Hochschule für Technik und Wirtschaft

Software Engineering 2

Super Superhirn

GitLab: https://gitlab.rz.htw-berlin.de/s0572802/wise23-24_superhirn_11

Alexander Yurovskyy, Matrikelnummer: 572782

Can Wrobel, Matrikelnummer: 585031

Carl Robert Weiß, Matrikelnummer: 572802

Linda Zubenin, Matrikelnummer: 584625

Rico Flemming, Matrikelnummer: 584322

Dozent: Prof. Dr. Stephan Salinger

02.01.2024

Inhaltsverzeichnis

Einleitung

Die Aufgabe besteht darin, das Spiel „Super Superhirn“ zu implementieren, bei dem das Programm als „Codierer“ sowie auch als „Rater“ agieren kann. Der Spieler setzt am Anfang die Anzahl der Farben und Steckplätze, die zur Verfügung stehen sollen. Dabei gibt es 4-5 Steckplätze für den Code und 2-8 Farben. Danach sucht sich der Spieler seine Rolle als Codierer oder Rater aus. Der Codierer legt zu Beginn einen geordneten Farbcode aus den wählbaren Farben fest, wobei sich Farben wiederholen dürfen. Der Rater versucht, diesen Code herauszufinden. Bei jedem Zug des Raters wird Feedback vom Codierer gegeben: Ein weißer Stein steht für die richtige Farbe, aber falsche Position, und ein schwarzer Stein steht für die richtige Farbe an der richtigen Position. Der Rater hat maximal 10 Rateversuche, um den Code zu erraten.

Bei Super Superhirn gibt es ebenfalls die Möglichkeit, das Spiel über das Netzwerk laufen zu lassen. Wenn der Spieler als Rater spielt, hat er die Auswahl, ob die Rolle des Codierers vom laufenden Programm oder über das Internet genutzt werden soll. Ebenfalls kann der Spieler auswählen, ob der NPC (Non-Player Character) als Rater gegen den Server spielt.

Beschreibung des gewählten SW-Prozesses

Die Ausarbeitung der Präsentationen von Zwischenständen scheint dem Ansatz des Wasserfallmodells zu ähneln. Dennoch liegt der Fokus des gewählten Software-Entwicklungsprozesses der Implementierung von „Super Superhirn“ auf einem iterativen Vorgehen und agilen Methoden. Mit dem iterativen Ansatz kann man sich schrittweise an das Projekt nähern. Dadurch wird eine kontinuierliche Verbesserung und Anpassung ermöglicht. Die gesenkte Komplexität der einzelnen Arbeitsschritte erlaubt es, einfacher mit unklaren oder veränderten Anforderungen umzugehen. Iteratives Vorgehen verlangt auch engere Kommunikation aller Gruppenmitglieder. Die Entwicklung basiert konkret auf einem Prototypmodell und dem evolutionären, iterativen Modell (vgl. Salinger 2021a, 30ff).

In der Analyse- und Implementierungsphase wurden Prototypen erstellt. Diese dienten dazu, die kritischen Anforderungen besser zu verstehen. Des Weiteren ermöglichte das evolutionäre Modell das schrittweise Bauen des Gesamtsystems, wobei bei Anforderungsänderungen neue Teile hinzugefügt, aber auch wenn nötig existierende verändert wurden (vgl. ebd.). Ein

Beispiel für die Verwendung wäre die Anforderungsveränderung Nummer 1, die dem Spieler erlaubt hat, die Anzahl der Farben und des Codes auszuwählen.

Insbesondere wurde Scrum in Kombination von Linear, einem Projektmanagement-Tool, eingesetzt (Linear, https://linear.app/). Scrum ist dabei ein leichtgewichtiges Framework, das iterative und inkrementelle Ansätze zur Optimierung der Vorhersagbarkeit und Risikokontrolle verwendet. Es fördert selbstorganisierte Teams, die eigenverantwortlich arbeiten, und bietet Transparenz, indem die entstehende Arbeit auch für diejenigen sichtbar ist, die sie empfangen (vgl. Schwaber et al. 2011, 4ff). Sprints und Sprint-Retrospektiven ermöglichen dem Team, den Entwicklungsprozess zu analysieren und kontinuierlich zu verbessern (vgl. ebd., 12ff). Somit erleichtert die Arbeit mit Scrum sich an Anforderungsänderungen anzupassen. Während der Arbeit an „Super Superhirn“ wurde nach jedem Sprint ein Review durchgeführt, um die Arbeit zu überprüfen und zu entscheiden, ob die Ausarbeitungen den aktuellen Anforderungen entsprechen. Mit Linear wurden die Bearbeitungsschritte in ein Backlog eingefügt. Jede Aufgabe, abhängig vom Ausführungsgrad, wurde in „Todo“, „In Progress“, „In Review“ und „Done“ eingeteilt.

Analyse

Beschreibung des gewählten Analyseprozesses

Für die Erhebung von Anforderungen wurden sowohl klassische als auch representation-based approaches verwendet. Zu den klassischen Methoden gehören Introspektion, Sichtung, Interviews, Prototyping und Brainstorming im Team. Bei der Methode der Introspektion reflektieren die Teammitglieder im Vorfeld ihre eigenen Gedanken über potenzielle Anforderungen. Allerdings müssen die gefundenen Anforderungen sorgfältig überprüft werden, da sie eventuell mit den Vorstellungen des Stakeholders nicht übereinstimmen (vgl. Salinger 2021b, 29ff).

Um eine Basis der funktionalen und nichtfunktionalen Anforderungen zu schaffen, wurde die Sichtung der PowerPoint: Software Engineering 2 Übung 4 (vgl. Salinger 2023, 7ff) durchgeführt. Diese PowerPoint-Folien enthielten relevante Informationen und Erkenntnisse zum Projekt und dienten als Grundlage. Ein Prototyp wurde erstellt, um ein greifbares,

visuelles Modell des Produktes herzustellen, das beim Sammeln von Feedback helfen sollte. Der Prototyp zeigt den Code und eine mögliche farbliche Visualisierung.

Danach wurde ein Interview mit dem relevanten Stakeholder durchgeführt, um weitere Informationen zu den Erwartungen und Anforderungen zu gewinnen. Durch den Prototypen konnte der Stakeholder die Ideen der Gruppe bewerten. Das Gespräch ermöglicht ein tieferes Verständnis des Projekts. Brainstorming sollte dem Team einen kreativen Raum schaffen, vielfältige Ideen zu sammeln und Verbesserungsvorschläge zu äußern.

Use Cases und das Sammeln von Szenarien gehören zu dem Szenario-basierten Verfahren. Durch Gespräche mit dem Stakeholder und Gruppenmitgliedern wurden erste Szenarien aufgezeichnet. Diese Szenarien dienten als Basis für das Schreiben von Anwendungsfällen. Die Use Cases beschreiben eine Abfolge von Aktionen. Sie liefern ein beobachtbarer Wertergebnis für einen Akteur (vgl. Salinger 2021b, 40ff).

Des Weiteren wurden Use Case Diagramme erstellt, die die Interaktionen zwischen den Akteuren und dem System auf einer höheren Ebene visualisieren. Das Objektdiagramm dient dazu, einen Überblick der statischen Struktur des Systems zu erhalten. Die einzelnen Entitäten mit ihren Attributen und Beziehungen wurden visuell dargestellt, um die Wechselwirkungen zu erkennen (vgl. Salinger 2021c, 17ff). Außer den statischen Modellierungen sollten auch dynamische Abläufe und Interaktionen zwischen den Objekten und Akteuren dargestellt werden. Die Sequenzdiagramme haben dabei geholfen, bestimmte Szenarien zu visualisieren und die zeitliche Abfolge zu verstehen.

Die Kombination dieser Techniken erlaubt eine umfassende Erfassung der Anforderungen, deren Darstellung im nächsten Abschnitt erfolgt.

Beschreibung der Analyseergebnisse

Hier sind die funktionalen und nichtfunktionalen Anforderungen im Detail:

Funktionale Anforderungen: Das Spiel Super Superhirn besteht aus einem Codierer und einem Rater. Sowohl das Programm als auch der Spieler können diese Rollen übernehmen. Der Spieler wählt seine Rolle und den Modus über das Terminal nach dem Start des Programms aus. Im Terminal können weitere Kommandos wie „quit“, „reset“ und „help“

verwendet werden. „quit“ beendet das Spiel, „reset“ lässt das Spiel von neuem starten und „help“ gibt die Kommandos aus.

