Hochschule für Technik
und Wirtschaft Berlin

University of Applied Sciences

Projektbeschreibung

Software Engineering 2

an der

Hochschule für Technik und Wirtschaft Berlin

Fachbereich 4: Informatik, Kommunikation und Wirtschaft

Studiengang Angewandte Informatik

vorgelegt von

Gruppe 23
Marco Michaelis: 569805
Matthias Weyrich: 579274
Richard Middendorf :582204
Ghamdan Al-Sufyani: 58125

Berlin, 07.01.2025

1. Einleitung

Das Ziel des Projektes war die Entwicklung des Spiels „Super Superhirn“, einer erweiterten Version des klassischen Spiels „Superhirn“. Dabei sollte das Spiel sowohl die Rolle des Kodierers, der eine geheime Farbkombination festlegt, als auch die des Raters, der diese Kombination erraten muss, unterstützen. Neben der eigentlichen Implementierung lag ein besonderer Fokus auf der Qualitätssicherung sowie der konsequenten Anwendung von Prinzipien und Methoden der objektorientierten Analyse, des objektorientierten Designs und der objektorientierten Programmierung. Im Rahmen des Projektes mussten wir uns an eine Vielzahl von Rahmenbedingungen halten. Die wichtigsten dieser Bedingungen sind wie folgt dargestellt:

1. Es ist ein iteratives Vorgehensmodell einzusetzen
2. Aller Code muss in Form von Pairprogramming erstellt werden
3. Das Spiel muss objektorientiert in Python 3 implementiert werden
4. Das Spiel muss auf der Konsole/im Terminal zu spielen sein.

Des Weiteren wurden uns funktionale und nicht-funktionale Anforderungen vorgegeben, die im Laufe des Projektes geändert wurden. Eine Besonderheit des Projektes war, dass wir jede einzelne Prozessphase unserem Professor präsentieren mussten, der die Rolle des Kunden übernahm. Dies ermöglichte regelmäßiges Feedback und gab uns die Möglichkeit, Anpassungen vorzunehmen, um seine Anforderungen optimal zu erfüllen. Im weiteren Verlauf dieser Projektbeschreibung erfolgt eine detaillierte Erläuterung des gewählten Vorgehensmodells sowie der Analyse-, Design- und Implementierungsprozesse und der gewählten Qualitätssicherungsstrategie.

1.1 Ausgewähltes Vorgehensmodell

Die Realisierung des vorliegenden Projektes erfolgt mittels einer modifizierten Form von Scrum als Vorgehensmodell. Diese Entscheidung wurde maßgeblich durch die Tatsache beeinflusst, dass das betreffende Modell bereits allen Mitgliedern des Projektes vertraut war und genügend Flexibilität aufweist, um potenzielle Änderungen der Anforderungen zu bewältigen. Aufgrund der geringen Komplexität des Projektes wurde auf die Spezifikation eines Product Owners verzichtet. Stattdessen wurde die Funktion des Product Owners von sämtlichen Personen übernommen. Darüber hinaus wurden die Dailys aufgrund möglicher Schwierigkeiten bei der Terminfindung durch zwei stattfindende Weeklys ersetzt. Die gesamte Kommunikation sowie die Scrum Meetings fanden dabei über Discord statt. In Bezug auf die Projektprozesse wurde für jeden von ihnen ein individueller Sprint konzipiert. Im Rahmen dessen erfolgte eine Einschätzung und Verteilung der Aufgaben. Die Abschlüsse der Sprints umfassten eine Retrospektive, in welcher die Effektivität der Prozesse sowie potenzielle Optimierungsbereiche diskutiert wurden.

2. Analyse

In der zweiwöchigen Analysephase des Projektes wurde dem Prinzip der objektorientierten Analyse gefolgt. Innerhalb dieser Phase wurde ein 15-minütiges Interview mit dem Kunden durchgeführt, in dem die Möglichkeit bestand, Fragen zu stellen und die vorgegebenen Anforderungen näher zu präzisieren. In der darauffolgenden Woche wurden die Ergebnisse der Analysephase dem Kunden präsentiert und erläutert. Initial ergab unsere Anforderungserhebung die folgenden funktionalen und nicht-funktionalen Anforderungen.

Funktionalen Anforderungen:

1. Als Spieler möchte ich als Codierer spielen, um einen geheimen Farbcode zu erstellen.
2. Als Spieler möchte ich als Rater spielen, um den geheimen Farbcode durch Rateversuche zu entschlüsseln.
3. Als Rater möchte ich nach jedem Versuch eine Rückmeldung erhalten, wie viele Farben und Positionen korrekt sind.
4. Als Rater möchte ich erkennen, wie viele Farben richtig, aber an der falschen Stelle sind.
5. Als Codierer möchte ich dem Computer Feedback geben.
6. Als Rater möchte ich die Möglichkeit haben, bis zu 12 Versuche zu unternehmen, um den Code zu knacken.
7. Als Spieler möchte ich, dass das Spielfeld sich nach jedem Zug aktualisiert, um den aktuellen Fortschritt zu sehen.
8. Als Spieler möchte ich die erweiterte Version „Super Superhirn“ spielen können, um mit 5 Steckplätzen und 8 Farben zu agieren.
9. Als Spieler möchte ich gegen das Programm antreten, das sowohl als Codierer als auch als Rater agieren kann.
10. Als Spieler möchte ich, dass Farben durch Zahlen (1-8) repräsentiert werden, um Eingaben einfach und schnell zu machen.
11. Als Spieler möchte ich das Spiel über die Konsole spielen.
12. Farbkodierung: 1 = Rot, 2 = Grün, 3 = Gelb, 4 = Blau, 5 = Orange, 6 = Braun, 7 = Weiß, 8 = Schwarz.

