Hochschule für Technik und Wirtschaft Berlin

Fachbereich 4 | Angewandte Informatik

Modul: Software Engineering II

Dozent: Prof. Dr. Stephan Salinger

Projektbeschreibung

Software Engineering II

WiSe 25/26

Superhirn

Gruppe 21

1. Einleitung

Im Rahmen der Veranstaltung Software Engineering II an der HTW Berlin wurde das Projekt Superhirn entwickelt. Die Anforderungen an das Projekt bzw. die Aufgabe ist ursprünglich gestellt von Prof. Dr. Stephan Salinger, der im Rahmen der Veranstaltung als Kunde fungiert hat.

Ziel ist die Konzeption und Umsetzung einer erweiterbaren, gut strukturierten Anwendung basierend auf dem Spiel Mastermind. Dabei wird in diesem Dokument auf den gesamten Entwicklungsprozess der Anwendung eingegangen. Die dazugehörigen Modelle, die Architektur der Anwendung, unter anderem der Ablauf und Details der Implementierungsphase sowie der Test Phase.

2. Grobe Beschreibung der Aufgabe

Wie bereits erwähnt, ist das Ziel eine Anwendung, in der man das Spiel Mastermind spielen kann. Konkret wurden uns mehrere funktionale und nicht-funktionale Anforderungen vom Kunden genannt, dies war dann Grundlage für die Entwicklung der Anwendung. Das grobe Vorgehen richtete sich dabei klassisch nach den Phasen einer Softwareentwicklung (Planung- Analyse- Design- ….). Darauffolgend waren anfangs wöchentlich Präsentationen angesetzt, um dem Kunden den Fortschritt des Prozesses aufzuzeigen und eine Grundlage für die Implementierungsphase zu schaffen. Ein wichtiger Aspekt dabei ist, dass die Anforderungen sich ändern können oder neue hinzukommen können, dies muss man beachten und dementsprechend die theoretischen Aspekte anpassen. Abschließend ist eine praktische und theoretische Präsentation geplant. Zum einen soll das Programm selbst vom Kunden abgenommen werden und bei der theoretischen Präsentation soll der ganze Entwicklungsprozess und bestimmte interessante Merkmale konkret aufgezeigt werden.

3. Teammitglieder

- **Hamdi Hawari (593118):** Head of Req. Engineering/ Head of Project Coordination
- **Johannes Huber (594626):** Software Architect
- **Daniel Kujawa (594429):** Lead Developer
- **Nikolaj Tkatschew (592564):** Head of Quality Ensurance

4. Beschreibung des gewählten Software Prozesses

Für das Projekt Superhirn wurde ein iteratives und inkrementelles Vorgehen gewählt, das sich an klassischen Phasen der Softwareentwicklung orientiert, jedoch bewusst flexibel umgesetzt wurde. Zu Beginn stand eine Planungs- und Analysephase, in der die fachlichen Anforderungen aus Sicht der Spieler erfasst und in Use Cases sowie ein Objektmodell überführt wurden. Auf dieser Grundlage wurden in der Entwurfsphase eine Architektur sowie ein Design entwickelt, das klare Verantwortlichkeiten und Schnittstellen definiert. Die Implementierung erfolgt schrittweise auf Basis des Entwurfs. Dabei wurden einzelne Komponenten unabhängig voneinander umgesetzt. Neue oder angepasste Anforderungen, insbesondere die nachträglich hinzugekommene Online-Anforderung, konnten durch das modulare Design und die saubere Schichtentrennung integriert werden, ohne das bestehende Domänenmodell oder die Spiellogik grundlegend zu verändern.

Insgesamt war der SW- Prozess sehr “SCRUM” ähnlich. Es wurden Aufgaben gestellt, die unabhängig von den jeweiligen Deadlines bearbeitet wurden. Daraufhin hat jeder seine Aufgaben erledigt und abschließend wurde ein kurzes Meeting gehalten (am Ende der Woche), bei welchem die Aufgaben einzeln besprochen wurden, bezüglich ihrer Vollständigkeit bzw. Korrektheit oder Problemen die aufgetreten sind und geklärt werden müssen. Basierend auf den Ergebnissen des Meetings wurden neue Aufgaben gestellt oder alte überarbeitet, bei existierendem Bedarf. Das gleiche gilt auch für neue Anforderungen, die uns kommuniziert wurden.

5. Analyse

5.1 Beschreibung des gewählten Analyseprozesses

In der Analysephase führten wir auf Basis der bereitgestellten Dokumente und Daten eine Gruppenerhebung durch. Die entsprechenden Dokumente wurden vom Kunden zur Verfügung gestellt. Zusätzlich wurden dem Kunden persönliche Fragen gestellt, um bestehende Unklarheiten zu klären. Abschließend wurden auf Grundlage aller gesammelten Informationen Use Cases sowie ein Use-Case-Diagramm erstellt.

5.2 Beschreibung der Analyseergebnisse

Spielkonzept

Superhirn ist ein Logikspiel, bei dem ein Geheimcode aus Farben erraten werden muss. Jeder Rateversuch wird durch ein Feedback bewertet:

- **schwarz**: richtige Farbe am richtigen Platz
- **weiß**: richtige Farbe am falschen Platz

Unterstützte Varianten:

Superhirn

- 4 Steckplätze
- 6 Farben

Super-Superhirn

- 5 Steckplätze
- 8 Farben (inkl. Schwarz und Weiß)

Die Varianten unterscheiden sich ausschließlich in der Anzahl der Steckplätze und Farben.

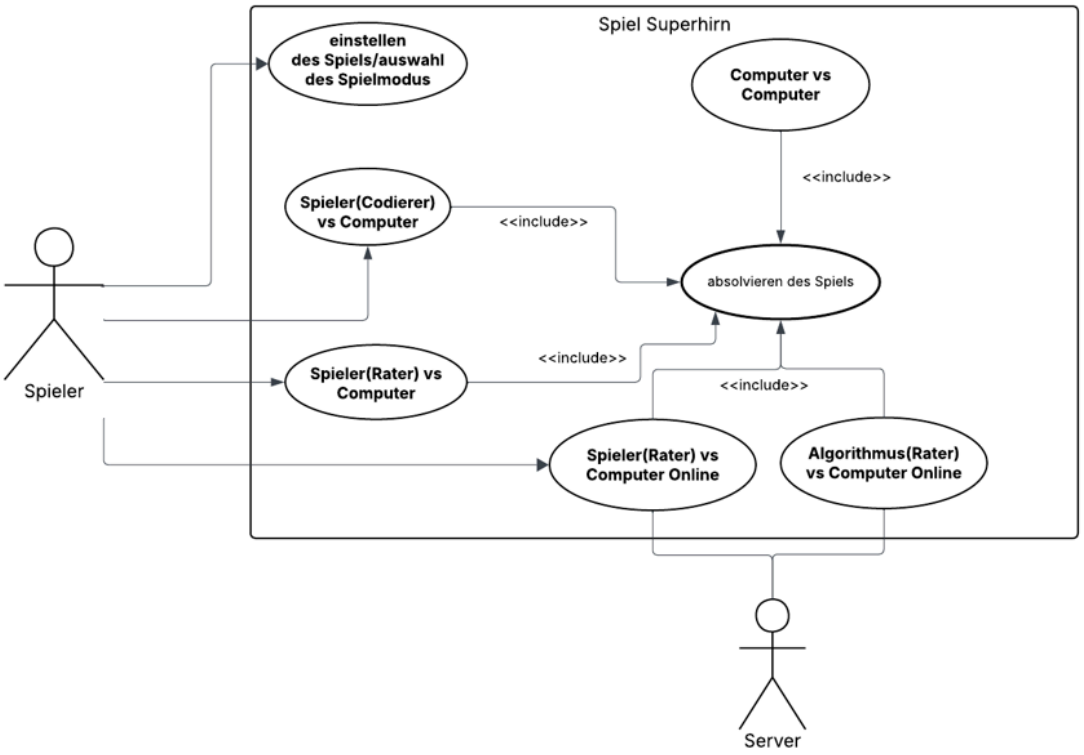
Online-Anbindung (neue Anforderung)

Das System soll so konzipiert werden, dass es auch über das Internet gespielt werden kann. Die Kommunikation erfolgt konzeptionell über HTTP-POST Anfragen und JSON Nachrichten. Spielzüge werden als JSON Objekte übertragen

Ein externer Server übernimmt dabei:

- die Vergabe einer **GameId**
- die Erstellung eines Geheimcodes, welchen nur der Server kennt
- die Bewertung von Rateversuchen

Spiel Konstellationen - Modi



Mensch–Computer

Im Mensch–Computer-Modus übernimmt der menschliche Spieler die Rolle des Codierers, während der Computer als Rater agiert. Der Mensch legt den Geheimcode fest und kann optional Parameter für den Computer-Algorithmus (z. B. Algorithmus-Typ, Verzögerung) konfigurieren.