Wählt der Spieler den Rater aus, kann der Spieler danach aussuchen, ob er das Spiel lokal oder im Netzwerk spielt. Wird das Netzwerk ausgewählt, hat der Spieler die Möglichkeit den NPC Rater (der Rater vom Programm) auszuwählen, der gegen den Codierer im Netzwerk spielt.

Der Spieler legt ebenfalls fest, ob die Codelänge 4 oder 5 beträgt und wie viele Farben verwendet werden. Der Spieler kann zwischen min. 2 und max. 8 Farben wählen. Die Farben werden dabei als Zahlen dargestellt. Die Farbkodierung lautet: 1=Rot, 2=Grün, 3=Gelb, 4=Blau, 5=Orange, 6=Braun, 7=Weiß (Code sowie Bewertung), 8=Schwarz (Code sowie Bewertung). Das Spiel hat nur einen Schwierigkeitsgrad und der Code muss ohne Lücken sein.

Der Codierer legt zu Beginn den verdeckten Farbcode fest, der mehrmals dieselbe Farbe enthalten kann. Der Rater versucht danach, den Code herauszufinden, indem er einen Farbcode setzt. Dieser Farbcode erhält ein Feedback vom Codierer. Weiß bedeutet, dass es die richtige Farbe war, aber falsche Position. Schwarz bedeutet, dass die Farbe als auch Position richtig ist. Das Spielfeld erlaubt maximal 10 Rateversuche. Hat der Rater den Code erraten, erhält er nur schwarze Pins und gewinnt. Wurden aber alle Rateversuche verbraucht, ohne den Code zu erraten, gibt es eine Anzeige, dass der Rater verloren hat. In beiden Fällen wird der vorgegebene Code angezeigt. Neustart der Runde ist jederzeit möglich.

Nichtfunktionale Anforderungen: Das Programm soll nur in deutscher Sprache verfügbar sein. Alle Spielmechaniken müssen präzise und korrekt von der Software ausgeführt werden. Ebenfalls muss die Software leicht modifizierbar und wartbar sein. Insbesondere stehen Erweiterbarkeit, Stabilität und Modifizierbarkeit im Vordergrund. Die Software muss robust und zuverlässig sein. Sie soll eine Fehlertoleranz gegen falsche Eingaben des Spielers aufweisen. Das Programm soll auf verschiedenen Betriebssystemen wie Mac, Windows und Ubuntu lauffähig sein.

Im nächsten Abschnitt werden die Use Cases und die Diagramme präsentiert:

Spieler - Use Case: Spieler wählt seine Rolle aus

Primärakteur	Spieler
Anwendungsbereich	Superhirn
Niveau	Anwenderziel
Stakeholder und Interessen	Spieler: Möchte seine Rolle auswählen Auftraggeber (Prof. Dr. Salinger): Benötigt eine funktionierende Auswahl für die Rolle
Vorbedingung	Spieler hat das Spiel auf seinem PC, auf dem Windows/Mac oder Ubuntu läuft. Python neueste Version ist installiert. Spieler kennt die Regeln von Superhirn. Benutzer hat Superhirn gestartet.
Mindestzusicherung	Spiel gibt bei Fehlern Fehlermeldung aus.
Zusicherung im Erfolgsfall	Spieler hat seine Rolle als Codierer oder Rater gewählt
Haupterfolgsszenario	1. Spieler wählt als Spielmodus Rater oder Codierer aus.
Erweiterungen	1a. Spieler wählt falschen Spielmodus. Spieler gibt Neustart Kommando(reset) ein, um wieder in der Modusauswahl zu landen. 1b. Spieler betätigt die falsche Eingabe. Fehlermeldung, die darauf hinweist, wie der Spieler seine Rolle auswählt.

Rater - Use Case: Spieler verliert Superhirn

Primärakteur	Spieler
Anwendungsbereich	Superhirn
Niveau	Anwenderziel
Stakeholder und Interessen	Spieler: Möchte als Rater Superhirn spielen Auftraggeber (Prof. Dr. Salinger): Benötigt ein funktionierendes Spiel
Vorbedingung	Spieler hat das Spiel auf seinem PC, auf dem Windows/Mac oder Ubuntu läuft. Der PC hat Internetzugriff. Python neueste Version ist installiert. Spieler kennt die Regeln von Superhirn. Benutzer hat Superhirn gestartet.

Mindestzusicherung	Spiel gibt bei Fehlern Fehlermeldung aus.
Zusicherung im Erfolgsfall	Spieler konnte als Rater Superhirn erfolgreich spielen.
Haupterfolgsszenario	1. Spieler wählt als Spielmodus Rater aus. 2. Spieler wählt den lokalen Modus aus. 3. Spieler wählt den Benutzernamen, die Codelänge 5 und Farbanzahl 8. 4. Spieler wartet darauf, dass der Codierer einen Code vorgegeben hat. 5. Spieler rät das erste Mal, indem er den Code als Zeichenfolge von 5 Zahlen eingibt. 6. Spieler wartet auf Feedback des Codierers. 7. Spieler rät anhand des Feedbacks einen neuen Code, falls er nicht den richtigen erraten hat. Dies wiederholt sich bis zum 10. Raten des Spielers, falls er den richtigen Code nicht erraten hat. 8. Nach dem 10. falschen Rateversuch gewinnt der Codierer, und der Spieler verliert. Es wird “Verloren” auf dem Terminal ausgegeben. 9. Der vorgegebene Code vom Codierer wird gezeigt.
Erweiterungen	1a. Spieler wählt falschen Spielmodus. Spieler gibt Neustart Kommando(reset) ein, um wieder in der Modusauswahl zu landen. 2a. siehe 1a 3a. siehe 1a 5a. Spieler gibt die falsche Anzahl von Zahlen ein. Fehlermeldung, die nochmal darauf hinweist, dass der Code 5 Zahlen lang sein muss. 5b. Spieler gibt andere Zeichen als Zahlen ein. Fehlermeldung, die nochmal darauf hinweist, dass der Code 5 Zahlen lang sein muss. 7a. siehe 5a und 5b 8a.Spieler gibt erneut Eingaben ein. Fehlermeldung, die darauf hinweist, dass die Runde vorbei ist und neu gestartet wird.

Rater - Use Case: Spieler gewinnt

Primärakteur	Spieler
Anwendungsbereich	Superhirn
Niveau	Anwenderziel
Stakeholder und Interessen	Spieler: Möchte als Rater Superhirn spielen Auftraggeber (Prof. Dr. Salinger): Benötigt ein

	funktionierendes Spiel
Vorbedingung	Spieler hat das Spiel auf seinem PC, auf dem Windows/Mac oder Ubuntu läuft. Der PC hat Internetzugriff. Python neueste Version ist installiert. Spieler kennt die Regeln von Superhirn. Benutzer hat Superhirn gestartet.
Mindestzusicherung	Spiel gibt bei Fehlern Fehlermeldung aus.
Zusicherung im Erfolgsfall	Spieler konnte als Rater Superhirn erfolgreich spielen.
Haupterfolgsszenario	1. Spieler wählt als Spielmodus Rater aus. 2. Spieler wählt den lokalen oder Netzwerk Modus aus. 3. Spieler wählt den Benutzernamen, die Codelänge 5 und Farbanzahl 8. 4. Spieler wartet darauf, dass der Codierer einen Code vorgegeben hat. 5. Spieler rät den Code, indem er den Code als Zeichenfolge von 5 Zahlen eingibt. 6. Spieler wartet auf Feedback des Codierers. 7. Falls falsch geraten: Rät der Rater erneut bis zu maximal 10 mal. 8. Falls richtig, bis zum einschließlich 10. Mal raten: Codierer gibt Feedback, dass es der richtige Code ist, in dem er 5 mal Schwarz als Feedback gibt. 9. Spieler bekommt ein “Gewonnen” auf dem Terminal ausgegeben. 10. Der vorgegebene Code vom Codierer wird angezeigt.
Erweiterungen	1a. Spieler wählt falschen Spielmodus aus. Spieler gibt Neustart Kommando(reset) ein, um wieder in der Modusauswahl zu landen. 2a. siehe 1a 3a. siehe 1a 5a. Spieler gibt die falsche Anzahl an Zahlen ein. Fehlermeldung, die nochmal darauf hinweist, dass der Code 5 Zahlen lang sein muss. 5b. Spieler gibt andere Zeichen als Zahlen ein. Fehlermeldung, die nochmal darauf hinweist, dass der Code 5 Zahlen lang sein muss. 9a. Spieler gibt erneut Eingaben ein. Fehlermeldung, die darauf hinweist, dass die Runde vorbei ist und neu gestartet wird.