Nicht-funktionale Anforderungen:

1. Zuverlässigkeit, insbesondere Fehlertoleranz und Robustheit
2. Änderbarkeit und Wartbarkeit, insbesondere Modifizierbarkeit, Erweiterbarkeit & Stabilität
3. Benutzbarkeit & Attraktivität

2.1 Interview

Zur Vorbereitung auf das Interview sollte jeder Gruppenteilnehmer eigenständig 2 bis 3 Fragen formulieren, die an den Kunden gerichtet werden konnten. Diese Fragen haben wir vor dem Interview gemeinsam in der Gruppe gesammelt und abgestimmt.
Dabei sind die folgenden Fragen entstanden:

1. Gibt es spezielle Anforderungen an die künstliche Intelligenz oder Logik des Raters? Soll der Rater etwa verschiedene Schwierigkeitsstufen oder intelligente Ratemethoden wie Algorithmen für Mustererkennung haben?
2. Gibt es Anforderungen an die Logik des Kodierers?
3. Soll es Einstellungsmöglichkeiten geben?
4. Es gibt Varianten mit Lücken als zusätzliche Farbe. Sollen wir das auch ermöglichen?
5. Sollen die Standard Spielregeln für Super Superhirn umgesetzt werden? Wieviele Runden? Normale Bewertungskriterien?
6. Haben Sie spezielle Anforderungen an Benutzbarkeit/Attraktivität? Hätten Sie gerne eine GUI?
7. Soll die Antwort des Kodierers manuell eingegeben werden oder automatisch erstellt werden? Wenn manuell, dürfen falsche Angaben gemacht werden?
8. Sollen am Ende eines Spiels die Rollen getauscht werden, oder soll das Spiel einfach beenden?
9. Soll das Spiel abbrechbar sein, oder wird es nur beendet, wenn man gewinnt?
10. Soll es einen Reset Button geben?
11. Gibt es Anforderungen an die Usereingabe im CLI?
12. Sollen fehlerhafte Eingaben erlaubt sein? Wenn ja, was soll passieren? Abgefangen, mit feedback
13. Welche Qualitätssicherungsmaßnahmen sind spezifisch gefordert?
14. In welchem Umfang und welcher Form sollen UML-Diagramme erstellt werden?
15. Gibt es spezifische Anforderungen an den Detaillgrad der Projektdokumentation?

Aufgrund der limitierten Zeit von lediglich 15 Minuten war es nicht möglich, sämtliche Fragestellungen zu behandeln, obschon wertvolle Informationen generiert werden konnten, die eine substanzielle Erleichterung hinsichtlich des Verständnisses und gegebenenfalls der Präzisierung der Anforderungen mit sich brachten. Die nachfolgenden Antworten wurden im Rahmen der Beantwortung der Fragen formuliert.

1. Als Spieler möchte ich, dass der Computer-Rater optimal rät und schnell ist.
2. Als Spieler möchte ich, dass der Computer-Kodierer einen zufälligen Code generiert.
3. Als Spieler möchte ich in der Lage sein, jederzeit zur Laufzeit die Sprache zu ändern.
4. Als Spieler möchte ich nach dem Beenden eines Spiels ein neues Spiel starten können.
5. Als Spieler möchte ich jederzeit das Spiel abbrechen können.
6. Als Codierer möchte ich falsches Feedback geben (lügen) können. Wenn der Computer dies erkennt, soll dies mit einer Meldung („DU HAST GELOGEN!!!“) angezeigt werden.
7. Als Spieler möchte ich, dass fehlerhafte Eingaben abgefangen werden.

2.2 Überführung der Anforderungen

Um uns einen besseren Überblick über die gesammelten funktionalen Anforderungen zu geben, haben wir als erstes User-Stories erstellt. Die User-Stories lauten wie folgt:

1. Als ein Super Superhirn Spieler möchte ich in der Lage sein, als Rater zu spielen, um die richtige Farbkombination zu erraten.
   Akzeptanzkriterien:
   Das Programm stellt eine Farbkombination, ermöglicht es dem Spieler, Rateversuche abzugeben und bewertet diese.

2. Als ein Super Superhirn Spieler möchte ich in der Lage sein, als Kodierer Farbkombinationen zu setzen.
   Akzeptanzkriterien:
   Der Spieler kann eine Farbkombination setzen, die der Rater erraten soll.

3. Als ein Super Superhirn Spieler möchte ich in der Lage sein, als Kodierer Rateversuche zu bewerten.
   Akzeptanzkriterien:
   Der Spieler sieht die vom Computer erstellten Rateversuche und kann diese mit schwarzen und weißen Pins bewerten.

4. Als ein Super Superhirn Spieler möchte ich die Möglichkeit haben, das Spiel jederzeit zu beenden, um in das Menü zu gelangen.
   Akzeptanzkriterien:
   Ein laufendes Spiel kann jederzeit beendet werden und der Spieler wird wieder in das Menü geführt.