Spielablauf:

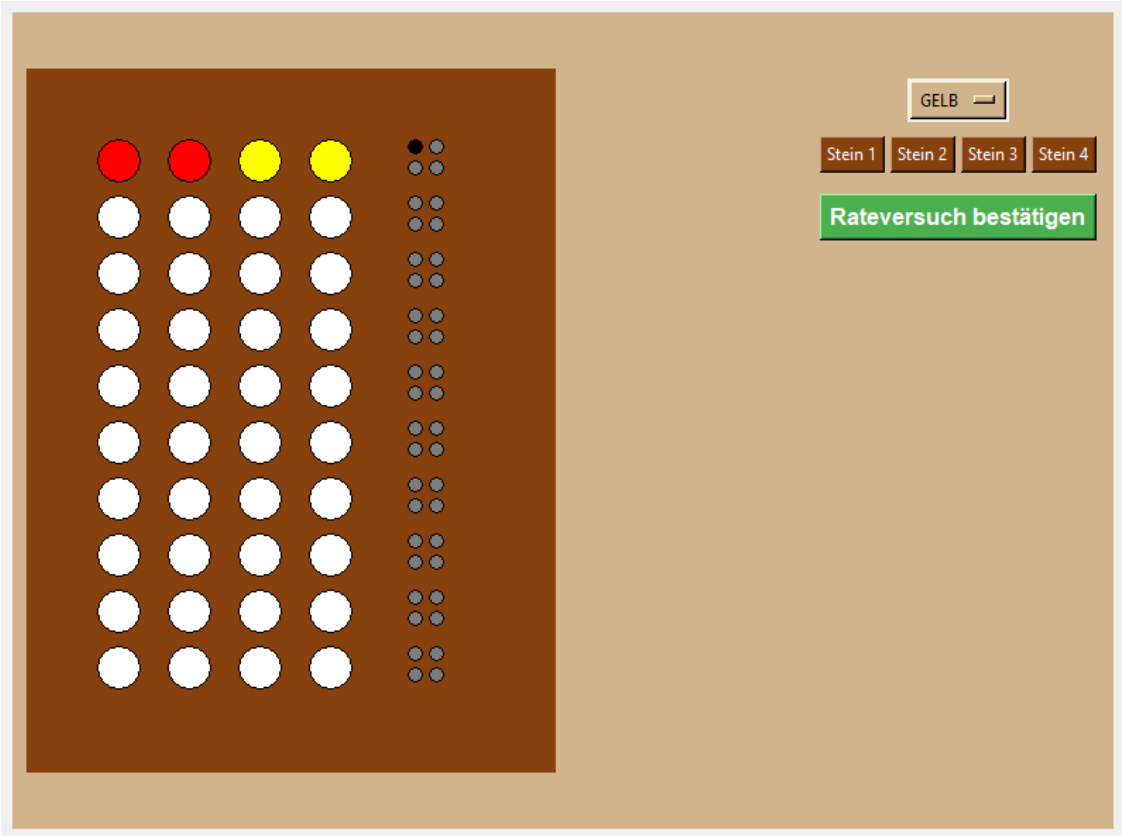
- Der Mensch setzt einen Farbcode
- Der Computer startet als Rater und führt automatisierte Rateversuche durch:
    - Der ausgewählte Algorithmus generiert einen Rateversuch (z. B. ["GRÜN", "ORANGE", "GRÜN", "GELB"]).
    - Feedback : vergleicht den Rateversuch mit dem Geheimcode und gibt ein Feedback zurück (z. B. schwarz = 1, weiß = 2).
    - Der Algorithmus nutzt das Feedback, um den nächsten Rateversuch zu optimieren.
- Der Prozess wiederholt sich, bis der Computer den Code knackt oder die maximale Rundenzahl erreicht ist.

Computer-Mensch

Im Computer–Mensch-Modus agiert der Computer als Codierer, während der menschliche Spieler als Rater fungiert. Der Computer generiert einen zufälligen Geheimcode, und der Mensch hat 10 Runden Zeit, diesen zu erraten. Dies tut er, indem er jede Runde einen Rateversuch abgibt.

Spielablauf:

- Der Computer generiert einen Farbcode
- Der menschliche Spieler gibt pro Runde einen Rateversuch ein (z. B. ["ROT", "ROT", "GELB", "GELB"]).
- Der FeedbackBerechner vergleicht den Rateversuch mit dem Geheimcode und gibt ein Feedback zurück (z. B. schwarz = 1, weiß = 1).
- Der Spieler nutzt das Feedback, um seinen nächsten Rateversuch zu optimieren.
- Der Prozess wiederholt sich, bis der Spieler den Code knackt oder die 10 Runden aufgebraucht sind.



Computer–Computer (Beobachtung Szenario)

Im Computer–Computer-Modus (C–C) agiert der Computer sowohl als Codierer als auch als Rater. Der menschliche Spieler hat hier keine aktive Spielrolle, sondern fungiert als Beobachter des automatisierten Spielablaufs. Seine einzige Aufgabe besteht darin, den Algorithmus für den Rater auszuwählen

Spielablauf:

- Der Computer startet als Codierer und erzeugt den Spielcode, als Rater führt er dann automatisierte Rateversuche durch:
    - Der ausgewählte Algorithmus generiert einen Rateversuch (z. B. ["GRÜN", "ORANGE", "GRÜN", "GELB"]).
    - Feedback : vergleicht den Rateversuch mit dem Geheimcode und gibt ein Feedback zurück (z. B. schwarz = 1, weiß = 2).
    - Der Algorithmus nutzt das Feedback, um den nächsten Rateversuch zu optimieren.
- Der Prozess wiederholt sich, bis der Computer den Code knackt oder die maximale Rundenzahl erreicht ist.

Mensch-Computer VS. (erweitertes M-CSzenario)

Ähnlich wie im Mensch-Computer-Modus agiert der Mensch als Codierer, während der Computer als Rater agiert. Der Mensch legt den Geheimcode fest und kann Parameter für den Computer-Algorithmus (z. B. Algorithmus-Typ, Verzögerung) konfigurieren. Jedoch kann der Codierer dies jetzt für zwei Spielbretter tun, sodass dann zwei Algorithmen beide gleichzeitig den Code des Codierers erraten müssen.

Spielablauf:

- Der Mensch startet als Codierer und legt einen Code fest, sowie die Algorithmen der beiden Spielbretter und die Verzögerung
- Die Computer starten als Rater und führen automatisierte Rateversuche durch
- Der Prozess wiederholt sich, bis einer der Computer den Code knackt oder die maximale Rundenzahl erreicht ist.

Computer-Mensch Online

Im Computer–Mensch-Online-Modus agiert ein externer Server als Codierer, während der menschliche Spieler lokal als Rater fungiert.

- Der Server generiert einen zufälligen Geheimcode und gibt das Feedback zurück.
- Der menschliche Spieler hat 10 Runden Zeit, den Code zu knacken, indem er Rateversuche abgibt.
- Die Kommunikation zwischen Client (Spielprogramm) und Server erfolgt über HTTP-POST-Anfragen mit einem definierten Protokoll.
    - Der Client sendet Rateversuche an den Server.
    - Der Server antwortet mit Feedback (schwarz/weiß).
    - Die Nachrichten werden in Spielcodes/Feedback übersetzt.

Computer-Computer Online

Im Computer–Computer-Online-Modus agiert ein externer Server als Codierer, während der lokale Computer mit einem ausgewählten Algorithmus versucht, den Code zu knacken.

- Der Server generiert einen zufälligen Geheimcode und gibt das Feedback zurück.
- Der lokale Computer nutzt einen konfigurierbaren Algorithmus, um den Code zu erraten.
- Die Kommunikation erfolgt über HTTP-POST-Anfragen mit einem definierten JSON-Protokoll.
- Der menschliche Beobachter kann den Spielverlauf verfolgen, ohne aktiv einzugreifen.

Nichtfunktionale Anforderungen:

- Erweiterbarkeit (neue Algorithmen, Varianten, Sprachen)
- Plattformunabhängigkeit, die Größe der Oberfläche und der weiteren Widgets in den Frames sind auf Windows, Linux und MacOS nutzbar

- Wartbarkeit (klare Verantwortlichkeiten, keine Redundanzen)
- Testbarkeit (einzelne Komponenten sind isoliert testbar
- Sprachunterstützung
    - Standardsprache: Deutsch
    - Optional: Englisch
    - Weitere Sprachen über externe Sprachdateien (nachladbar)

6. Entwurf – Systemarchitektur

6.1 Ziel der Entwurfsphase

Ziel der Entwurfsphase ist es, die fachlichen Anforderungen in eine klar strukturierte, erweiterbare Architektur zu überführen. Dazu werden die zentralen Systembausteine definiert, ihre Verantwortlichkeiten festgelegt und die Schnittstellen sowie die Kommunikationsregeln zwischen den einzelnen Schichten beschrieben. Der Entwurf bildet die Grundlage für die anschließende Implementierung und stellt sicher, dass das System später leicht erweiterbar ist und sich seine Komponenten isoliert testen lassen. Als Leitprinzipien gelten eine eindeutige Trennung der Schichten, sowie die Trennung des Domänenmodells von konkreter Technik, zusätzlich sind wohldefinierte Schnittstellen zwischen den Komponenten sicherzustellen.

6.2 Architekturübersicht (Schichtenmodell)

Das System folgt einer klaren Schichtenarchitektur, bei der jede Schicht ausschließlich mit ihren unmittelbar benachbarten Schichten kommuniziert.
Ziel dieses Aufbaus ist eine geringe Kopplung der Komponenten sowie eine hohe Austauschbarkeit und Testbarkeit einzelner Systemteile.