Codierer - Use Case: Spieler gewinnt

Primärakteur	Codierer
Anwendungsbereich	Superhirn
Niveau	Anwenderziel
Stakeholder und Interessen	Spieler: Möchte als Codierer das Spiel Superhirn spielen und gewinnen Auftraggeber (Prof. Dr. Salinger): Benötigt ein funktionierendes Spiel
Vorbedingung	Spieler hat das Spiel auf seinem PC, auf dem Windows/Mac oder Ubuntu läuft. Python neueste Version ist installiert. Spieler kennt die Regeln von Superhirn. Benutzer hat Superhirn gestartet.
Mindestzusicherung	Spiel gibt bei Fehlern Fehlermeldung aus.
Zusicherung im Erfolgsfall	Spieler konnte als Codierer das Spiel Superhirn erfolgreich spielen.
Haupterfolgsszenario	1. Spieler wählt als Spielmodus Codierer aus. 2. Spieler wählt den Benutzernamen, die Codelänge 5 und Farbanzahl 8. 3. Spieler gibt 5 Zahlen ein, von 1 bis 8. 4. Spieler wartet auf die Eingabe des Raters. 5. Spieler erhält Versuch vom Rater und 5.1 - setzt weder die Zahlen 7 noch 8, da nichts erraten wurde. 5.2 - setzt die Zahl 7 einmal oder mehrmals, da eine Zahl oder mehrere Zahlen erraten wurden, jedoch die Stelle falsch war. 5.3 - setzt die Zahl 8 einmal oder mehrmals, da eine oder mehrere Zahlen an der richtigen Stelle erraten wurden. 5.4 - setzt sowohl nichts als auch die Zahlen 7 oder 8, da einige Eingaben falsch waren, einige richtig aber an falscher Stelle und einige richtig, als auch an der richtigen Stelle waren. 6. Spieler wartet auf die Eingabe des Raters anhand des Feedbacks, falls der nicht den richtigen Code erraten hat, erhält einen neuen Versuch und setzt erneut Feedback. Dies wiederholt sich bis zum 10. Versuch des Raters, falls der den richtigen Code nicht erraten hat. 7. Nach dem 10. falschen Rateversuch gewinnt der Spieler, und der Rater verliert. Es wird “Gewonnen” auf dem Terminal ausgegeben. 8. Der vorgegebene Code vom Spieler wird gezeigt.

Erweiterungen	1a. Spieler wählt falschen Spielmodus. Spieler gibt Neustart Kommando(reset) ein, um wieder in der Modusauswahl zu landen. 2a. siehe 1a 3a. Spieler gibt die falsche Anzahl von Zahlen oder falsche Zahlen ein. Ergibt eine Fehlermeldung, die darauf hinweist, dass der Code 5 Zahlen lang sein muss und nur Zahlen zwischen 1 bis 8 akzeptiert. 3b. Spieler gibt andere Zeichen als Zahlen ein. Fehlermeldung, die nochmal darauf hinweist, dass der Code 5 Zahlen lang sein muss und nur Zahlen zwischen 1 bis 8 akzeptiert. 5a Spieler gibt die falsche Anzahl von Zahlen oder falsche Zahlen ein oder falsche Zeichen. Ergibt eine Fehlermeldung, die darauf hinweist, dass das Feedback eine leere Eingabe oder 5 Stellen lang sein kann und nur Zahlen 7 und/oder 8 akzeptiert. 7a. Spieler gibt erneut Eingaben ein. Fehlermeldung, die darauf hinweist, dass die Runde vorbei ist und neu gestartet wird.

Codierer - Use Case: Spieler verliert.

Primärakteur	Spieler
Anwendungsbereich	Superhirn
Niveau	Anwenderziel
Stakeholder und Interessen	Spieler: Möchte als Codierer das Spiel Superhirn spielen Auftraggeber (Prof. Dr. Salinger): Benötigt ein funktionierendes Spiel
Vorbedingung	Spieler hat das Spiel auf seinem PC, auf dem Windows/Mac oder Ubuntu läuft. Python neueste Version ist installiert. Spieler kennt die Regeln von Superhirn. Benutzer hat Superhirn gestartet.
Mindestzusicherung	Spiel gibt bei Fehlern Fehlermeldung aus.
Zusicherung im Erfolgsfall	Spieler konnte als Codierer das Spiel Superhirn erfolgreich spielen.
Haupterfolgsszenario	1. Spieler wählt als Spielmodus Codierer aus. 2. Spieler wählt den Benutzernamen, die Codelänge 5 und Farbanzahl 8 3. Spieler gibt 5 Zahlen ein, von 1 bis 8. 4. Spieler wartet auf die Eingabe des Raters.

	5. Spieler erhält Versuch vom Rater und 6. Falls falsch geraten (Codierer - Use Case: Spieler gewinnt). 7. Falls richtig: Spieler gibt Feedback, dass es der richtige Code ist, in dem der Spieler 5 mal Schwarz als Feedback gibt. 8. Spieler bekommt ein "Verloren" auf dem Terminal ausgegeben. 9. Der vorgegebene Code vom Codierer wird angezeigt.
Erweiterungen	1a. Spieler wählt falschen Spielmodus. Spieler gibt Neustart Kommando(reset) ein, um wieder in der Modusauswahl zu landen. 2a. siehe 1a 3a. Spieler gibt die falsche Anzahl von Zahlen oder falsche Zahlen ein. Ergibt eine Fehlermeldung, die darauf hinweist, dass der Code 5 Zahlen lang sein muss und nur Zahlen zwischen 1 bis 8 akzeptiert. 3b. Spieler gibt andere Zeichen als Zahlen ein. Fehlermeldung, die nochmal darauf hinweist, dass der Code 5 Zahlen lang sein muss und nur Zahlen zwischen 1 bis 8 akzeptiert. 5a Spieler gibt die falsche Anzahl von Zahlen oder falsche Zahlen ein oder falsche Zeichen. Ergibt eine Fehlermeldung, die darauf hinweist, dass das Feedback eine leere Eingabe oder 5 Stellen lang sein kann und nur Zahlen 7 und/oder 8 akzeptiert. 7a. Spieler gibt erneut Eingaben ein. Fehlermeldung, die darauf hinweist, dass die Runde vorbei ist und neu gestartet wird.

Codierer - Use Case: Spieler gibt falsches Feedback.

Primärakteur	Spieler
Anwendungsbereich	Superhirn
Niveau	Anwenderziel
Stakeholder und Interessen	Spieler: Möchte als Codierer das Spiel Superhirn spielen Auftraggeber (Prof. Dr. Salinger): Benötigt ein funktionierendes Spiel
Vorbedingung	Spieler hat das Spiel auf seinem PC, auf dem Windows/Mac oder Ubuntu läuft. Python neueste Version ist installiert. Spieler kennt die Regeln von Superhirn. Benutzer hat Superhirn gestartet.

Mindestzusicherung	Spiel gibt bei Fehlern Fehlermeldung aus.
Zusicherung im Erfolgsfall	Spieler konnte als Codierer das Spiel Superhirn erfolgreich spielen.
Haupterfolgsszenario	1. Spieler wählt als Spielmodus Codierer aus. 2. Spieler wählt den Benutzernamen, die Codelänge 5 und Farbanzahl 8. 3. Spieler gibt 5 Zahlen ein für die Positionen, jeweils von 1 bis 8. 4. Spieler wartet auf die Eingabe des Raters. 5. Spieler erhält Versuch vom Rater 6. Spieler gibt Feedback, das nicht mit dem Code und mit dem bisherigen Feedback nachvollziehbar ist. 7. Rater merkt, dass das Feedback nicht stimmen kann. 8. Rater gibt Rückmeldung, dass das Feedback nicht stimmt, das Spiel wird neu gestartet.
Erweiterungen	1a. Spieler wählt falschen Spielmodus. Spieler gibt Neustart Kommando(reset) ein, um wieder in der Modusauswahl zu landen. 2a. siehe 1a 3a. Spieler gibt die falsche Anzahl von Zahlen oder falsche Zahlen ein. Ergibt eine Fehlermeldung, die darauf hinweist, dass der Code 5 Zahlen lang sein muss und für die Positionen nur Zahlen zwischen 1 bis 8 akzeptiert. 4b. Spieler gibt andere Zeichen als Zahlen ein. Fehlermeldung, die darauf hinweist, dass der Code 5 Zahlen lang sein muss und für die Positionen nur Zahlen zwischen 1 bis 8 akzeptiert. 5a Spieler gibt die falsche Anzahl von Zahlen oder falsche Zahlen ein oder falsche Zeichen. Ergibt eine Fehlermeldung, die darauf hinweist, dass das Feedback eine leere Eingabe oder 5 Stellen lang sein kann und nur Zahlen 7 und/oder 8 akzeptiert.