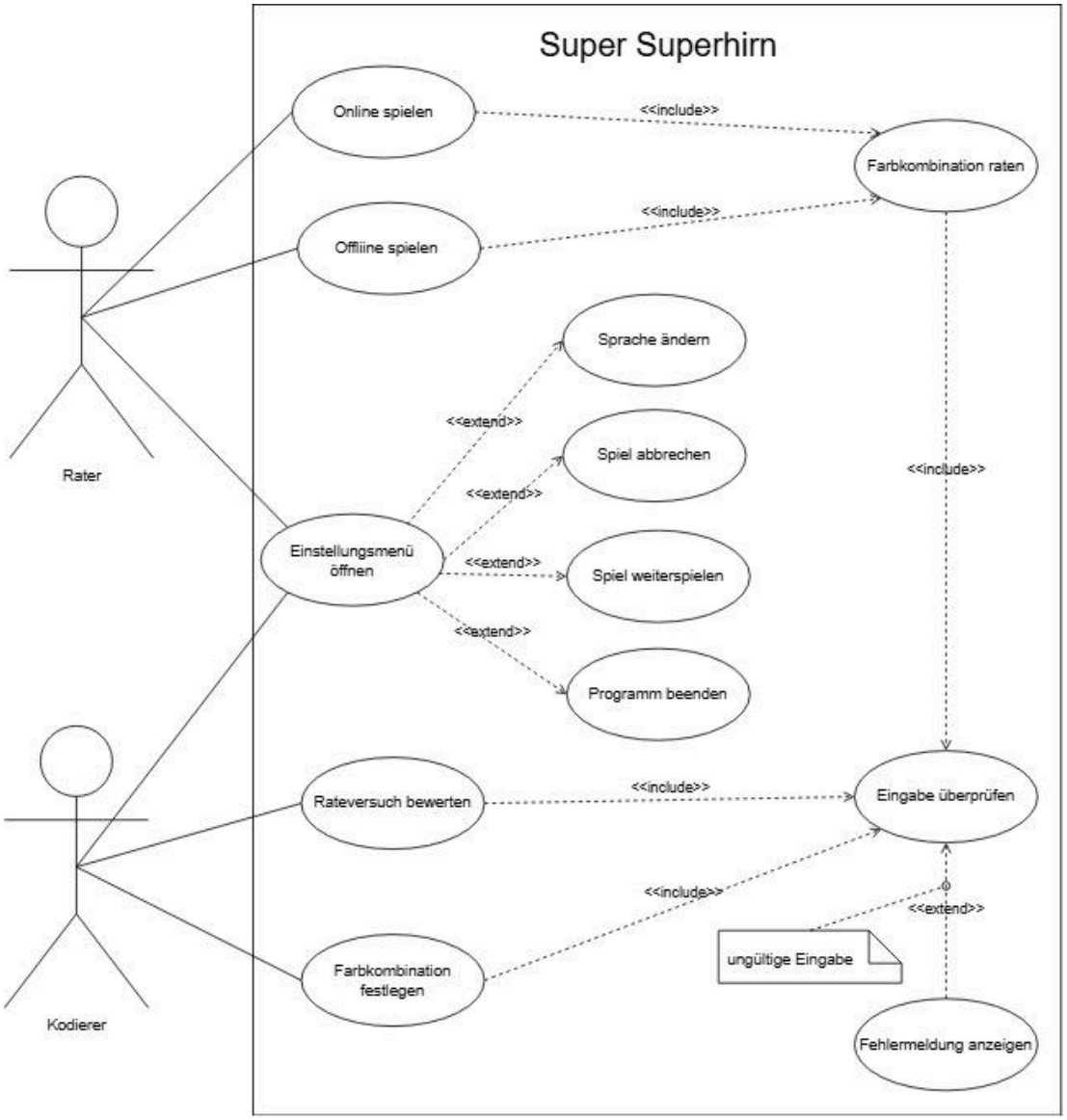
5. Als ein Super Superhirn Spieler möchte ich das Programm verlassen können, um es zu schließen, wenn ich fertig mit dem Spielen bin.
   Akzeptanzkriterien:
   Dem Spieler wird eine Möglichkeit gegeben, das Programm sicher zu beenden.

6. Als ein Super Superhirn Spieler möchte ich die Sprache ändern können, um das Spiel in meiner bevorzugten Sprache zu spielen.
   Akzeptanzkriterien:
   Mehrere Sprachen stehen dem Spieler zur Auswahl, in denen das Spiel spielbar ist.

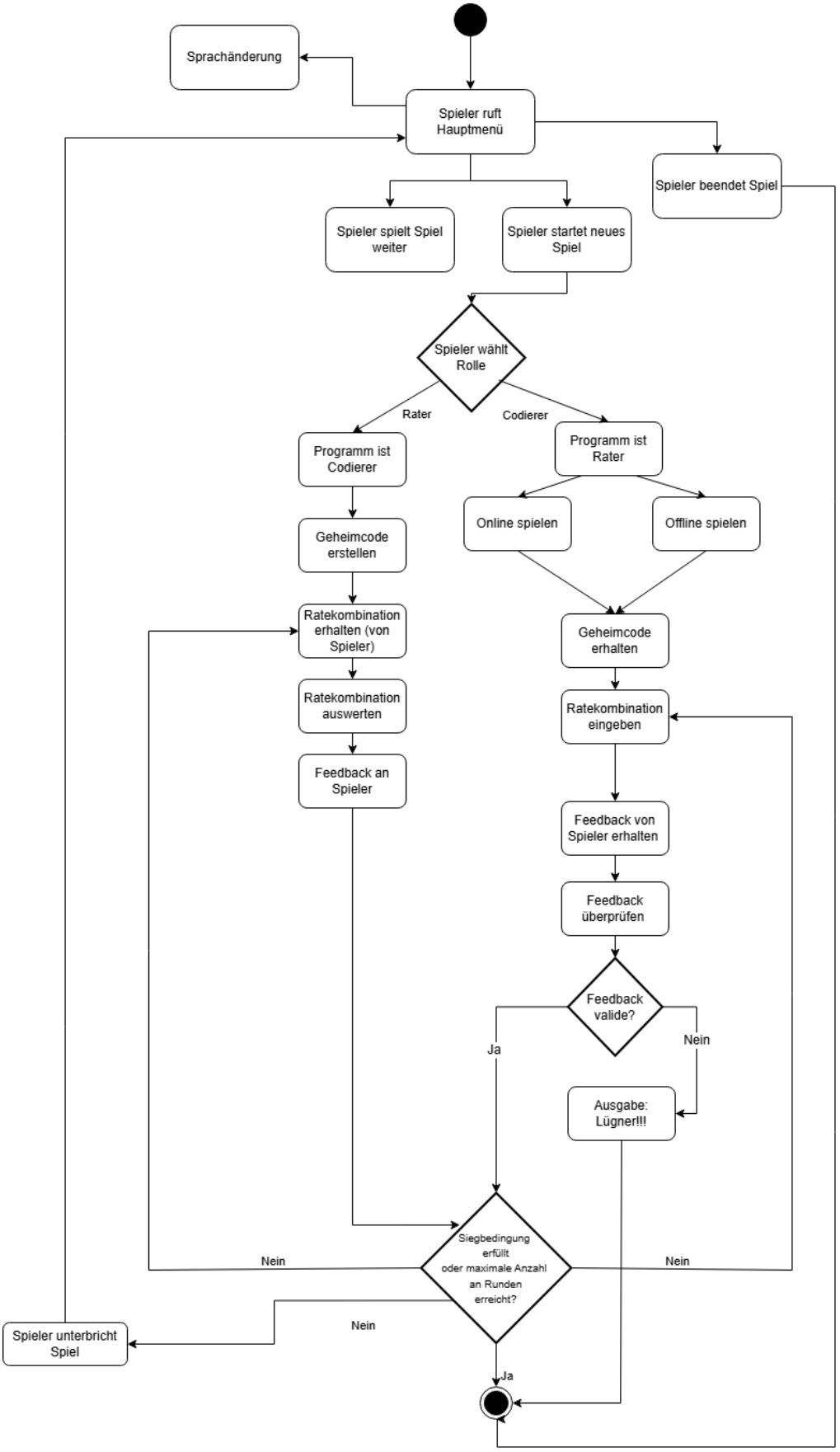
7. Als ein Super Superhirn Spieler möchte ich, dass das Programm mich darauf hinweist, wenn ich eine ungültige Eingabe getätigt habe, so dass ich diese korrigieren kann.
   Akzeptanzkriterien:
   Ungültige Eingaben werden abgefangen, der Spieler wird auf diese hingewiesen und er hat die Möglichkeit, eine erneute Eingabe zu tätigen.