Den fachlichen Kern bildet der **Game-Layer**, der das Domänenmodell und die zentralen Spielregeln kapselt. Er enthält die wesentlichen fachlichen Klassen wie Spiel, Spielrunde, Spielbrett, Code und Feedback und ist vollständig unabhängig von technischen Aspekten wie Benutzeroberfläche oder Netzwerkkommunikation.

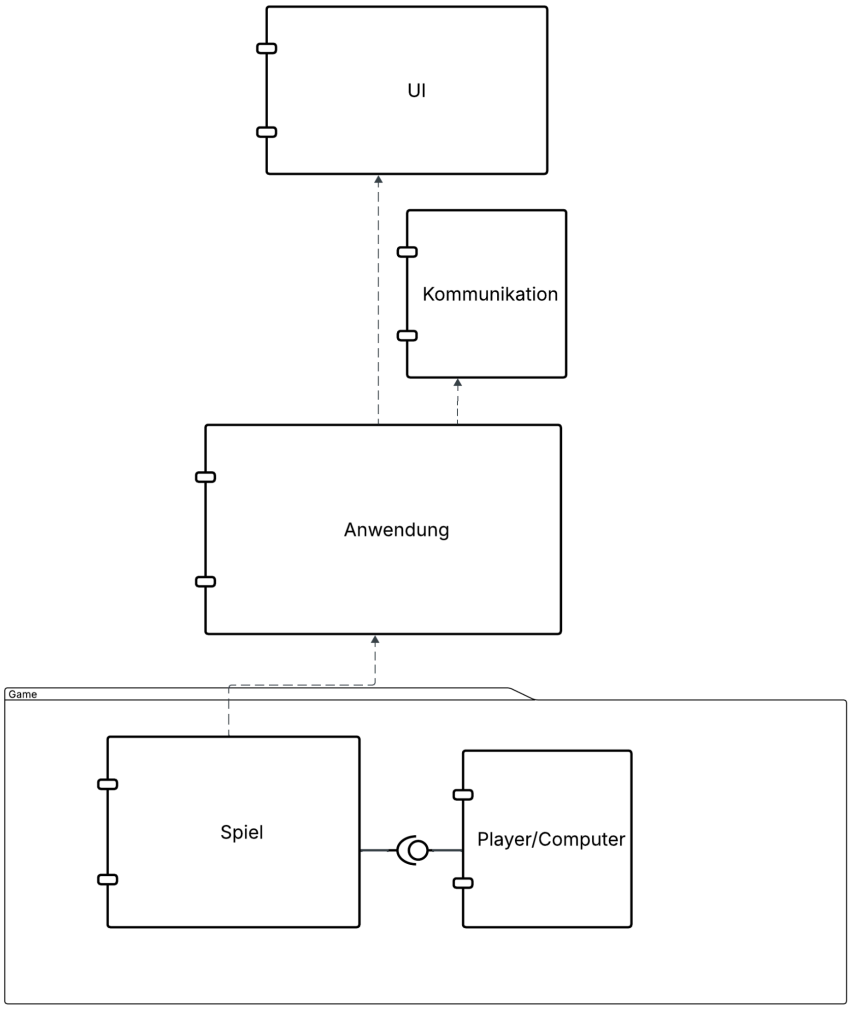
Der **Strategie- und Player-Layer** erweitert die fachliche Domäne um unterschiedliche Spielerrollen sowie austauschbare Ratealgorithmen und ist eine Subkomponente des Gamelayers. Es gibt eine Algorithmen Factory, die Algorithmen dem Spieler zuweist. So können neue Algorithmen hinzugefügt werden, ohne Änderungen an der Spiellogik vorzunehmen.

Der **Anwendungs-Layer** übernimmt die Orchestrierung des Spielablaufs.
Er initialisiert Spielinstanzen auf Basis von Spielparametern, steuert den Ablauf der Spielrunden und koordiniert die Interaktion zwischen UI, Game und falls aktiviert der Online-Kommunikation.

Der **Online-Kommunikations-Layer** ist optional und kapselt ausschließlich die technische Kommunikation mit externen Systemen. Er stellt dem Anwendungs-Layer eine klar definierte Schnittstelle zur Verfügung, ohne dass der Game-Layer von Online-Funktionalität oder technischen Protokollen abhängt.

Der **UI-Layer** bildet die oberste Schicht des Systems und ist für die Darstellung sowie die Entgegennahme von Benutzereingaben verantwortlich.
Er enthält keine fachliche Logik und greift nicht direkt auf das Domänenmodell zu.

Durch diese Architektur haben Änderungen an der Benutzeroberfläche oder an technischen Kommunikationsmechanismen keinen Einfluss auf die fachliche Spiellogik.
Alle Schichten sind klar voneinander getrennt, isoliert testbar und bei Bedarf austauschbar.



6.3 UI-Layer:

Der UI-Layer ist für die Darstellung des Spielzustands sowie die Erfassung von Benutzereingaben zuständig, ohne dabei selbst Logik oder Entscheidungen zu treffen. Seine Hauptaufgabe besteht darin, dem Spieler alle relevanten Informationen anzuzeigen, wie etwa den aktuellen Spielstand, die bisherigen Versuche, das Feedback zu den Zügen, die getätigten Eingaben und die verfügbaren Konfigurationsoptionen. Dabei bleibt die Darstellung rein neutral – sie übernimmt lediglich die vom Anwendungs-Layer bereitgestellten Daten und gibt sie unverändert wieder, ohne sie zu interpretieren oder zu manipulieren.

Gleichzeitig nimmt der UI-Layer Benutzereingaben entgegen, etwa die Auswahl von Spielvarianten, Modi oder Sprachen, die Konfiguration von Spieleinstellungen wie Farben, Algorithmen oder Verzögerungen sowie die Bestätigung von Spielzügen oder anderen Aktionen. Diese Eingaben werden direkt an den Anwendungs-Layer weitergeleitet, ohne dass eine Validierung oder Verarbeitung im UI-Layer stattfindet. Auf diese Weise bleibt der UI-Layer frei von Spiellogik, Regeln oder Algorithmen, alle fachlichen Aspekte werden an den Anwendungs-Layer delegiert, der die eigentliche Domäne steuert.

Die Struktur der Benutzeroberfläche ist in drei zentrale Bereiche unterteilt, die den Spielablauf abdecken und dem Nutzer eine intuitive Interaktion ermöglichen. Der Übersichtsbildschirm dient als Einstiegsseite und bietet dem Spieler die grundlegenden Spieloptionen vor dem Start an. Hier kann der Nutzer die gewünschte Spielvariante, den Spielmodus sowie die Sprache der Benutzeroberfläche auswählen. Sobald die Auswahl getroffen und bestätigt wurde, werden diese Informationen an den Anwendungs-Layer weitergegeben, der daraufhin das Spiel mit den gewählten Parametern initialisiert.

Im Anschluss folgt der Bereich der Spieleinstellungen, der eine detaillierte Konfiguration des Spiels ermöglicht. Hier kann der Nutzer spezifische Anpassungen vornehmen, wie etwa die Auswahl der Farben, sofern dies für den gewählten Spielmodus relevant ist, die Festlegung des Algorithmus oder die Einstellung eines Delays für die Spielabläufe. Auch diese Einstellungen werden nach der Bestätigung durch den Spieler an den Anwendungs-Layer übermittelt, der die entsprechenden Spielparameter setzt und das Spiel vorbereitet.

Sobald das Spiel gestartet ist, kommt die Spieloberfläche als Hauptansicht zum Einsatz. Sie zeigt den aktuellen Spielzustand an, inklusive der bisherigen Versuche, des erhaltenen Feedbacks sowie des Spielbretts oder anderer Spielelemente. Zudem bietet sie Optionen zur Steuerung des Spiels, sodass der Nutzer seine Züge ausführen oder weitere Interaktionen vornehmen kann. Jede Aktion wird direkt an den Anwendungs-Layer weitergeleitet, der die Gamelogik ansteuert und die Ergebnisse zurück an den UI-Layer liefert. Dieser aktualisiert daraufhin die Darstellung, sodass der Spieler stets den aktuellen Stand des Spiels vor Augen hat.

6.4 Anwendungs-Layer

Der Anwendungs-Layer bildet die Steuerungsebene des Systems und vermittelt zwischen der Benutzeroberfläche und der fachlichen Domäne. Seine primäre Aufgabe besteht darin,

den Spielablauf zu koordinieren, ohne selbst fachliche Logik zu enthalten. Er entscheidet, welche Komponenten in welcher Reihenfolge aktiv werden, leitet Benutzeraktionen an die Domäne weiter und sorgt dafür, dass die Ergebnisse korrekt an die UI zurückgemeldet werden.

Verantwortlichkeiten

Dieser Layer ist zuständig für:

- Initialisierung und Konfiguration des Spiels basierend auf den vom Nutzer gewählten Einstellungen.
- Auswahl des Spielmodus und die Zusammenstellung der benötigten Komponenten.
- Orchestrierung des Spielablaufs
- Weiterleitung von Eingaben aus der UI an die Domäne und Rückgabe der Ergebnisse in einem für die Darstellung geeigneten Format.
- Verwaltung von Spielsitzungen, einschließlich des Starts, der Unterbrechung und des Neustarts von Runden.