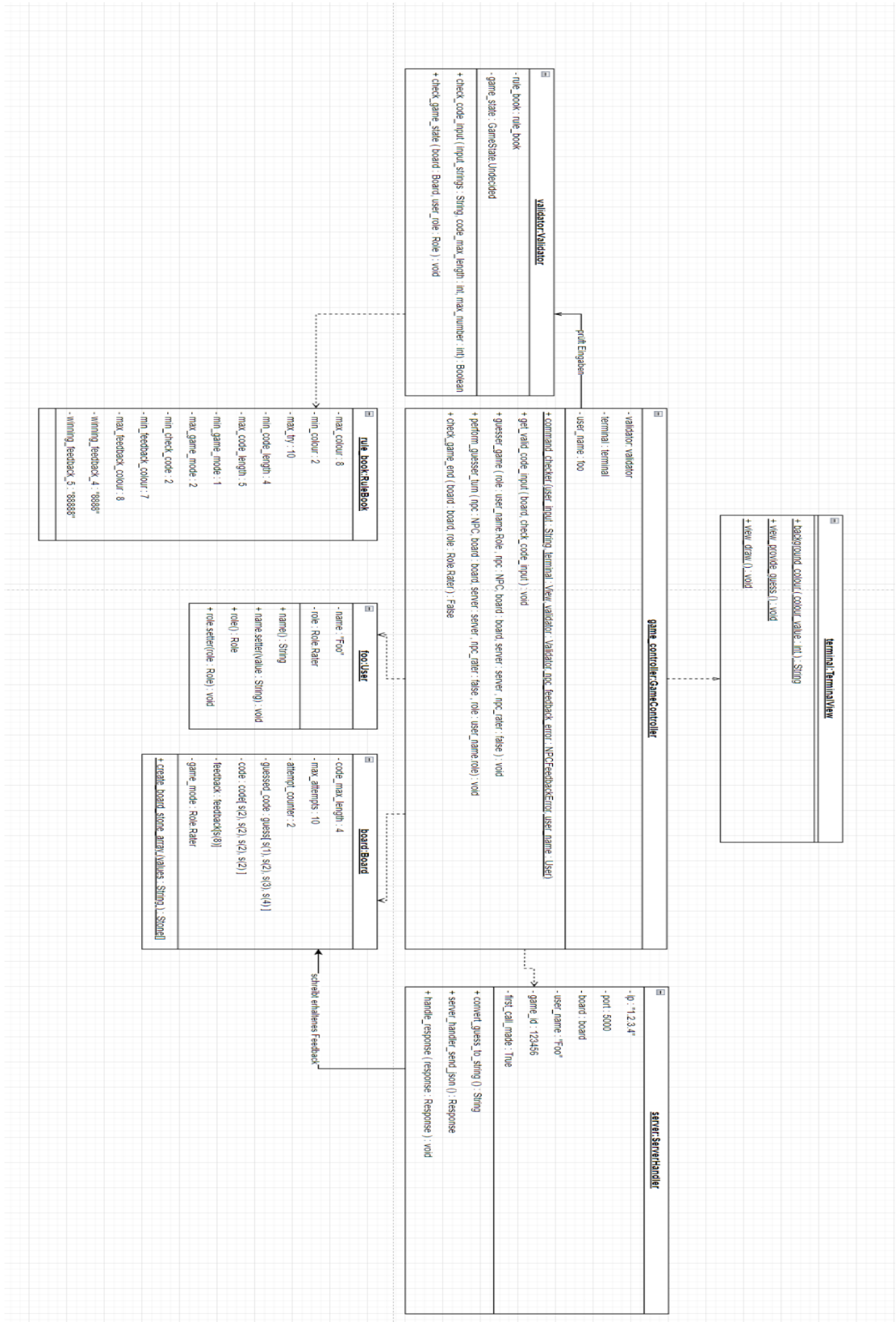
Rater - Use Case: Spieler gibt an, dass das Feedback falsch ist.

Primärakteur	Spieler
Anwendungsbereich	Superhirn
Niveau	Anwenderziel
Stakeholder und Interessen	Spieler: Möchte als Rater das Spiel Superhirn spielen Auftraggeber (Prof. Dr. Salinger): Benötigt ein

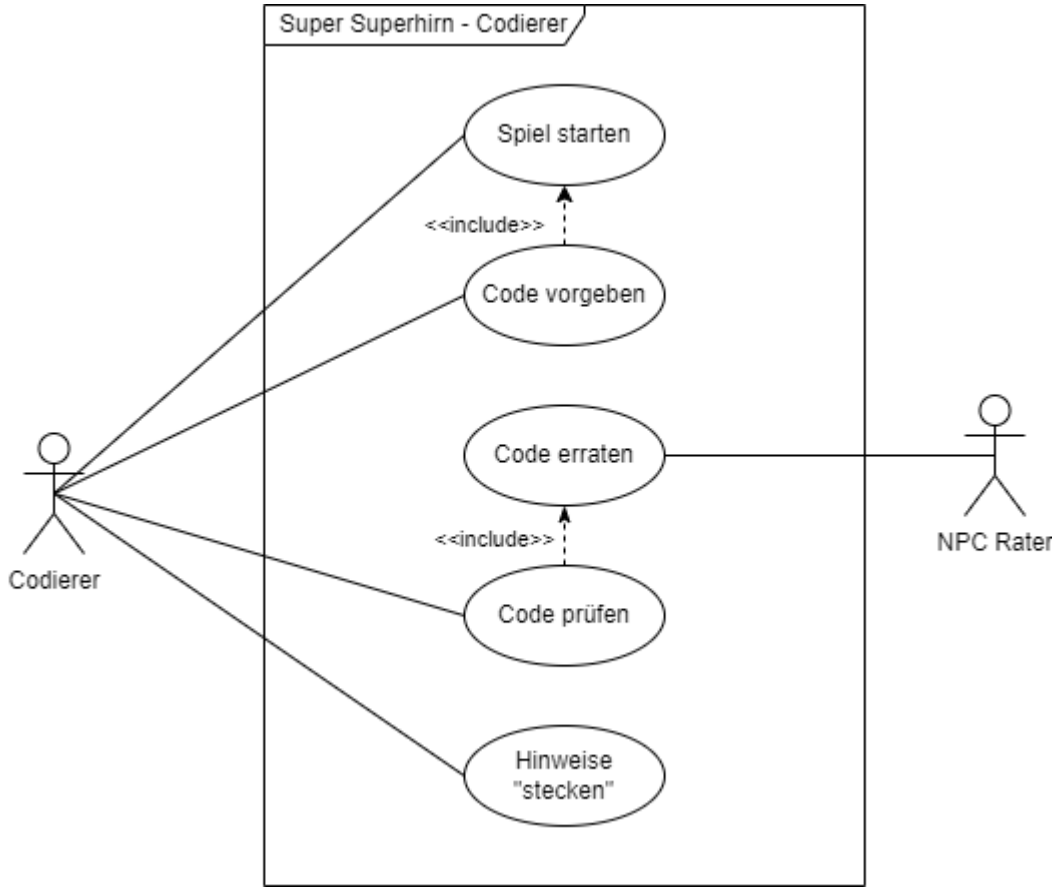
	funktionierendes Spiel
Vorbedingung	Spieler hat das Spiel auf seinem PC, auf dem Windows/Mac oder Ubuntu läuft. Python neueste Version ist installiert. Spieler kennt die Regeln von Superhirn. Benutzer hat Superhirn gestartet.
Mindestzusicherung	Spiel gibt bei Fehlern Fehlermeldung aus.
Zusicherung im Erfolgsfall	Spieler konnte als Rater das Spiel Superhirn erfolgreich spielen.
Haupterfolgsszenario	1. Spieler wählt als Spielmodus Rater aus. 2. Spieler wählt den lokalen oder Netzwerk Modus aus. 3. Spieler wählt den Benutzernamen, die Codelänge 5 und Farbanzahl 8. 4. Spieler wartet darauf, dass der Codierer einen Code vorgegeben hat. 5. Spieler rät den Code, indem er den Code als Zeichenfolge von 5 Zahlen, bei jeder Position jeweils von 1 bis 8, eingibt. 6. Spieler wartet auf Feedback des Codierers. 7. Codierer gibt Feedback, das nicht mit Code und mit bisherigen Feedback nachvollziehbar ist. 8. Spieler merkt, dass das Feedback nicht stimmen kann. 9. Spieler gibt Rückmeldung, dass das Feedback nicht stimmt, Runde wird beendet.
Erweiterungen	1a. Spieler wählt falschen Spielmodus. Spieler gibt Neustart Kommando(reset) ein, um wieder in der Modusauswahl zu landen. 2a. siehe 1a 3a. siehe 1a 5a. Spieler gibt die falsche Anzahl von Zahlen oder falsche Zahlen ein. Ergibt eine Fehlermeldung, die darauf hinweist, dass der Code 5 Zahlen lang sein muss und für die Positionen jeweils nur Zahlen zwischen 1 bis 8 akzeptiert. 5b. Spieler gibt andere Zeichen als Zahlen ein. Fehlermeldung, die nochmal darauf hinweist, dass der Code 5 Zahlen lang sein muss und für die Positionen jeweils nur Zahlen zwischen 1 bis 8 akzeptiert. 6a Spieler gibt die falsche Anzahl von Zahlen oder falsche Zahlen ein oder falsche Zeichen. Ergibt eine Fehlermeldung, die darauf hinweist, dass das Feedback eine leere Eingabe oder 5 Stellen lang sein kann und nur Zahlen 7 und/oder 8 akzeptiert.