Im darauffolgenden Schritt wurde ein Use Case Diagramm aus den zuvor formulierten User Stories kreiert. Mittels der Sprachanalyse von Abbott wurden daraufhin ein Objektmodell und ein Aktivitätsdiagramm erstellt. Die Erstellung dieser Diagramme half, die Problemwelt sowohl auf statischer als auch dynamischer Art und Weise besser zu verstehen und dem Kunden in der abschließenden Präsentation des Analyseprozesses zu veranschaulichen. Zusätzlich wurde ein Mockup erstellt, um die Benutzeroberfläche und das geplante Interaktionsdesign visuell darzustellen.

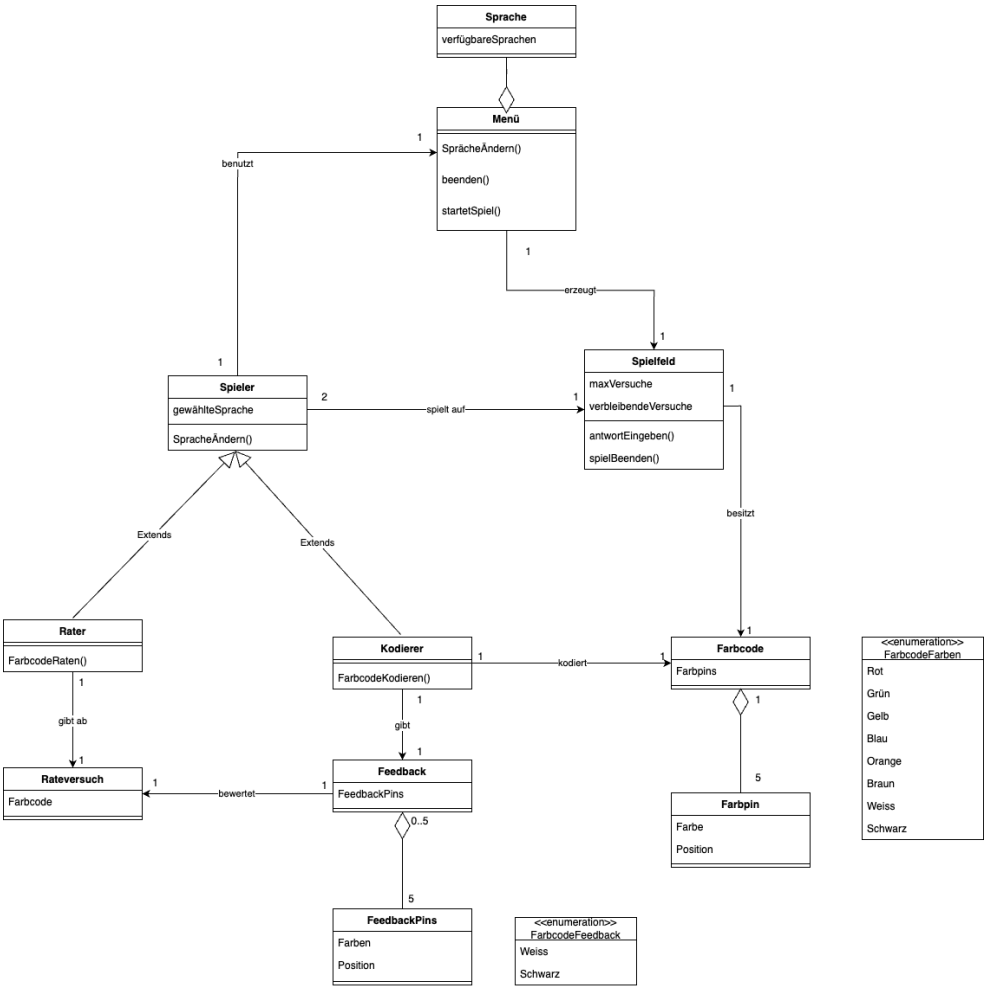
Use Case Diagramm



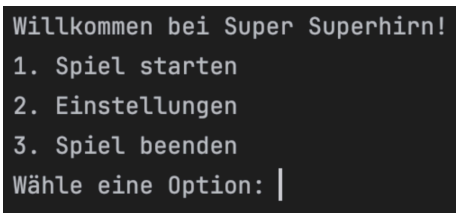
Aktivitätsdiagramm

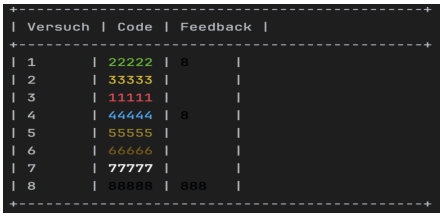


Objektmodell der Analysephase



Mockup





3. Anforderungsänderung der Analysephase

Im Vorfeld der Präsentation zum Analyseprozess wurde seitens des Kunden eine Anforderungsänderung dargelegt, welche in den nachfolgenden Phasen zu berücksichtigen ist. Die vorgenannte Anforderungsänderung lautet wie folgt:

1. Als Rater möchte ich eine laufende Partie jederzeit unterbrechen können.
2. Als Rater möchte ich nach einer Unterbrechung ins Hauptmenü zurückkehren.
3. Als Rater möchte ich eine unterbrochene Partie jederzeit vom Hauptmenü aus fortsetzen können.
4. Als Rater möchte ich, dass das Spielbrett beim Fortsetzen der Partie genauso aussieht wie vor der Unterbrechung.
5. Als Rater möchte ich eine wieder aufgerufene Partie erneut unterbrechen können.
6. Als Rater möchte ich, dass zu jedem Zeitpunkt nur eine Partie unterbrochen werden kann.
7. Als Rater möchte ich beim Versuch, eine zweite Partie zu unterbrechen, eine Warnung erhalten, dass die erste unterbrochene Partie verloren geht. Ich möchte dann die Möglichkeit haben zu entscheiden, ob ich fortfahre oder nicht.

4. Design

4.1 Designprozess

Im Zuge der Erstellung eines Designs, das auf den in der Analysephase erhobenen und modellierten Anforderungen basiert, erfolgte zunächst eine Ausformulierung der nicht-funktionalen Anforderungen. Ziel war es, eine geeignete Architektur zu identifizieren, die dazu beiträgt, die Erfüllung dieser Anforderungen zu gewährleisten. Dabei sind diese wie folgt definiert:

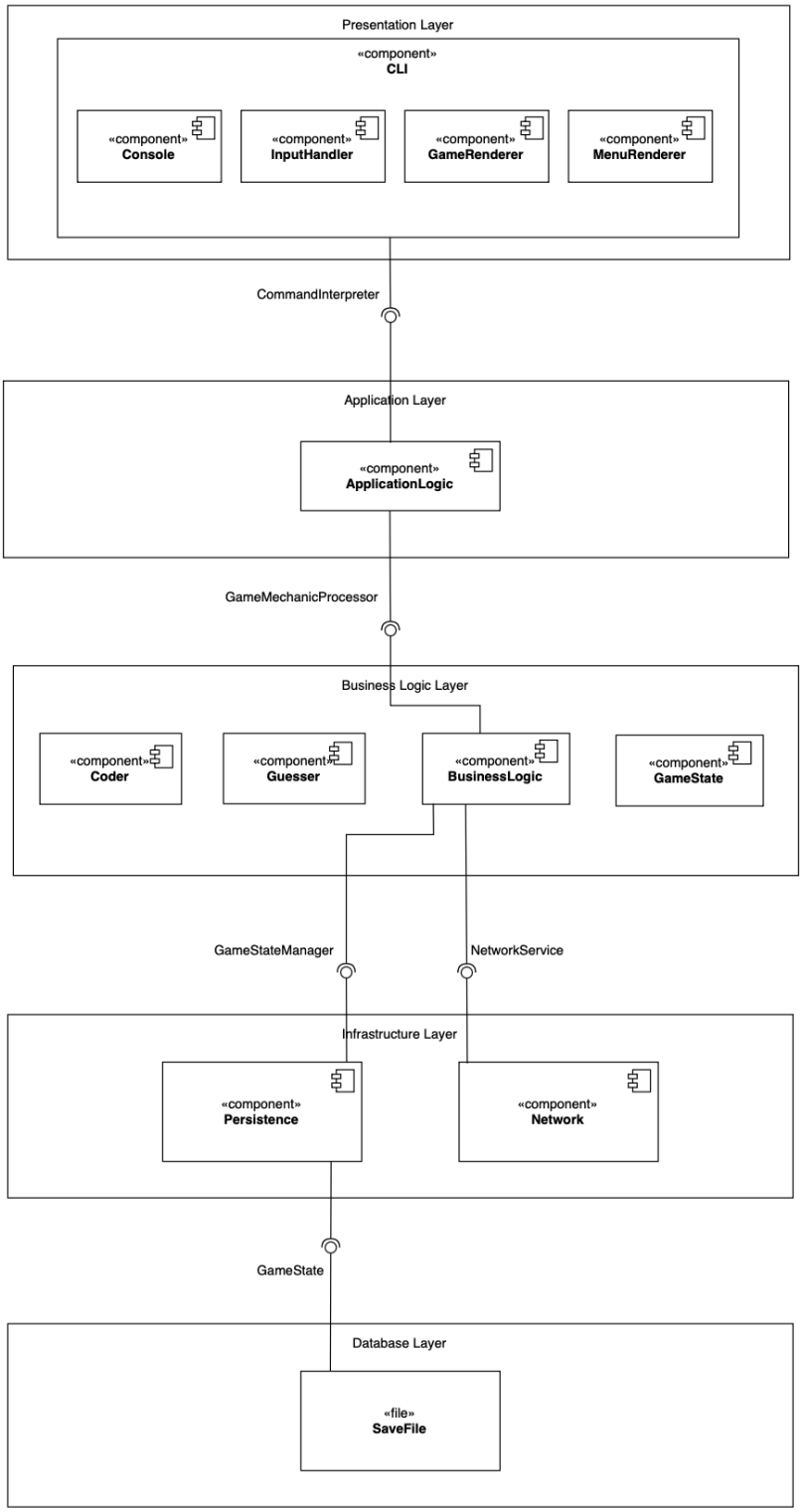
1. **Skalierbarkeit und Erweiterbarkeit**: Das System sollte so gestaltet sein, dass neue Funktionalitäten problemlos hinzugefügt werden können, ohne bestehende Funktionen zu beeinträchtigen. Ebenso sollte es flexibel an veränderte Anforderungen angepasst werden können.
2. **Benutzerfreundlichkeit**: Die Struktur des Systems sollte so gestaltet sein, dass eine klare Trennung zwischen der Benutzerinteraktion und der eigentlichen Anwendungslogik ermöglicht wird, um eine intuitive Bedienbarkeit sicherzustellen.
3. **Zuverlässigkeit**: Zur Sicherstellung einer hohen Zuverlässigkeit ist eine klare Struktur erforderlich, die sowohl die Validierung von Eingaben als auch die Bereitstellung von Feedback für den Nutzer ermöglicht.
4. **Wartbarkeit**: Das System sollte einfach zu verstehen und zu modifizieren sein. Eine klare Trennung von Zuständigkeiten und eine gute Strukturierung der Komponenten waren daher essenziell.
5. **Portabilität**: Um die plattformübergreifende Nutzbarkeit zu gewährleisten, musste das System unabhängig von betriebssystemspezifischen Funktionen und Technologien gestaltet werden.
6. **Richtigkeit**: Die Architektur musste so gestaltet werden, dass sie umfassende Tests auf verschiedenen Ebenen (Einheit, Integration, Gesamtsystem) unterstützt.