Struktur und Schnittstellen

Der Layer ist so konzipiert, dass er keine fachliche Logik enthält, sondern lediglich Abläufe steuert. Dazu nutzt er:

- Eine zentrale Steuerkomponente, die als Vermittler zwischen UI, Domäne und Com Layer fungiert. Sie empfängt Nutzeraktionen (z. B. Spielstart oder Parameteränderungen), leitet diese an die Domäne weiter und gibt die Ergebnisse strukturiert zurück.
- Modus-spezifische Ablaufsteuerungen, die je nach Spielvariante (z. B. Einzelspiel vs. Algorithmenvergleich) unterschiedliche Komponenten kombinieren. Dabei bleibt die fachliche Logik unverändert, nur die Reihenfolge der Aufrufe und die Kombination der Module passen sich an.
- Eine einheitliche Parameterstruktur, die alle konfigurierbaren Einstellungen (z. B. Spielvariante, Schwierigkeitsgrad, technische Optionen) bündelt und an die Domäne sowie die UI weitergibt. Diese Struktur stellt sicher, dass alle Komponenten mit denselben Grundlagen arbeiten, ohne direkte Abhängigkeiten voneinander zu haben.

6.5 Spiel-Layer

Der Spiel-Layer enthält die fachliche Kern Logik des Spiels und ist vollständig unabhängig von technischen Implementierungsdetails wie Benutzeroberflächen und Netzwerkkommunikation. Er definiert die Spielregeln, Datenstrukturen und Geschäftsprozesse, die das Spiel ausmachen, und stellt sicher, dass diese konsistent und wiederverwendbar sind.

Verantwortlichkeiten

Der Spiel-Layer ist verantwortlich für:

1. **Modellierung der fachlichen Objekte**
    - Abbildung der zentralen Spielkonzepte (z. B. Spiel, Runde, Brett, Code, Feedback) als Klassen mit klaren Verantwortlichkeiten.
    - Kapselung der Spielregeln und Validierungslogik (z. B. gültige Züge, Abbruchbedingungen, Gewinnbedingungen).
2. **Verwaltung des Spielzustands**
    - Führung des aktuellen Spielstands, inklusive der Historie, Spielerrollen und Rundenergebnisse.
    - Bereitstellung von Methoden zur Zustandsänderung (z. B. Zug ausführen, Runde beenden), die die Integrität der Daten sicherstellen.
3. **Berechnung von Spiellogik**
    - Durchführung fachlicher Berechnungen (z. B. Feedback-Berechnung, Gültigkeitsprüfung von Zügen).
    - Entkopplung dieser Logik von Steuerungs- oder Darstellung Aspekten.

6.6 Strategie-Layer

Dieses Subsystem ist Teil des Spiel-Layers und kapselt Rollen (Mensch/Computer) sowie austauschbare Ratealgorithmen und weist diese den Spielertypen zu. Auch die Implementierung der Algorithmen befindet sich in diesem Layer. Innerhalb des Spiel-Layers übernimmt das Subsystem die Generierung von Geheimcodes, insofern diese nicht vom menschlichen Spieler eingegeben wurden, sowie die Generierung von Rateversuchen mit Hilfe der implementierten Algorithmen. Es ist eine Schnittstelle definiert, über die die Spiellogik auf diesen Layer zugreifen kann. Somit soll es möglich sein, Rateversuche oder den Geheimcode zu generieren. Da dieser Layer ein Subsystem des Spiel-Layers ist, soll nur der Spiel-Layer darauf zugreifen.

Verantwortlichkeiten

Der Strategie Layer ist verantwortlich für:

1. **Kapselung der Spielerrollen**
    - beinhaltet Spielerklassen vom Typ Mensch und Computer, die von außen aufrufbar sein sollen

2. **Rate Algorithmen**

    - beinhaltet Implementierung von Ratealgorithmen, wobei das Hinzufügen von neuen Algorithmen einfach möglich ist
3. **Generierung von Codes**

    - die Computerklasse ist dazu in der Lage Geheimcodes zu generieren
    - der Spielertyp Computer bekommt einen Algorithmus zugewiesen, mit deren Hilfe er die Spielhistorie lesen und neue Rateversuche generieren kann

6.7 Online-/Kommunikations-Layer

Die Kommunikationsschicht soll eine Online-Spiel Funktionalität erfüllen, indem die Schicht die Kommunikation mit einem Server und der Anwendungs-Schicht ermöglicht. Dies soll über HTTP Post Anfragen geschehen, die Nachrichten werden dann verarbeitet in brauchbare Spielcodes, welche anschließend von der Kommunikationsschicht übergeben werden sollen. Zusätzlich erhält die Kommunikationsschicht von der Spiel-Schicht Spielcodes, welche dann in Nachrichten verarbeitet werden, um eine gegenseitige Kommunikation zwischen Spiel und Server zu ermöglichen.

6.8 Zentrale Entwurfsentscheidungen

- Klare Trennung zwischen Fachlogik und Technik
- Spielobjekte werden als eigenständige Klassen modelliert
- Algorithmen sind vollständig austauschbar
- Online-Funktionalität ist optional und modular integrierbar

7. Schnittstellen (Entwurfsphase)

Dieses Kapitel beschreibt die fachlichen und konzeptionellen Schnittstellen zwischen den einzelnen Schichten des Systems Superhirns.

Ziel ist es, eine klare Trennung der Verantwortlichkeiten sicherzustellen und direkte Abhängigkeiten zwischen nicht-benachbarten Schichten zu vermeiden.

Die Schnittstellen legen fest, welche Informationen zwischen den Schichten ausgetauscht werden, nicht wie dies konkret implementiert wird.

7.1 Überblick über die Schichten

Das System folgt einem klaren Schichtenmodell:

1. UI-Layer
2. Online-/Kommunikations-Layer
3. Anwendungs-Layer
4. Game-Layer mit Strategie-Layer

Jede Schicht darf nur über definierte Schnittstellen mit der darunterliegenden Schicht kommunizieren.

7.2 Schnittstelle: UI-Layer → Anwendungs-Layer

Die Schnittstelle zwischen dem UI-Layer und dem Anwendungs-Layer dient als klare Trennlinie zwischen der reinen Darstellung und der eigentlichen Spiellogik.

Der UI-Layer ist ausschließlich für die Anzeige von Informationen und die Entgegennahme von Benutzerinteraktionen verantwortlich. Er enthält selbst keine Regeln oder Logik des Spiels, sondern leitet Eingaben wie Spielstarts, Konfigurationen oder Rateversuche lediglich an die darunterliegende Anwendungsschicht weiter. Umgekehrt empfängt er von dort Spielstände und Feedback, um sie dem Nutzer anzuzeigen.

Die wichtigsten Komponenten auf Seiten der UI sind der Übersichtsbildschirm (zur Auswahl von Variante und Modus), die Spieleinstellungen (für Details wie Farben oder Algorithmus) sowie die Spieloberfläche, die den aktuellen Stand der Rateversuche und Rückmeldungen darstellt. Im Anwendungs-Layer übernehmen der SpielStarter (für die Initialisierung) und die SpielEngine (für Logik wie Codevergleich) die Verarbeitung.

Die Kommunikation erfolgt über definierte Datenstrukturen, vor allem die SpielParameter, die alle relevanten Einstellungen wie Variante, Modus, Algorithmus, Code und Verzögerungszeiten enthalten. Als zentrale Schnittstelle dient das Interface **StarterInt***,* das den UI-Layer in die Lage versetzt, ein Spiel zu starten, indem es eine passende Engine instanziiert. Dadurch bleibt die UI unabhängig von der internen Logik und kann ohne Auswirkungen auf die Spielmechanik angepasst oder erweitert werden.

Der UI Layer bedient sich der letzten Spielrunde über den Anwendung-Layer. Obwohl die Spielrunde ein Attribut des Game Layers ist.

7.3 Schnittstelle: Anwendungs-Layer → Spiel-Layer

Die Schnittstelle zwischen dem Anwendungs-Layer und dem Spiel-Layer definiert, wie der Spielablauf gesteuert wird, ohne dass der Anwendungs-Layer selbst die fachliche Spiellogik enthält. Der Anwendungs-Layer übernimmt dabei eine koordinierende Rolle, während der Spiel-Layer die eigentliche Logik des Spiels umsetzt.

Im Anwendungs-Layer sind die **SpielEngine und die VergleichsEngine** die zentralen Komponenten. Die SpielEngine ist für die Steuerung des Spielablaufs verantwortlich, indem sie die Ausführung von Zügen organisiert und die Ergebnisse an die Benutzeroberfläche zurückgibt.

Der Spiel-Layer enthält die fachlichen Objekte des Spiels. Dazu gehören das Spiel selbst, das den Spielzustand verwaltet, und die Spielrunde, die einen einzelnen Rateversuch repräsentiert. Weiterhin sind hier die Klassen Code und Feedback angesiedelt, die den Geheimcode, die Rateversuche und die Bewertung der Versuche abbilden.

Der Anwendungs-Layer kommuniziert mit dem Spiel-Layer, indem er Methodenaufrufe nutzt, um Aktionen wie das Ausführen eines Zugs oder das Starten eines Spiels zu initiieren. Ein Beispiel hierfür ist der Aufruf **fuehreZugAus(Code**) durch die SpielEngine an das Spiel-Objekt. Daraufhin wird im Spiel-Layer das Feedback berechnet, etwa durch den Aufruf **berechneFeedback(Code, Geheimcode)**. Das Ergebnis wird anschließend an den Anwendungs-Layer zurückgegeben, der es an die Benutzeroberfläche weiterleitet.