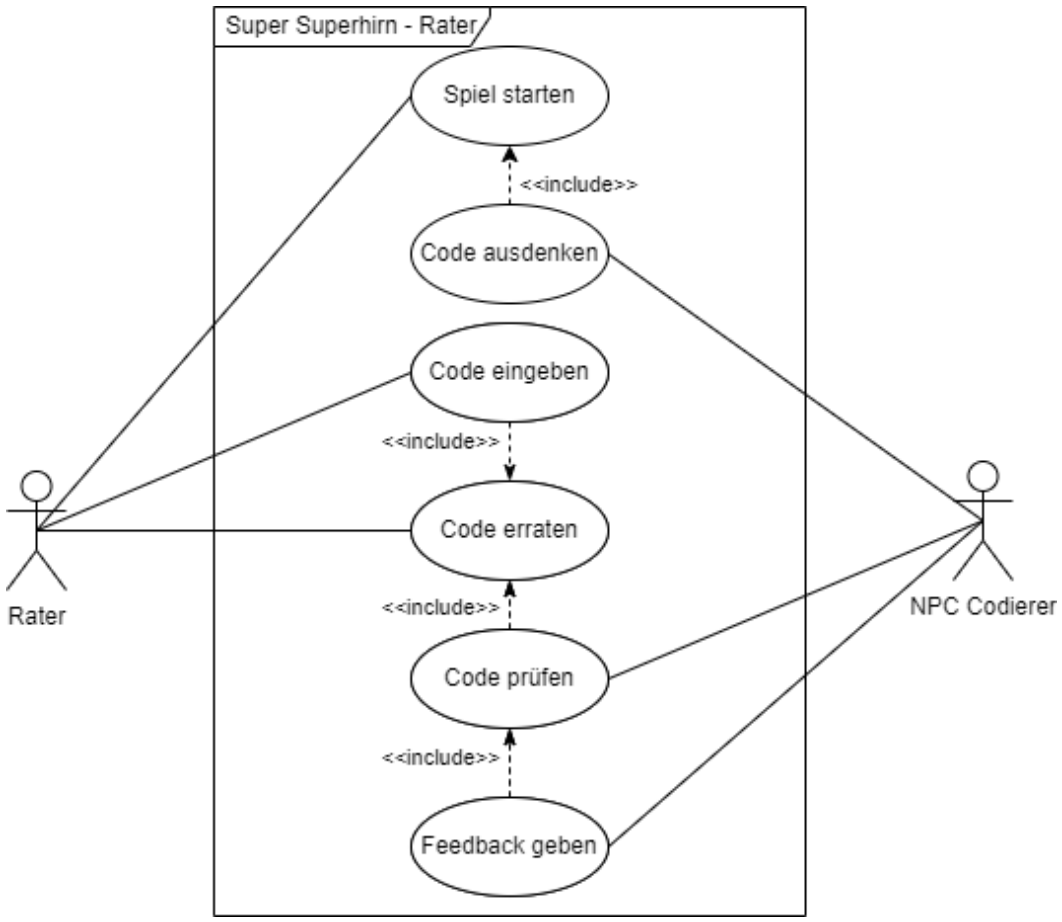
	8a. Spieler gibt erneut Eingaben ein. Fehlermeldung, die darauf hinweist, dass die Runde vorbei ist und neu gestartet wird.