Nach Evaluation der oben genannten Kriterien wurde zunächst eine Event-Driven Architecture in Betracht gezogen. Die ausschlaggebenden Kriterien für diese Überlegung waren die lose Kopplung, die Flexibilität, die Modularität, die klare Verantwortlichkeitsverteilung und die gute Testbarkeit der einzelnen Event-Handler. Allerdings wurde der Aspekt der Machbarkeit im Rahmen der zeitlichen Begrenzung sowie die erhöhte Komplexität dieser Architektur nicht ausreichend berücksichtigt.

Daher wurde letztlich die Entscheidung getroffen, eine Schichtenarchitektur zu implementieren. Diese wäre dem Umfang des Projektes deutlich angemessener und bietet vergleichbare Vorteile, wie eine lose Kopplung zwischen den Schichten, die einfache Erweiterbarkeit durch das Hinzufügen neuer Funktionalitäten, eine klare Modularität und Verantwortlichkeitsverteilung sowie eine gute Unterstützung bei der Testbarkeit. Abschließend haben wir ein Komponentendiagramm erstellt, um unsere Architektur zu beschreiben. Dieses wurde dem Kunden am Ende des Designprozesses vorgestellt.

4.2 Beschreibung des Designs

Komponentendiagramm der Schichtenarchitektur



Unsere Schichtenarchitektur ist in die folgenden Schichten und Module unterteilt, wobei jede Schicht und jedes Modul eine spezifische Aufgabe erfüllt

1. **Presentation Layer:** Der zentrale Fokus des Presentation Layers liegt in der Bereitstellung einer Schnittstelle zwischen dem Nutzer und dem System. In dieser Funktion agiert der Presentation Layer lediglich als Intermediär, indem er die Benutzereingabe empfängt und an die nachfolgende Schicht weiterleitet. Dabei haben wir die folgenden Module implementiert innerhalb dieser Schicht:
    - Console: Koordinator aller UI-Komponenten und verwaltet den Haupt-Spielablauf
    - Menu Renderer: Zuständig für die Anzeige aller Menüs
    - Game Renderer: Zuständig für die Visualisierung des Spielstands
    - Input Handler: Verarbeitet Benutzereingaben

2. **Application Layer:** Ist für die Steuerung der Ablauflogik der Anwendung zuständig und die Koordination zwischen den verschiedenen Schichten. Sie bildet eine Schnittstelle zwischen dem Presentation Layer und dem Business Layer.

3. **Business Layer:** Ist der zentrale Ort für die gesamte Spiellogik und zeichnet sich durch die vollständige Abkapselung der Kernlogik und der Spielregeln von den anderen Schichten aus. Hier werden alle relevanten Berechnungen und Prozesse durchgeführt, um das Spiel gemäß den definierten Regeln zu steuern und den Spielablauf sicherzustellen. Innerhalb der dieser Schicht befinden sich die folgenden Module:
    - Business Logic: übernimmt die zentrale Koordination der Spiellogik und die Verwaltung des Spielstands. Sie integriert die Module innerhalb dieser Schicht und koordiniert zudem die Interaktion mit der darunterliegenden Schicht.
    - Guesser: umfasst sowohl die Implementierung für den menschlichen Rater als auch die für den Computer-Rater, der den Knuth-Five-Guess-Algorithmus verwendet.
    - Coder: umfasst sowohl die Implementierung für den menschlichen Kodierer als auch für den Computer-Kodierer, der zufällig eine gültige Geheimfarbkombination generiert.
    - Game Turn und Game State: dienen als Zustandsverwaltung und repräsentieren sowohl einen einzelnen Spielzug als auch den aktuellen Spielzustand des Spiels.

4. **Infrastructure Layer:** Ist verantwortlich für die Bereitstellung der technischen Infrastruktur sowie für die Verwaltung der externen Kommunikation der Anwendung. Dabei besteht es aus den folgenden zwei Modulen:
    - **Network:** ermöglicht die Online-Spiel-Funktionalität, indem sie die Verbindung zum Spielserver über HTTP verwaltet und Daten zwischen dem Spiel-Format und dem HTTP-Format konvertiert. Zusätzlich sorgt sie für die Validierung der Daten und behandelt mögliche Verbindungsfehler zuverlässig.
    - **Persistence:** verantwortlich für das Speichern und Laden von Spielständen und kapselt alle Dateisystem-Zugriffe.

5. **Database Layer:** übernimmt die Speicherung des Spielzustandes in einem trivialen Datei-basierten Speicher.