Ein zentrales Designprinzip ist, dass der Game-Layer keine Kenntnis vom Anwendungs-Layer besitzt. Es gibt keine Rückreferenzen, was bedeutet, dass die fachliche Logik vollständig unabhängig von der Steuerungsebene bleibt. Dies ermöglicht eine klare Trennung der Verantwortlichkeiten und erleichtert die Wartung und Erweiterung beider Layer.

7.4 Schnittstelle: Anwendungs-Layer (Online-/Kommunikations-Layer)

Die Kommunikation zwischen Spiel und Server erfolgt durch die Schnittstelle ComPort, dabei definiert diese Schnittstelle die Eingabewerte und Ausgabewerte, um eine Kommunikation zu ermöglichen. Die Eingabewerte sind dabei Farbcodes als Rateversuche und die Ausgabewerte sind dann ein Feedback welches vom Server empfangen und im Kommunikations-Layer verarbeitet wird. Unter anderem wird diese Schnittstelle ausschließlich im Online-Modus verwendet. Sie dient außerdem der Initialisierung eines Online-Spiels. Im Online-Modus delegiert der Anwendungs-Layer die Kommunikation an den

Kommunikations-Layer.Rateversuche werden als JSON Nachrichten an den Server übermittelt, und die vom Server gelieferten Antworten werden vom Anwendungs-Layer fachlich interpretiert und in den weiteren Spielablauf integriert.

7.5 Schnittstelle: Anwendungs-Layer → Strategie/Player-Layer

Die Verbindung zwischen dem **Anwendungs-Layer** und dem Subsystem **Strategie/Player-Layer** dient dazu, die Erzeugung von Rateversuchen zu koordinieren, ohne dass der Anwendungs-Layer selbst die konkrete Logik der Algorithmen kennt. Diese Trennung ermöglicht es, verschiedene Spielstrategien flexibel einzusetzen und auszutauschen.

Der **Game-Layer** enthält die fachliche Logik des Spiels, während der **Strategie/Player-Layer** die Aufgabe hat, Rateversuche zu generieren. Dieser Layer unterstützt verschiedene Algorithmen, wie etwa **Knuth** oder **Step-by-Step**, die jeweils unterschiedliche Strategien verfolgen, um den Geheimcode zu erraten.

Die Kommunikation zwischen diesen Komponenten erfolgt über das **Interface Player**. Dieses Interface definiert zwei zentrale Methoden: **generiereVersuch()**, die einen neuen Rateversuch basierend auf den vorherigen Spielrunden erstellt, und **generiereGeheimCode()**, die den Geheimcode zu Beginn eines Spiels erzeugt. Das Interface dient als Vertrag, der sicherstellt, dass jede Implementierung eines Players, sei es ein menschlicher Spieler oder ein Computer-Player, diese Methoden bereitstellt.

7.6 Zusammenfassung der Schnittstellenprinzipien

- Jede Schicht kommuniziert nur über klar definierte Schnittstellen
- Keine direkten Zugriffe von UI auf Domain
- Keine technischen Details (HTTP, JSON) im Game
- Online-Funktionalität ist austauschbar
- Das Design bleibt erweiterbar und testbar

8. Implementierung

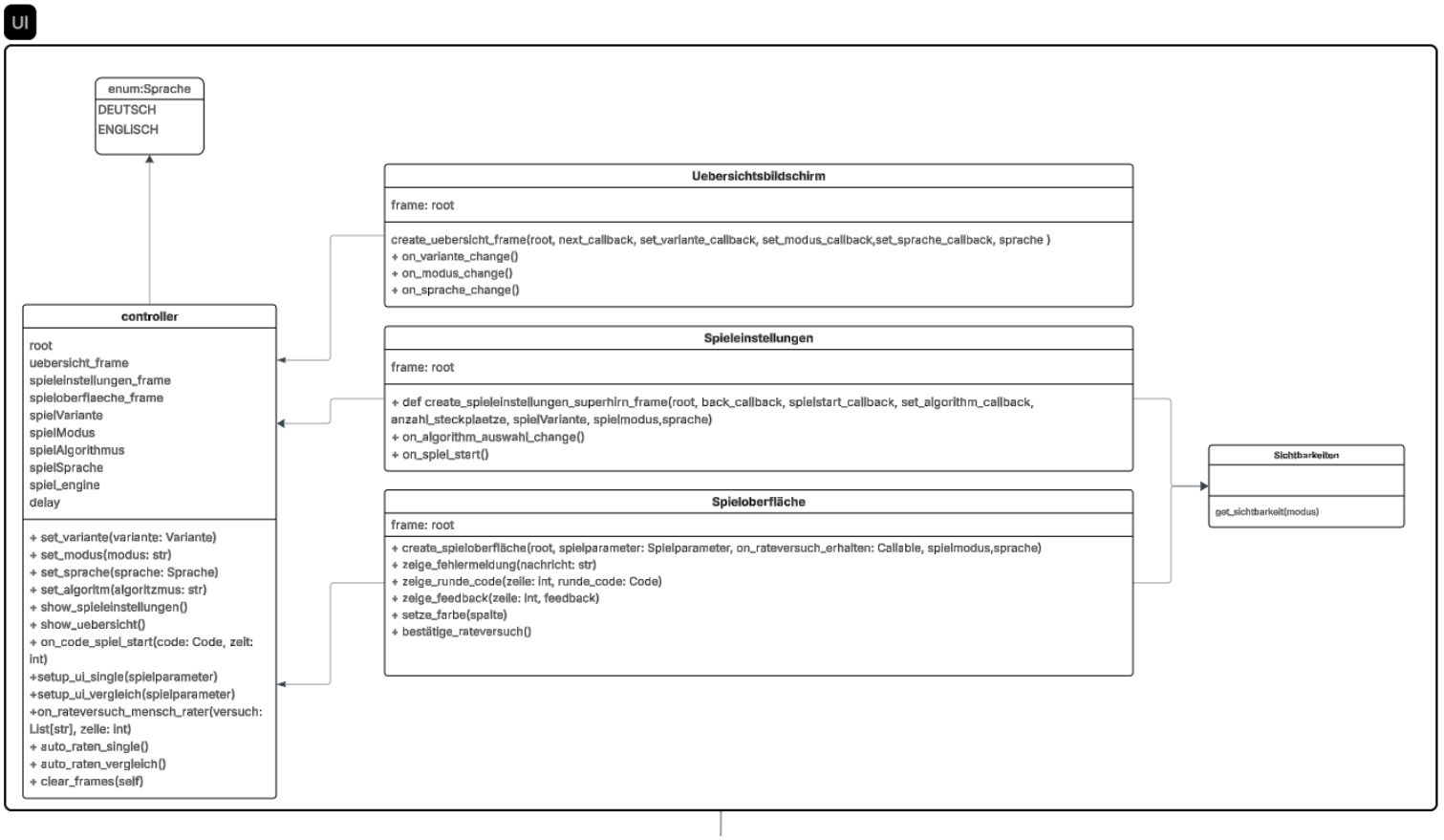
Im Allgemeinen ist festzuhalten, dass es vereinzelt Klassen gibt, die global verwendet werden, jedoch innerhalb spezifischer Packages liegen. Dazu zählen unter anderem:

- **SpielParameter**, welche die Spieleinstellungen speichern,
- das **Enum Modus**, das die einzelnen Spielmodi definiert,

- das **Enum Farbe**, welches die möglichen Farben festlegt,
- das **Enum Variante**, das die Varianten *Super* und *SuperSuperhirn* beschreibt,
- **Feedback**, welches die Anzahl der schwarzen und weißen Pins repräsentiert,
- **Code**, eine Liste von Farben, die sowohl Rateversuche als auch den Geheimcode darstellen,
- **Spielrunde**, in der ein einzelner Spielzug gespeichert wird.

Im Nachhinein betrachtet hätten diese Klassen sinnvollerweise in einem gemeinsamen „Utility“-Package gebündelt werden sollen, auf das global zugegriffen werden hätte können.

8.1 UI-Layer



Der UI-Layer übernimmt die reine Darstellung des Spiels sowie die Entgegennahme von Benutzereingaben, ohne dabei selbst Spielregeln, Logik oder Algorithmen zu implementieren. Seine einzige Aufgabe besteht darin, Benutzerinteraktionen zu erfassen und diese an den Anwendungs-Layer weiterzuleiten, während er gleichzeitig die vom System gelieferten Daten visuell aufbereitet.