Objektdiagramm

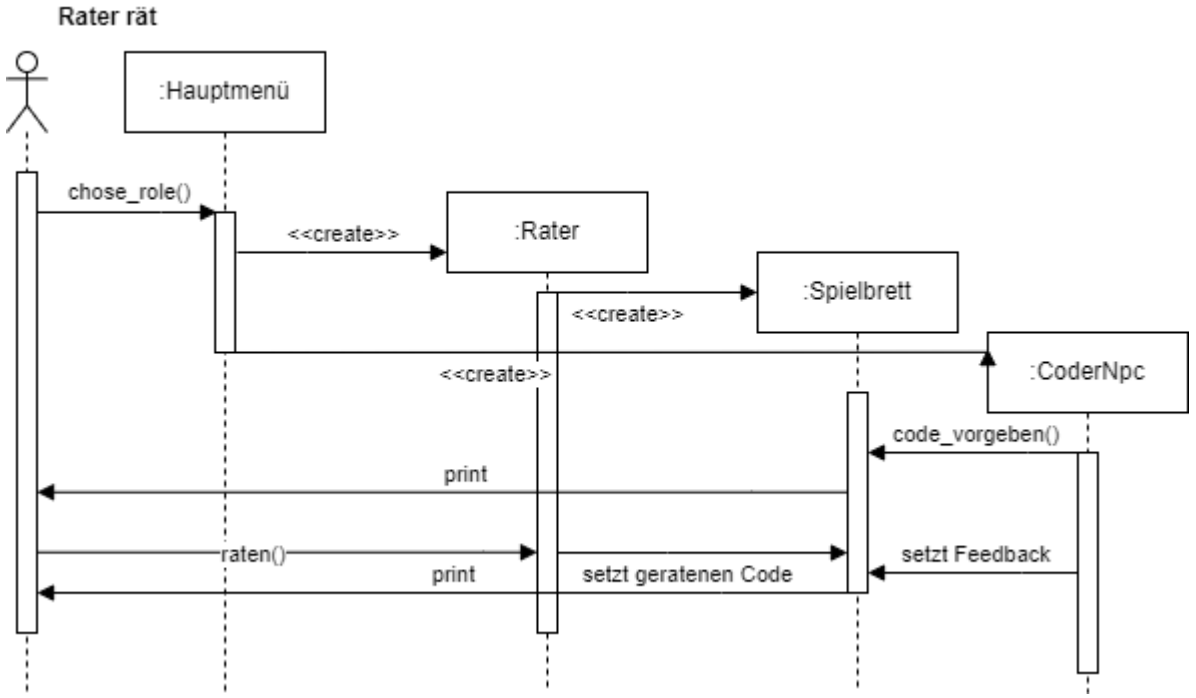


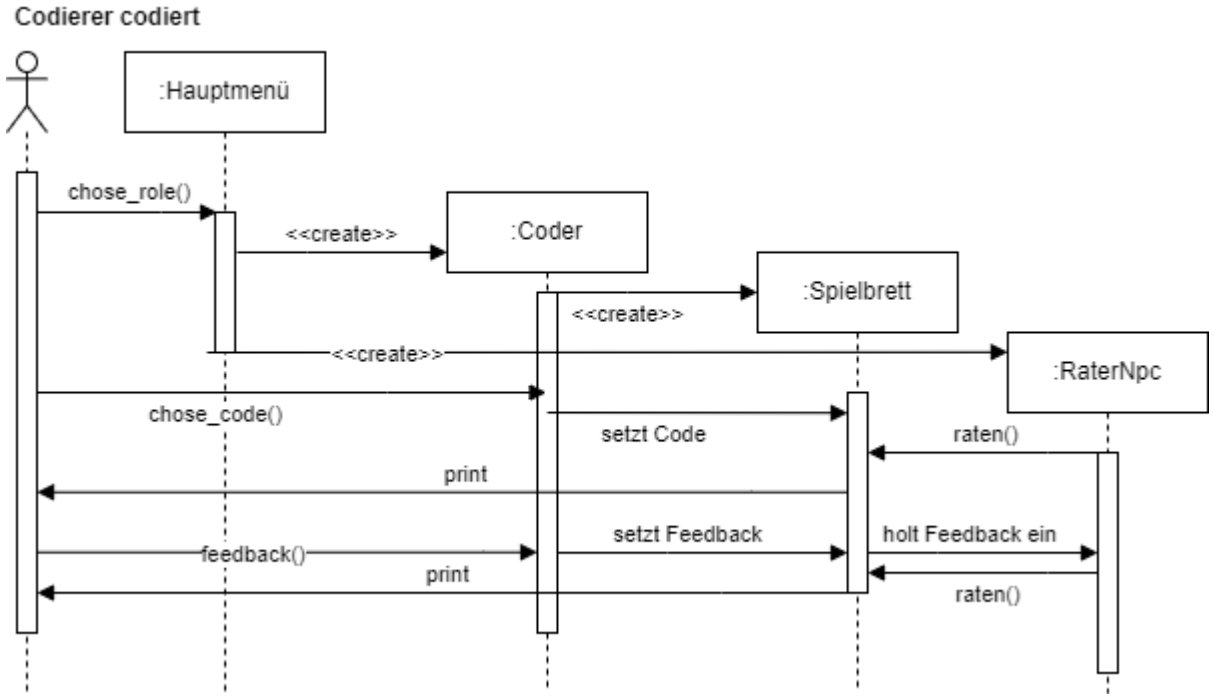
Use-Case Diagramme





Sequenzdiagramme





Design

Beschreibung des gewählten Designprozesses

Basierend auf den Ergebnissen der Analysephase wurde der Designprozess entwickelt. Nachdem die funktionalen und nichtfunktionalen Anforderungen erfasst wurden, wurde entschieden, wie diese umgesetzt werden sollten. Hierzu wurde das Gesamtsystem in überschaubare Einzelteile zerlegt.

Die Zerlegung diente dazu, die Aufgaben im Team zu verteilen und die Gesamtkomplexität zu reduzieren. Zuerst wurde ein Grobentwurf des Systems, wobei das Objektmodell als Leitfaden diente, erstellt. Das Hauptziel der Zerlegung des Projekts war die Identifizierung von kleineren unabhängigen Subsystemen. Es sollten Module, Einheiten von zusammengehörigen Programmelementen, geschaffen werden. Die Elemente innerhalb eines Moduls sind von denselben Entwurfsentscheidungen betroffen, verbergen diese aber vor dem restlichen System. Das dient dem „Information Hiding“ oder auch Verkapselung genannt. Für die Visualisierung wurde der UML Diagrammtyp Moduldiagramm verwendet. Solche Diagramme können eine Ansammlung weniger Klassen bis hin zu großen Subsystemen darstellen. Ein Vorteil der Moduldiagramme ist, dass das Verhalten der Komponente durch ihre benötigten und realisierten Schnittstellen definiert ist (vgl. Salinger 2021d, 13-20).

Beim Erstellen der Module standen die zwei Regeln der hohen Kohäsion und geringen Kopplung, als auch das Geheimnisprinzip, im Vordergrund. Die hohe Kohäsion bedeutet, dass der Inhalt, die Klassen, eines Moduls in Beziehung stehen sollen, als auch ähnliche Aufgaben erfüllen sollen. Die geringe Kopplung hingegen besagt, dass Änderungen an einem Subsystem nur geringe Auswirkungen auf ein anderes haben sollen (vgl. ebd., 23ff).

Ebenfalls wurde die Zwiebelarchitektur (Onion Architecture) verwendet. Die Zwiebelarchitektur ist ein Architekturstil, der das Ziel hat, die Abhängigkeiten in einem System zu kontrollieren und zu organisieren. Die Architektur definiert nicht nur die Bausteine eines Systems, sondern auch deren Beziehungen untereinander und deren wesentliche Merkmale (vgl. Salinger 2021e, 9f). Der Name Zwiebelarchitektur leitet sich von den Schichten ab, die ähnlich wie die Schichten einer Zwiebel angeordnet sind. Die klare Trennung der Verantwortlichkeiten ermöglicht es, die Test- und Wartbarkeit des Codes zu verbessern. Fundamental kann der Code innerhalb eines Ringes nur abhängig von dem Code

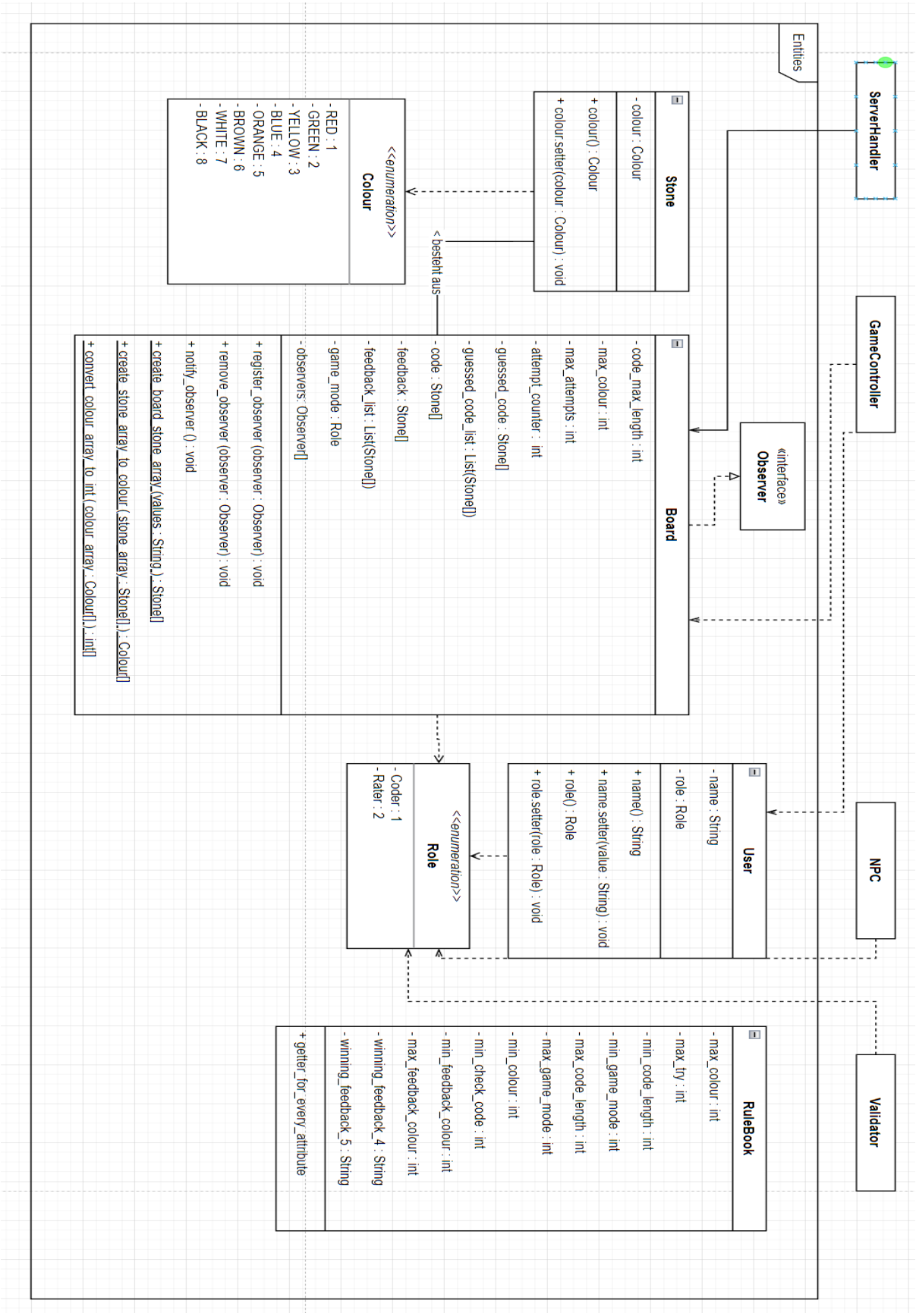
eines Rings sein, der nahe am Zentrum liegt, und nicht weiter außerhalb (vgl. Marbach 2015). Deswegen wurde diese Architektur für Super Superhirn gewählt. Entities repräsentieren die Kerndaten des Spiels, die Businesslogic Schicht implementiert die Spielregeln und die Presentation-Schicht ist für die Benutzerinteraktionen zuständig.