5. Anforderungsänderung der Designphase

Im Vorfeld der Präsentation zum Designprozess wurde seitens des Kunden eine weitere Anforderungsänderung dargelegt, welche in den nachfolgenden Phasen zu berücksichtigen ist. Die vorgenannte Anforderungsänderung lautet wie folgt:

1. Als Spieler möchte ich das Spiel über das Internet mit anderen spielen können.
2. Als Rater möchte ich meine Bewertungen über das Netz erhalten.
3. Als Spieler möchte ich, dass der Computer-Codierer lokal oder über das Internet agieren kann.
4. Als Spieler möchte ich eine IP-Adresse und einen Port für die Verbindung hinterlegen.
5. Als Rater möchte ich, dass der Code von einem entfernten Codierer kommt.
6. Als Rater möchte ich eine gamerid auswählen können.
7. Als Entwickler möchte ich, dass der Rater eine gameid erhält und diese bei jedem Rateversuch nutzt.
8. Als Entwickler möchte ich sicherstellen, dass die gamerid & gameid bei jedem Rateversuch gleich bleiben.
9. Als Entwickler möchte ich, dass die gameid = 0 ein neues Spiel startet.
10. Als Entwickler möchte ich, dass die Kommunikation über HTTP POST erfolgt.
11. Als Entwickler möchte ich, dass application/json als Content-Type verwendet wird.
12. Als Entwickler möchte ich, dass Requests und Responses ein einheitliches JSON-Format haben.

6. Implementierung

In diesem Abschnitt werden die Konzepte der objektorientierten Programmierung und die eingesetzten Technologien detailliert beschrieben. Darüber hinaus werden die wichtigsten Methoden und deren Aufgaben erläutert, die für die Entwicklung eines funktionierenden Spiels essentiell waren. Bei der Umsetzung des Spiels wurden mehrere klar definierte Schritte verfolgt, um eine strukturierte und effiziente Entwicklung sicherzustellen.

6.1 Verwendete Konzepte und Technologien

Im Rahmen der Implementierung wurden die Konzepte der Kapselung, Abstraktion, Vererbung und Polymorphismus angewendet. Darüber hinaus fanden das Separation of Concern und das Dependency Inversion Principle Anwendung. Das Spiel “Super Superhirn” wurde in der Programmiersprache Python 3 entwickelt mit Hilfe der Entwicklungsumgebungen PyCharm.

6.2 Ablauf der Implementierung

In Anbetracht des gewählten Vorgehensmodells sowie der implementierten Schichtenarchitektur wurden für die 4-wöchige Implementierungsphase die folgenden klar definierten Sprints festgelegt:

1. Sprint: Der initiale Sprint erstreckte sich über einen Zeitraum von sieben Tagen und zielte darauf ab, die vollständige Implementierung der CLI des Presentation Layers sowie der gesamten Application Layer abzuschließen.
2. Sprint: Im Rahmen des zweiten Sprints, welcher sich wiederum über einen Zeitraum von sieben Tagen erstreckte, wurde das klar definierte Ziel verfolgt, die Game-Logik sowie die Rater- und Kodierer-Logik des Spielers und des Computers zu implementieren.
3. Sprint: Aufgrund der Weihnachtsferien erstreckte sich der dritte Sprint über einen Zeitraum von 14 Tagen. Ziel war die Implementierung des vollständigen Infrastructure Layer, welcher die Integration des Persistence Managers, des Network Service umfasste. Im Zuge dessen wurde auch unsere triviale Database Layer mit unserem save file erstellt.
4. Sprint: In den zurückliegenden sieben Tagen wurde das Hauptaugenmerk auf die Überprüfung und Anwendung der Qualitätssicherung gelegt. Dies beinhaltete das Refactoring des Codes, die Sicherstellung der Erfüllung der PEP-8-Anforderungen sowie die Überprüfung des erfolgreichen Durchlaufs der finalen Tests und die Auswertung der Dokumentation.

7. Qualitätssicherungsmaßnahmen

Im Rahmen der Qualitätssicherung-Strategie wurden die folgenden Technologien eingesetzt, um die nachfolgend aufgeführten Maßnahmen zu unterstützen: Die CI/CD-Pipeline-Funktionalität von GitLab, welche Flake8, ein Python-Tool zur statischen Code-Analyse, und Pytest, ein Testing-Framework, einsetzt. Zusätzlich wurde lokal von jedem Gruppenmitglied der Code-Formatter Black verwendet.

CI/CD-Pipeline:
Bei jedem Commit wird der gepushte Code durch eine CI/CD-Pipeline von GitLab auf Korrektheit überprüft. Im Zuge dessen werden alle Tests automatisch durchgeführt, um sicherzustellen, dass bestehende Funktionalitäten nicht beeinträchtigt werden. Zusätzlich wird der Code beim Merge auf die Einhaltung der PEP8-Richtlinien geprüft. Die Testabdeckung wird angezeigt, um gegebenenfalls auf unzureichende Tests hinzuweisen und sicherzustellen, dass kritische Komponenten ausreichend geprüft werden. Wir haben festgelegt, dass die Line Coverage mindestens 90% betragen muss.

Test Driven Development (TDD) und Pair Programming:
Eine zentrale Maßnahme war die Kombination von Test Driven Development (TDD) und Pair Programming, bei der immer ein Gruppenmitglied zuerst die Tests für eine neue Funktionalität schreibt, bevor die Implementierung erfolgt. Das zweite Gruppenmitglied verfolgt die Entwicklung per Bildschirmübertragung und gibt aktiv Verbesserungsvorschläge oder Ideen zur Umsetzung. Die Rollen wechseln dynamisch und flexibel, um eine gleichmäßige Verteilung der Aufgaben zu gewährleisten und die Codequalität zu erhöhen.

Code Review und Merge-Prozess:
Nach Abschluss der Implementierung und des Testens erstellt das verantwortliche Paar einen Merge-Request. Die anderen beiden Teammitglieder übernehmen den Review-Prozess, um sicherzustellen, dass die Anforderungen korrekt umgesetzt und keine Fehler übersehen wurden. Wenn der Code die Prüfung besteht, wird der Branch in den aktuellen Sprint integriert. Nach dem Sprint Review wird der Sprint Branch in den Main-Branch gemerged und der nächste Sprint vorbereitet.

Scrum-Prozess und Trello-Board:
Die Qualitätssicherung wurde durch den Einsatz von Scrum als agiles Vorgehensmodell unterstützt. Jeder Sprint begann mit der Priorisierung und Planung der Aufgaben im Backlog. Das Trello-Board spielte dabei eine entscheidende Rolle, um den Fortschritt zu visualisieren und alle anstehenden Aufgaben klar zu strukturieren. Dies half nicht nur bei der Organisation der Arbeit, sondern stellte sicher, dass keine kritischen Aufgaben übersehen wurden. Wöchentliche Sprint-Meetings und Retrospektiven ermöglichten es uns, kontinuierlich die Effizienz zu steigern, Fehlerquellen zu identifizieren und den Entwicklungsprozess zu optimieren.