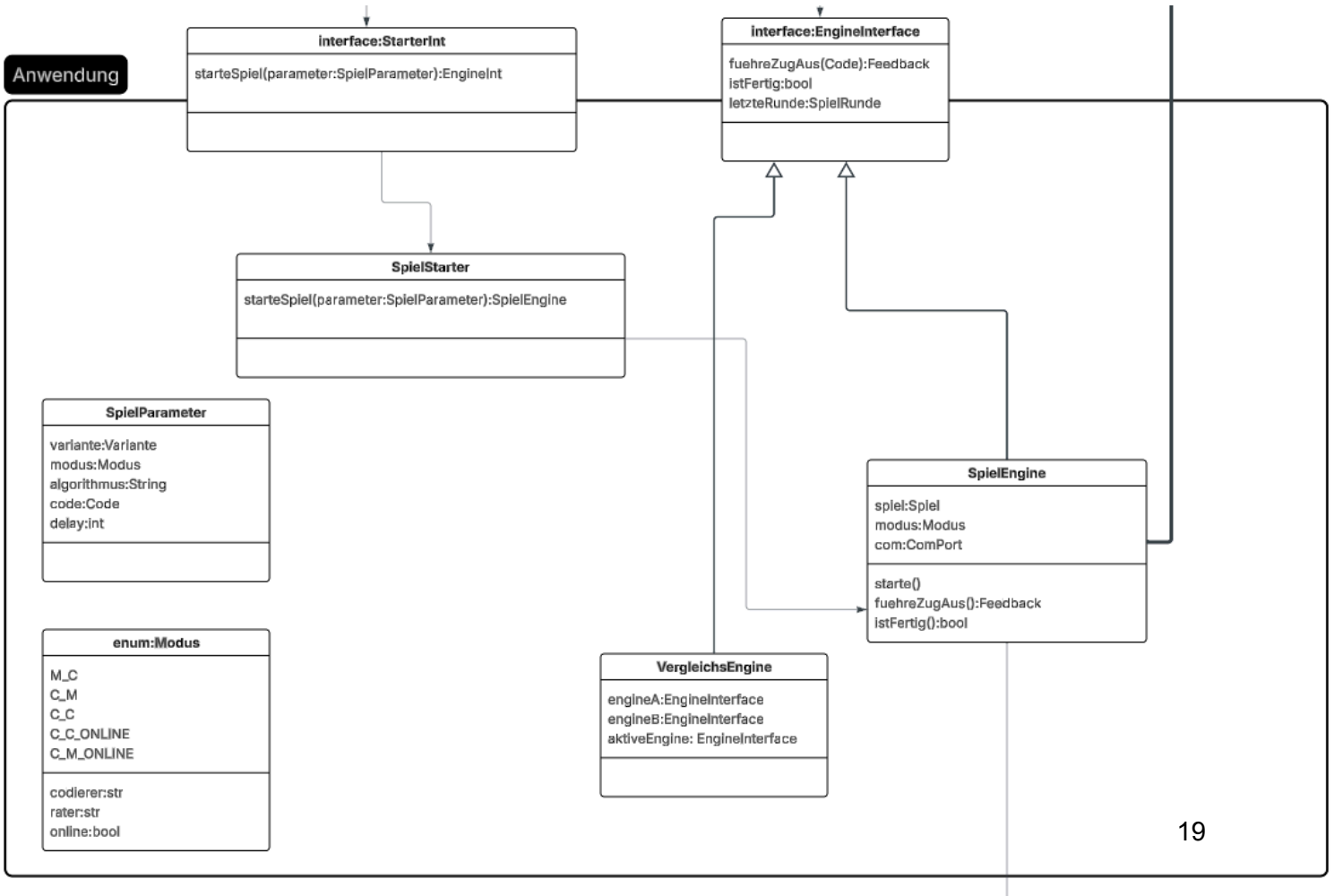
Die Benutzeroberfläche gliedert sich in drei Hauptbereiche: Zunächst präsentiert der **Übersichtsbildschirm** die grundlegenden Auswahlmöglichkeiten wie Spielvariante,

Spielmodus und Sprache, die der Nutzer vor Spielbeginn festlegen kann. Im Anschluss ermöglicht der Bereich **Spieleinstellungen** eine detaillierte Konfiguration, etwa die Auswahl von Farben, dem zu verwendenden Algorithmus für computergesteuerte Gegner oder einer optional einstellbare Verzögerung zwischen den Spielzügen. Der eigentliche Spielverlauf wird schließlich in der **Spieloberfläche** dargestellt, die den aktuellen Stand der Versuche sowie das entsprechende Feedback anzeigt.

Die Kommunikation der UI mit dem Rest des Systems erfolgt ausschließlich über den SpielStarter im Anwendungs-Layer. Dieser Mechanismus stellt sicher, dass die Benutzeroberfläche keine direkten Abhängigkeiten zur Domänenlogik aufweist. Statt komplexer Objekte werden lediglich primitive Datentypen wie Zahlen, Zeichenketten oder einfache Wahrheitswerte ausgetauscht, was die UI besonders flexibel und austauschbar macht. Durch diese klare Trennung bleibt die Benutzeroberfläche unabhängig von der inneren Funktionsweise des Spiels und kann ohne Auswirkungen auf die Spiellogik angepasst oder ersetzt werden.

Die Implementierung folgt dabei dem Prinzip der **dummen UI,** die Oberfläche empfängt lediglich Anweisungen zur Darstellung und meldet Nutzeraktionen zurück, ohne selbst Entscheidungen zu treffen. Diese strikte Trennung ermöglicht es, die UI später durch alternative Oberflächen, etwa eine Konsolenversion oder eine grafische Benutzeroberfläche zu ersetzen, ohne dass Änderungen an der eigentlichen Spiellogik notwendig wären.

8.2 Anwendungs-Layer



Der Anwendungs-Layer bildet die Steuerungsebene des Systems und vermittelt zwischen der Benutzeroberfläche und der fachlichen Domäne. Seine primäre Aufgabe besteht darin, den Spielablauf zu koordinieren, ohne selbst fachliche Logik zu enthalten. Er entscheidet, welche Komponenten in welcher Reihenfolge aktiv werden, leitet Benutzeraktionen an die Domäne weiter und sorgt dafür, dass die Ergebnisse korrekt an die UI zurückgemeldet werden.

Verantwortlichkeiten

Dieser Layer ist zuständig für:

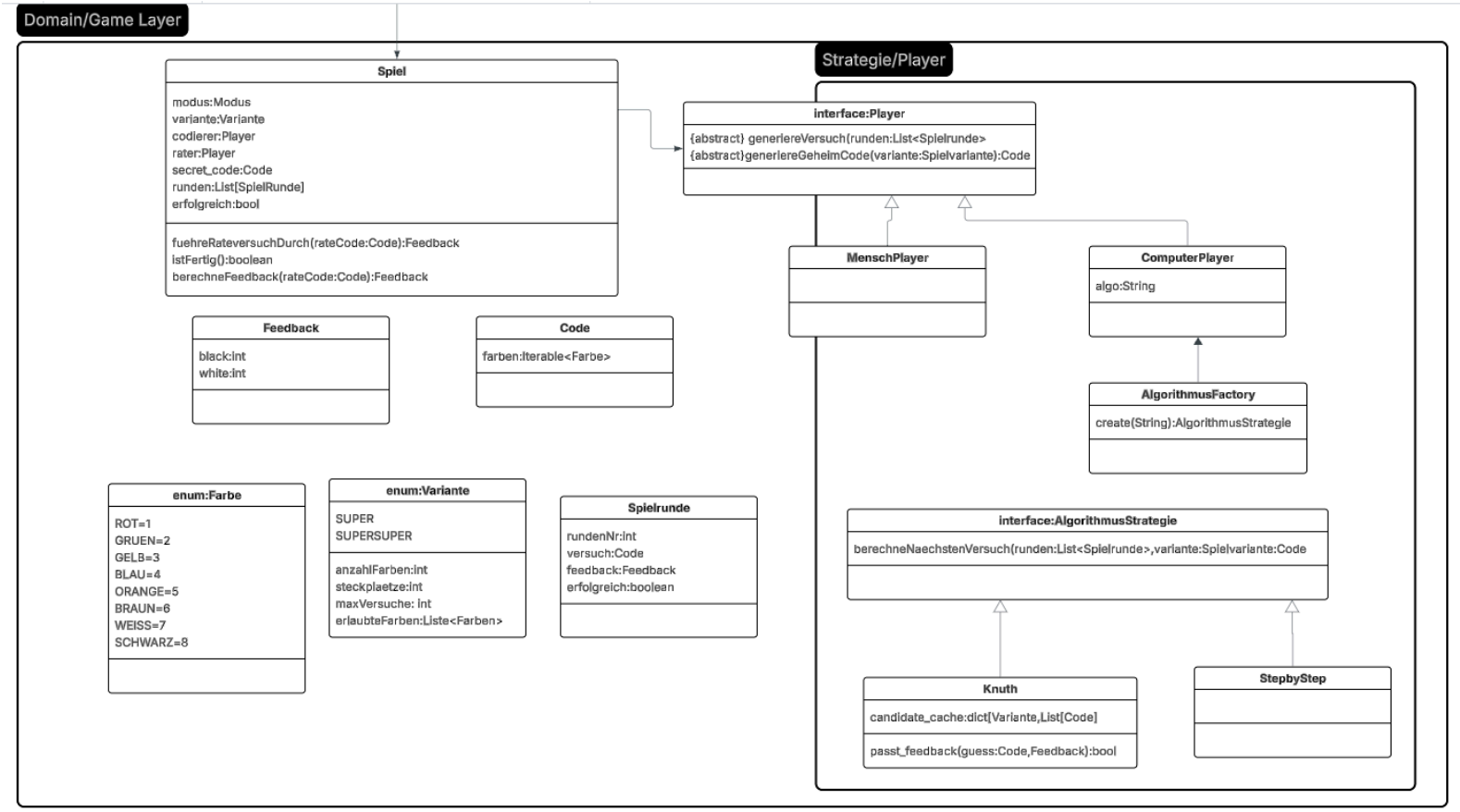
- Initialisierung und Konfiguration des Spiels basierend auf den vom Nutzer gewählten Einstellungen.
- Auswahl des Spielmodus und die Zusammenstellung der benötigten Komponenten.
- Orchestrierung des Spielablaufs, d. h. die Steuerung der Interaktion zwischen Spielern, Algorithmen und der Spiellogik.
- Weiterleitung von Eingaben aus der UI an die Domäne und Rückgabe der Ergebnisse in einem für die Darstellung geeigneten Format.
- Verwaltung von Spielsitzungen, einschließlich des Starts, der Unterbrechung und des Neustarts von Runden.