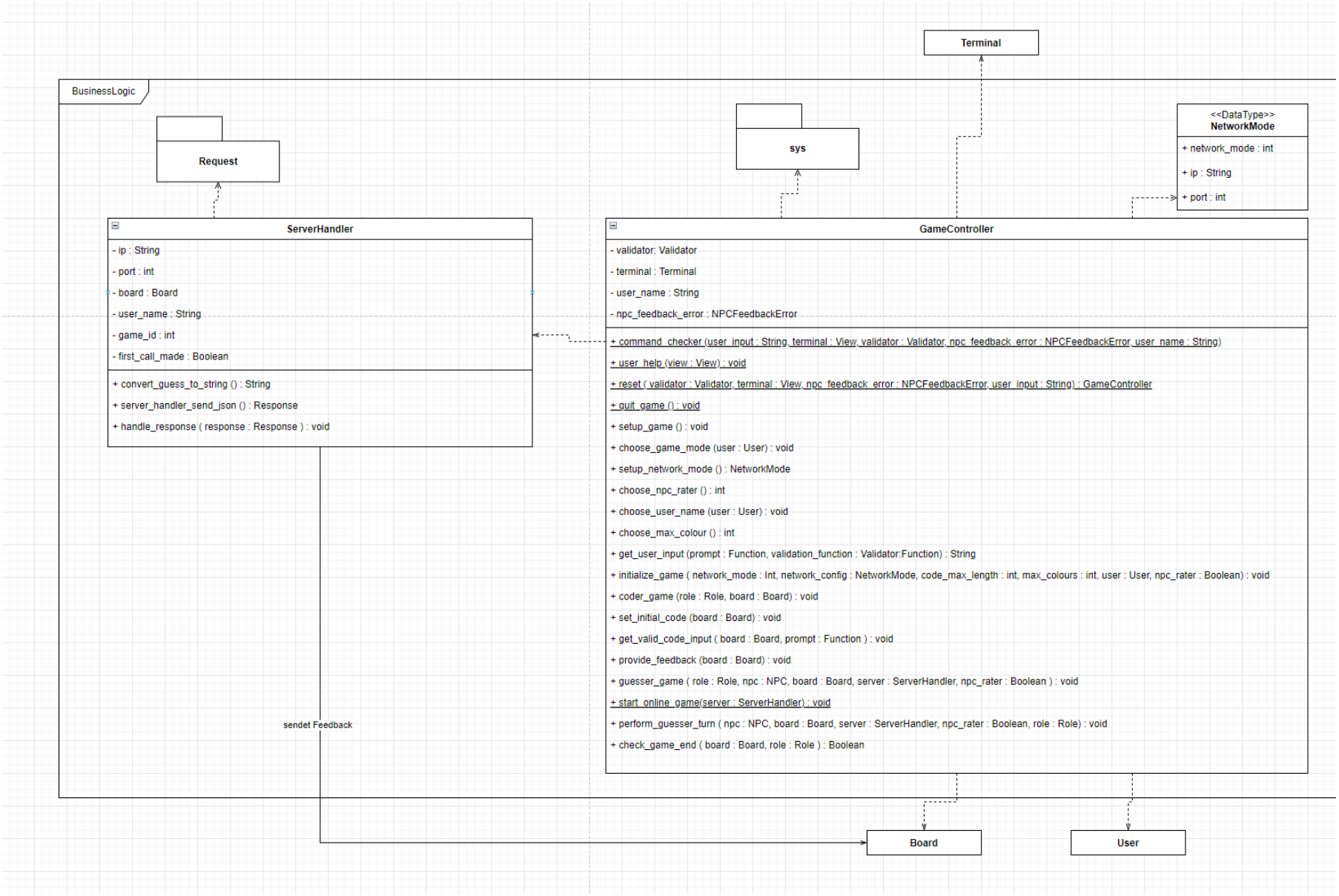
Zudem wurde das Entwurfsmuster des Observer Patterns verwendet. Es gehört zur Kategorie der Verhaltensmuster. Änderungen an einem Objekt werden automatisch an alle anderen abhängigen Objekte weitergeleitet. Es gibt ein Subject und Observer, die benachrichtigt werden. Diese sind aber nur lose gekoppelt, sodass Änderungen durchgeführt werden können, ohne den restlichen Teil des Systems zu beeinflussen (vgl. Eales et al., 2005, 163). In Super Superhirn sind Observer im Board implementiert.