Durch diese Maßnahmen wurde sichergestellt, dass die Qualität der Software kontinuierlich verbessert und neue Anforderungen effizient umgesetzt werden konnten. Die enge Verzahnung von automatisierten Tests, Pair Programming, der CI/CD-Pipeline und Scrum führte zu einem stabilen und wartbaren Code.

8. Fazit

Das Projekt "Super Superhirn" stellte eine wertvolle Erfahrung im Bereich der Softwareentwicklung dar und ermöglichte tiefgehende Einblicke in die Umsetzung von Architekturmustern sowie die Anwendung agiler Methoden, insbesondere Scrum.

Zu Beginn des Projektes wurde konsequent der Test-Driven Development (TDD)-Ansatz praktiziert. Für jede neue Funktionalität wurden zunächst Tests erstellt, bevor die Implementierung erfolgte. Diese Vorgehensweise resultierte in einer stabilen und umfassend getesteten Codebasis in der Anfangsphase. Mit zunehmender Komplexität der Anforderungen war jedoch eine Umstellung auf einen anderen Ansatz erforderlich, bei dem zunächst die Implementierung erfolgte und die Tests erst im Anschluss ergänzt wurden. Die Kombination dieser beiden Ansätze ermöglichte es, flexibel auf neue Herausforderungen zu reagieren und auch komplexe Anforderungen effizient umzusetzen.

Die Entscheidung, eine Schichtenarchitektur anstelle der ursprünglich geplanten Event-Driven Architecture zu verwenden, wurde nach Rücksprache mit dem Kunden Herrn Prof. Dr. Salinger getroffen. Der Hauptgrund für diese Entscheidung lag in der einfacheren Implementierung der Schichtenarchitektur, die sich besser mit den verfügbaren Ressourcen und dem Umfang des Projektes vereinbaren ließ. Die Aufteilung in Presentation Layer, Application Layer, Business Layer, Infrastructure Layer und Database Layer ermöglichte eine klare Trennung der Verantwortlichkeiten und begünstigte die modulare Erweiterbarkeit und Wartbarkeit der Anwendung.

Der Einsatz des agilen Vorgehensmodells Scrum unterstützte die Organisation des Projektes maßgeblich. Zu Beginn jedes Sprints erfolgte die Erstellung und Priorisierung der Aufgaben im Sprint Backlog. Wöchentliche Sprint-Meetings und Retrospektiven dienten der Bewertung von Fortschritten

und der kontinuierlichen Verbesserung des Entwicklungsprozesses. Das Trello-Board erwies sich dabei als ein essentielles Werkzeug, um die Aufgaben zu visualisieren, zu verwalten und die Übersicht über den aktuellen Stand des Projektes zu behalten.

Die durch Scrum erreichte klare Strukturierung des Projektes ermöglichte eine flexible Reaktion auf neue Anforderungen und eine Sicherstellung der einheitlichen Kenntnis des Projektinhalts durch alle Teammitglieder. Die CI/CD-Pipeline stellte eine zentrale Komponente des Qualitätssicherungskonzeptes dar, da sie die kontinuierliche Ausführung von Tests und die Sicherstellung der PEP8-Konformität des Codes ermöglichte. Obwohl die Pipeline in der Anfangsphase des Projektes vorteilhaft war, führte sie gegen Ende des Projektes zu Verzögerungen. Die strengen Prüfungen auf PEP8 führten zu einer erheblichen Verlangsamung bei der Integration neuer Features und bremsten den Entwicklungsprozess aus.

Das Pair Programming war ebenfalls ein fester Bestandteil des Projektes. Trotz der Herausforderung, die Terminfindung innerhalb des Teams zu koordinieren, erwies sich dieser Ansatz als außerordentlich effektiv. Er ermöglichte es, Fehler frühzeitig zu erkennen und die Codequalität durch gegenseitiges Feedback zu verbessern.

Das Projekt war insgesamt ein Erfolg. Es hat nicht nur technische, sondern auch organisatorische Fortschritte ermöglicht. Die Wahl der Schichtenarchitektur, der Einsatz von Scrum und TDD sowie die kontinuierliche Qualitätssicherung durch Pair Programming und CI/CD führten zu einer stabilen, skalierbaren und gut getesteten Anwendung, die alle Anforderungen erfüllt.

Was hätten wir besser machen können?
Obwohl das Projekt erfolgreich abgeschlossen wurde, gab es einige Aspekte, die im Nachhinein optimiert werden könnten:

- **CI/CD-Pipeline:** Die CI/CD-Pipeline hat sich als zweischneidiges Schwert erwiesen. Während sie die Codequalität sicherstellte, bremsten strenge Prüfungen gegen Ende den Entwicklungsprozess aus. An dieser Stelle hätten wir beim ersten Schreiben des Codes besser darauf achten sollen, dass dieser PEP8 konform ist.
- **Optimierung der Scrum-Termine:** Die Terminfindung für Pair Programming und Sprint-Meetings stellte sich gelegentlich als schwierig heraus. Eine striktere Planung und frühzeitige Festlegung der Termine hätten hier zu einer besseren Verfügbarkeit und weniger Verzögerungen geführt.
- **Frühzeitige Klärung der Architektur:** Die Entscheidung für die Schichtenarchitektur fiel erst nach der Evaluation der Event-Driven Architecture. Eine frühere Auseinandersetzung mit den Vor- und Nachteilen beider Ansätze hätte möglicherweise Zeit gespart und eine klarere Ausrichtung im Designprozess ermöglicht.
- **Mit der Businesslogic beginnen:** Wir haben mit der Implementierung der CLI begonnen. Angemessener wäre es gewesen, zuerst die Businesslogik zu entwickeln.