Struktur und Schnittstellen

Der Layer ist so konzipiert, dass er keine fachliche Logik enthält, sondern lediglich Abläufe steuert. Dazu nutzt er:

- Eine zentrale Steuerkomponente, die als Vermittler zwischen UI und Domäne fungiert. Sie empfängt Nutzeraktionen (z. B. Spielstart oder Parameteränderungen), leitet diese an die Domäne weiter und gibt die Ergebnisse strukturiert zurück.
- Modus-spezifische Ablaufsteuerungen, die je nach Spielvariante (z. B. Einzelspiel vs. Algorithmenvergleich) unterschiedliche Komponenten kombinieren. Dabei bleibt die fachliche Logik unverändert – nur die Reihenfolge der Aufrufe und die Kombination der Module passen sich an.
- Eine einheitliche Parameterstruktur, die alle konfigurierbaren Einstellungen (z. B. Spielvariante, Schwierigkeitsgrad, technische Optionen) bündelt und an die Domäne sowie die UI weitergibt. Diese Struktur stellt sicher, dass alle Komponenten mit denselben Grundlagen arbeiten, ohne direkte Abhängigkeiten voneinander zu haben.

8.3 Game-Layer mit Strategie/Player Layer



Game Layer

Der Game-Layer bildet den fachlichen Kern des Systems und enthält die zentralen Gameobjekte sowie die Spielregeln. Er ist vollständig unabhängig von technischen Aspekten wie Benutzeroberfläche oder Netzwerkkommunikation.

Zentrale fachliche Datenstrukturen sind **Code** und **Feedback**:

- Ein **Code** repräsentiert eine Folge von Farben und wird sowohl für den Geheimcode als auch für Rateversuche verwendet. Die Länge des Codes ergibt sich aus der gewählten Spielvariante.
- Ein **Feedback** beschreibt das Ergebnis eines Rateversuchs anhand der Anzahl korrekt platzierter Farben (schwarz) sowie korrekt erkannter Farben an falscher Position (weiß).

Die möglichen Farben und Varianten werden über fachliche Enumerationen modelliert:

- **Farbe** beschreibt die im Spiel verfügbaren Farben.

- **Variante** kapselt die wesentlichen Parameter der Spielkonfiguration (z. B. Anzahl Steckplätze, Anzahl Farben, maximale Versuche, erlaubte Farben).

Ein einzelner Spielzug wird als **SpielRunde** gespeichert. Eine Spielrunde enthält den Rateversuch (Code), das dazugehörige Feedback sowie die Rundennummer. Dadurch entsteht eine nachvollziehbare Historie der Spielentwicklung.

Die zentrale Klasse der Spiellogik ist **Game**. Sie verwaltet den vollständigen Spielzustand einer Partie, insbesondere:

- die aktuelle Spielkonfiguration (Variante/Parameter),
- die beteiligten Rollen (Rater und Codierer),
- den Geheimcode,
- sowie die Historie der bereits durchgeführten Runden List[SpielRunde].

Beim Ausführen eines Spielzugs verarbeitet **Game** einen neuen Rateversuch, lässt dazu das fachliche Feedback berechnen und speichert das Ergebnis als neue Spielrunde. Zusätzlich stellt **Game** eine Abbruchprüfung bereit, um zu entscheiden, ob das Spiel beendet ist (z. B. durch korrektes Erraten des Codes oder durch Erreichen der maximalen Versuchszahl).

Strategie / Player Layer

Der Strategie- und Player-Layer kapselt die verschiedenen Spielerrollen sowie die austauschbaren Ratealgorithmen des Systems.
Über das Interface Player wird ein einheitlicher Zugriff auf das Erzeugen von Rateversuchen sowie, falls erforderlich, das Festlegen eines Geheimcodes ermöglicht.

Es existieren zwei konkrete Implementierungen von Spielern:

- **MenschPlayer**, der seine Eingaben über die Benutzeroberfläche erhält,
- **ComputerPlayer**, der seine Entscheidungen vollständig an einen Ratealgorithmus delegiert.

Der **ComputerPlayer** besitzt dabei eine Referenz auf eine AlgorithmusStrategie, die das eigentliche Entscheidungsverhalten kapselt.
Dadurch können unterschiedliche Algorithmen verwendet werden, ohne die Spiellogik oder den Player selbst zu verändern.

Alle Algorithmen implementieren das Interface AlgorithmusStrategie mit der Methode berechneNaechstenVersuch(List<SpielRunde>, Variante), welche auf Basis des bisherigen Spielverlaufs einen neuen Rateversuch erzeugt.

Knuth

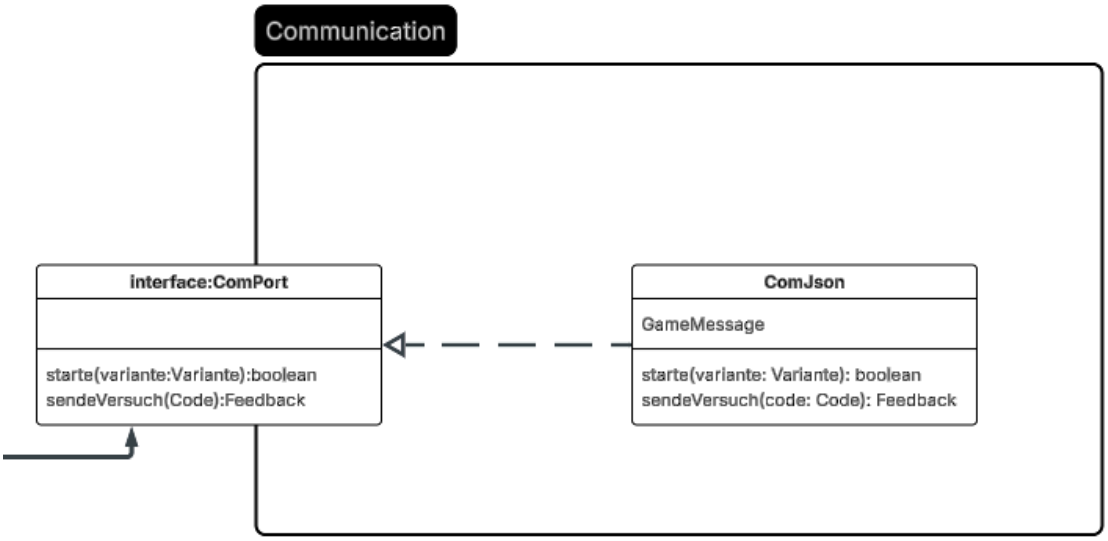
Der Knuth-Algorithmus verfolgt einen systematischen Ansatz zur Lösung des Mastermind-Problems.

Er reduziert nach jedem Feedback die Menge der möglichen Geheimcodes und wählt den nächsten Versuch so, dass die Anzahl der verbleibenden Möglichkeiten möglichst stark minimiert wird.
Dadurch erreicht der Algorithmus eine sehr effiziente Lösungsstrategie mit einer geringen durchschnittlichen Anzahl an Versuchen.

StepByStep

Der StepByStep-Algorithmus geht schrittweise und weniger komplex vor.
Er nutzt das Feedback der vorherigen Spielrunden, um den nächsten Versuch inkrementell anzupassen, ohne den gesamten Suchraum systematisch zu minimieren.
Dieser Algorithmus ist einfacher aufgebaut und dient insbesondere dem Vergleich mit komplexeren Strategien wie dem Knuth-Algorithmus.

8.4 Com Layer



Der Kommunikations-Layer kapselt die Online-Kommunikation mit einem externen Superhirn-Server. Ziel ist es, die technische Übertragung (HTTP/JSON) vollständig von Anwendungs- und Domänenlogik zu trennen, sodass der Domain-/Game-Layer unverändert und technikneutral bleibt.

Die Kommunikation wird über eine Schnittstelle (**ComPort**) abstrahiert. Diese Schnittstelle definiert fachlich, welche Operationen im Online-Modus benötigt werden, beispielsweise:

- Initialisierung eines Online-Spiels (Handshake / Start),
- Übermittlung eines Rateversuchs,

- Empfang und Rückgabe des zugehörigen Feedbacks.

Eine konkrete Implementierung dieser Schnittstelle übernimmt die Serialisierung und Deserialisierung der Nachrichten (Move/Feedback) und stellt dem Anwendungs-Layer ausschließlich fachlich interpretierbare Ergebnisse zur Verfügung. Der Anwendungs-Layer kann dadurch im Online-Modus die lokale Feedback-Berechnung ersetzen, ohne dass Änderungen an der eigentlichen Spiellogik erforderlich sind.

