungslosen Ablauf bei.

Einer der größten Erfolge war unsere Fähigkeit, das Projekt in modularer Weise zu strukturieren, was die parallele Entwicklung verschiedener Komponenten ermöglichte. Dies führte zu einer effizienten Nutzung der Ressourcen und Zeit.

Verbesserungspotenzial

Obwohl das Projekt erfolgreich war, gibt es Bereiche, in denen wir uns verbessern könnten:

1. **Frühzeitige Integration von Tests:** Obwohl unser Code testbar ist, hätten wir von Anfang an stärker auf Test-Driven Development (TDD) setzen sollen, um noch robustere Lösungen zu gewährleisten.
2. **Projektmanagement:** Obwohl unser agiles Vorgehen gut funktionierte, könnten wir in zukünftigen Projekten die Planung und das Zeitmanagement weiter optimieren, um Engpässe zu vermeiden.

Insgesamt sind wir mit dem Ergebnis und dem Prozess zufrieden, erkennen aber auch, dass es immer Raum für Verbesserungen gibt. Die Erfahrungen und Erkenntnisse aus diesem Projekt werden uns in zukünftigen Vorhaben zugutekommen und uns dabei helfen, noch effizienter und effektiver zu arbeiten.