Der Ablauf im Online-Modus ist fachlich wie folgt:

1. Der Anwendungs-Layer initialisiert ein neues Online-Spiel über den Kommunikations-Layer (Server vergibt eine gameid).
2. Jeder Rateversuch wird als standardisierte Nachricht an den Server übertragen.
3. Der Server liefert das Feedback zurück, das anschließend in die lokale Spielhistorie integriert wird.

9. Qualitätssicherung

9.1 Beschreibung des gewählten QS-Prozesses

Die Qualitätssicherung im Projekt Superhirn wurde begleitend über alle Phasen der Entwicklung hinweg durchgeführt und war eng mit Analyse, Entwurf und Implementierung verzahnt. Das Ziel war sowohl die fachliche Korrektheit des Systems als auch die Einhaltung der nicht-funktionalen Anforderungen sicherzustellen.

Bereits in der Analyse- und Entwurfsphase wurde großer Wert auf eine klare Strukturierung der Anforderungen sowie auf eine saubere Architektur gelegt. Durch die konsequente Trennung der Schichten (UI, Anwendung, Game, Kommunikation) konnten fachliche Fehler frühzeitig identifiziert und strukturelle Probleme vermieden werden.

Während der Implementierung kamen mehrere konkrete Qualitätssicherungsmaßnahmen zum Einsatz:

Strukturkontrolle:
Durch die klare Schichtentrennung und die Begrenzung der Abhängigkeiten zwischen den Komponenten wurde eine geringe Kopplung erreicht. Dies verbesserte die Wartbarkeit und Erweiterbarkeit des Systems.

Kommentare:

Sofern sich die Funktionalität einer Klasse nicht unmittelbar erschloss, wurde der Code durch kurze und prägnante Kommentare erläutert, um dem Leser ein schnelles Verständnis zu ermöglichen.

Plattformunabhängigkeit:
Das System wurde so entworfen, dass es unabhängig von einer konkreten

Benutzeroberfläche oder Laufzeitumgebung funktioniert. Dadurch konnte die Spiellogik ohne Anpassungen in unterschiedlichen Kontexten verwendet und getestet werden.

9.2 Teststrategie & Abdeckung:

Im Allgemeinen wurde überwiegend Bottom-up-Testing angewendet. Dabei wurden insbesondere die Game- und Com-Layer isoliert und unabhängig voneinander getestet. Integrationstests in Kombination mit dem Anwendungslayer wurden jedoch nicht durchgeführt.

Game Layer Tests

Die Qualitätssicherung des Game Layers konzentriert sich auf die isolierte Prüfung der Spiellogik, unabhängig von UI oder externen Systemen. Durch diese Trennung können die Kernfunktionen, insbesondere die Feedback-Berechnung und die Spielabbruch Bedingungen, präzise und ohne Störeinflüsse getestet werden.

Als Testframework kommt ein Pytest zum Einsatz, da es durch klare Assertions eine zuverlässige Überprüfung ermöglicht. Die Tests basieren auf vordefinierten Geheimcodes, um reproduzierbare Ergebnisse zu gewährleisten und Zufallsfehler auszuschließen. Dabei werden verschiedene Szenarien abgedeckt:

- Vollständige Übereinstimmungen (korrekter Rateversuch = 4 schwarze Pins)
- Partielle Treffer (Kombinationen aus schwarzen und weißen Pins, z. B. *2 schwarz, 1 weiß* wie im Testfall test gemischtes feedback) (siehe Screenshot; unterhalb)
- Vollständige Fehlversuche (keine Übereinstimmung)

```
def test_gemischtes_feedback(game):  Huber.Johannes
    # Erwartung: 2 schwarz (ORANGE an Position 0), 1 weiß (ROT und GELB falsche Position)
    rate = Code([Farbe.ORANGE, Farbe.ROT, Farbe.GELB, Farbe.BLAU])

    feedback = game.fuehreRateversuchDurch(rate)

    assert feedback.schwarz == 2
    assert feedback.weiss == 1
```

Die Teststrategie folgt dem Black-Box-Ansatz, bei dem ausschließlich das Verhalten des Systems und nicht seine Implementierung geprüft wird. Dies stellt sicher, dass die Spezifikationen der Spielregeln (z. B. korrekte Feedback-Rückgabe oder Gewinnbedingungen) exakt erfüllt werden.

Ein weiterer Schwerpunkt liegt auf der Validierung der Spielabbruch Bedingungen:

- Gewinn-Erkennung bei korrektem Rateversuch
- Verlust-Erkennung nach Ausschöpfen der maximalen Versuche

UI Layer Tests

Die Tests im UI Layer konzentrieren sich auf die funktionale Überprüfung der Benutzeroberfläche ohne tiefe Logikprüfung.

Es wird sichergestellt, dass der Wechsel zwischen den verschiedenen Ansichten reibungslos funktioniert und die richtigen UI-Elemente je nach Spielmodus ein- oder ausgeblendet werden. So erscheint etwa im Modus *Computer vs. Mensch* kein Eingabefeld für den Geheimcode, da dieser automatisch generiert wird.

Ein weiterer Fokus liegt auf der korrekten Weiterleitung von Benutzeraktionen wie Rateversuchen oder Spielstarts über Callbacks an den Game Layer. Zudem wird geprüft, ob Feedback wie schwarze und weiße Pins korrekt angezeigt wird und die Oberfläche konsistent auf Interaktionen reagiert.

Die Tests erfolgen manuell und durch simulierte Interaktionen, um eine nahtlose Bedienbarkeit und visuelle Konsistenz in allen Spielzuständen zu garantieren.

Com Layer Tests

Die Testabdeckung vom Kommunikationslayer bzw. der konkreten Implementierung erfolgte mit der Hilfe von der Response Bibliothek, diese wurden mit Pytest Tests geschrieben. Die genannte Bibliothek dient vor allem zum Mocken von Server Antworten. Dabei wurde jeweils ein Testobjekt erstellt und die Methoden aus dem Kommunikations-Layer wurden in bestimmten Reihenfolgen und übergebenen Parametern (je nach Testfall) aufgerufen. Folgendermaßen wurden dann entsprechend dem Testfall bestimmte Server Antworten gemockt, sodass die Funktionalität des Kommunikations-Layers geprüft werden kann.

Systemtests

Am Ende der Implementierungsphase wurden die Anforderungen strukturiert und mit vollständigen Systemtests überprüft. So konnte sichergestellt werden, dass alle Modi voll lauffähig sind. Hierbei wurden für die Tests der Online-Modi die Responses des Servers gemockt. Die Durchführung dieser Tests war sehr sinnvoll im Vorfeld der praktischen Präsentation.

10. Fazit

10.1 Beurteilung des eigenen Ergebnisses

Das Projekt Superhirn hat aus Sicht des Teams die gesetzten Ziele in gutem Maße erfüllt. Insbesondere die klare Trennung zwischen fachlicher Spiellogik, Anwendungssteuerung, Benutzeroberfläche und technischer Kommunikation ermöglichte ein strukturiertes und nachvollziehbares Systemdesign.

Die in der Analysephase definierten funktionalen und nicht-funktionalen Anforderungen konnten im Entwurf konsequent berücksichtigt werden.
Durch das modulare Design ist das System erweiterbar, beispielsweise durch neue Spielvarianten, zusätzliche Algorithmen oder alternative Benutzeroberflächen, ohne dass grundlegende Änderungen an der bestehenden Struktur notwendig sind.

Auch der Umgang mit nachträglich hinzugekommenen Anforderungen, insbesondere der Online-Anbindung, bestätigte die Qualität des gewählten Designs.
Die neue Funktionalität konnte als optionale Erweiterung integriert werden, ohne dass das bestehende Domänenmodell oder die Spiellogik verändert wurde.

Rückblickend würde das Team bei einer erneuten Durchführung des Projekts noch früher automatisierte Tests einführen, um fachliche Fehler bereits in frühen Entwicklungsphasen schneller zu erkennen. Darüber hinaus könnten zusätzliche Metriken zur Bewertung von Komplexität und Kopplung eingesetzt werden, um Designentscheidungen noch systematischer abzusichern.

Viele der aufgetretenen Probleme resultierten aus einem unzureichenden Zeitmanagement während der Implementierungsphase. Zwar wurden die zuvor beschriebenen Anforderungen größtenteils vollständig umgesetzt, jedoch sind durch den entstandenen Zeitdruck einige essentielle Aspekte der Implementierung in den Hintergrund gerückt. So fehlt beispielsweise die Möglichkeit, nach einem abgeschlossenen Spiel auf den Home-Screen zurückzukehren, wodurch das Programm nach jedem Spiel neu gestartet werden muss. Ebenso existiert keine Option, im Client eine IP-Adresse einzugeben oder zu ändern, da diese direkt im Code hart kodiert ist.

Auch die Robustheit des Systems konnte nicht vollständig getestet werden, insbesondere im Hinblick auf den Online-Modus. Fehlerhafte Antworten oder Netzwerkprobleme wurden hierbei nicht abgefangen. Zwar wurden die einzelnen Komponenten größtenteils isoliert getestet, jedoch fanden keine Integrationstests statt.

Im Hinblick auf den Abgabetermin wurden im Zeitdruck zudem vereinzelt kleinere Anpassungen in der Implementierung bezüglich der Umsetzung des Objektmodells vorgenommen.

Insgesamt stellt das Projekt eine solide Umsetzung der Anforderungen dar und bietet eine stabile Grundlage für Erweiterungen oder alternative Implementierungen.