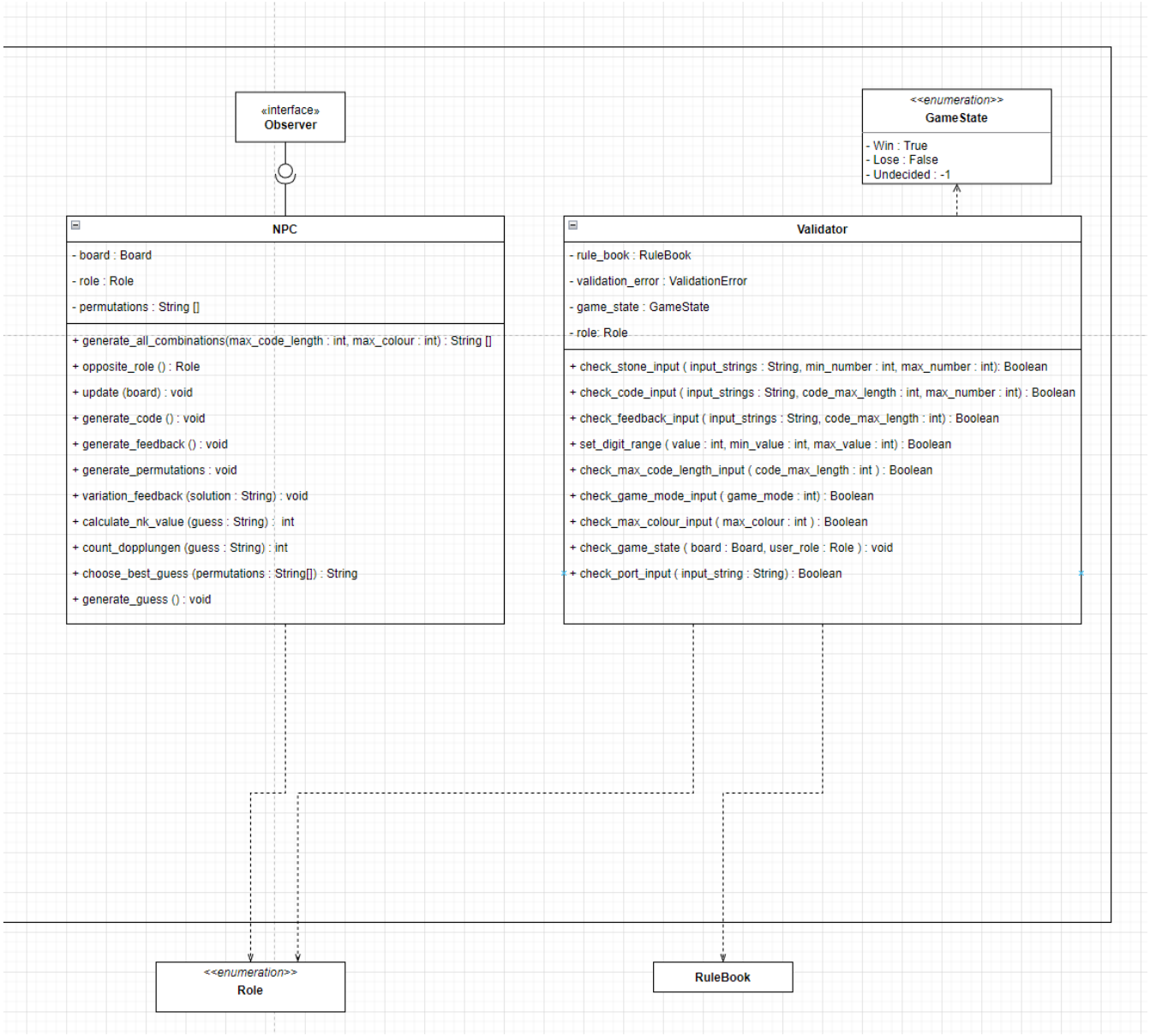
Beschreibung des entwickelten Designs

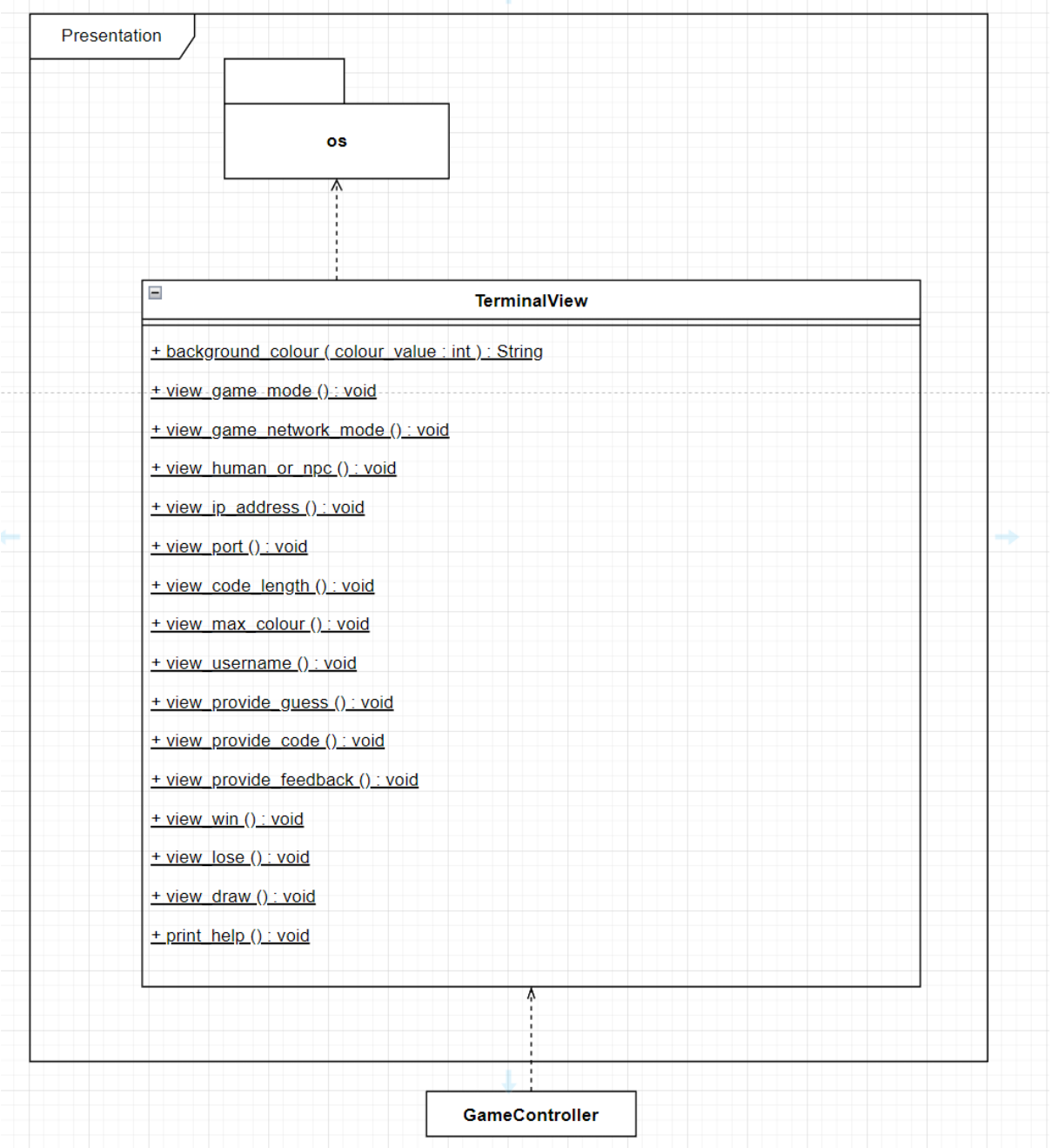
Der Aufbau von Super Superhirn ist dreigeteilt: Entities (Entitäten), Businesslogic und Presentation. Entities ist die innerste Schicht der Zwiebelarchitektur. Sie enthält die Entitäten oder Modelle, die den Kern des Datenmodells darstellen. Im Fall von Super Superhirn werden hier grundlegende Datenstrukturen wie zum Beispiel Stone, ein Objekt, welches einen Spielstein mit einer bestimmten Farbe modelliert, repräsentiert. Die Businesslogic Schicht liegt über der Schicht der Entities. Das ist der Kern der Anwendungslogik, der die Regeln des Spiels implementiert. Sie ist zum Beispiel verantwortlich für die Verarbeitung von Benutzereingaben. Die äußerste Schicht ist die Präsentationsschicht, die für die Benutzeroberfläche, als auch die Interaktionen mit dem Benutzer verantwortlich ist. Benutzereingaben werden entgegengenommen und Feedback wird präsentiert. Diese Schicht kommuniziert mit der Businesslogic Schicht. Die ausgewählte Onion Architecture ermöglicht es, die Komplexität des Spiels zu bewältigen und die Wartbarkeit des Codes zu erhöhen. Das Projekt kann effizient erweitert und gewartet werden, wobei jede Schicht eine klare Trennung der Verantwortlichkeiten hat. Ebenfalls haben Änderungen in einer Schicht minimale Auswirkungen auf die anderen.

Moduldiagramme

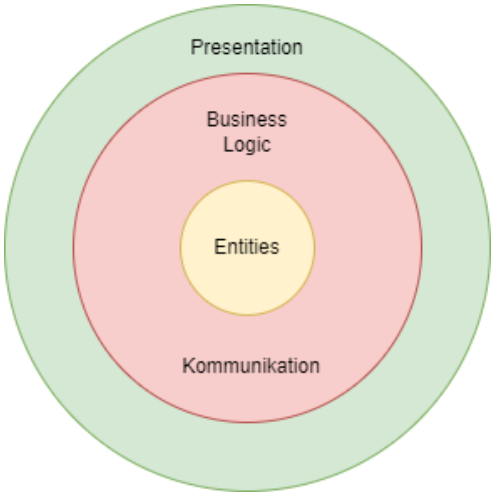



Terminal
BusinessLogic
Request
sys
<<DataType>>
NetworkMode
+ network_mode : int
+ ip : String
+ port : int
ServerHandler
- ip : String
- port : int
- board : Board
- user_name : String
- game_id : int
- first_call_made : Boolean
+ convert_guess_to_string () : String
+ server_handler_send_json () : Response
+ handle_response (response : Response) : void
GameController
- validator: Validator
- terminal : Terminal
- user_name : String
- npc_feedback_error : NPCFeedbackError
+ command_checker (user_input : String, terminal : View, validator : Validator, npc_feedback_error : NPCFeedbackError, user_name : String)
+ user_help (view : View) : void
+ reset (validator : Validator, terminal : View, npc_feedback_error : NPCFeedbackError, user_input : String) : GameController
+ quit_game () : void
+ setup_game () : void
+ choose_game_mode (user : User) : void
+ setup_network_mode () : NetworkMode
+ choose_npc_rater () : int
+ choose_user_name (user : User) : void
+ choose_max_colour () : int
+ get_user_input (prompt : Function, validation_function : Validator:Function) : String
+ initialize_game (network_mode : Int, network_config : NetworkMode, code_max_length : int, max_colours : int, user : User, npc_rater : Boolean) : void
+ coder_game (role : Role, board : Board) : void
+ set_initial_code (board : Board) : void
+ get_valid_code_input (board : Board, prompt : Function) : void
+ provide_feedback (board : Board) : void
+ guesser_game (role : Role, npc : NPC, board : Board, server : ServerHandler, npc_rater : Boolean) : void
+ start_online_game(server : ServerHandler) : void
+ perform_guesser_turn (npc : NPC, board : Board, server : ServerHandler, npc_rater : Boolean, role : Role) : void
+ check_game_end (board : Board, role : Role) : Boolean
sendet Feedback
Board
User


«interface»
Observer
<<enumeration>>
GameState
- Win : True
- Lose : False
- Undecided : -1
NPC
- board : Board
- role : Role
- permutations : String []
+ generate_all_combinations(max_code_length : int, max_colour : int) : String []
+ opposite_role () : Role
+ update (board) : void
+ generate_code () : void
+ generate_feedback () : void
+ generate_permutations : void
+ variation_feedback (solution : String) : void
+ calculate_nk_value (guess : String) : int
+ count_dopplungen (guess : String) : int
+ choose_best_guess (permutations : String[]) : String
+ generate_guess () : void
Validator
- rule_book : RuleBook
- validation_error : ValidationError
- game_state : GameState
- role: Role
+ check_stone_input (input_strings : String, min_number : int, max_number : int): Boolean
+ check_code_input (input_strings : String, code_max_length : int, max_number : int) : Boolean
+ check_feedback_input (input_strings : String, code_max_length : int) : Boolean
+ set_digit_range (value : int, min_value : int, max_value : int) : Boolean
+ check_max_code_length_input (code_max_length : int) : Boolean
+ check_game_mode_input (game_mode : int) : Boolean
+ check_max_colour_input (max_colour : int) : Boolean
+ check_game_state (board : Board, user_role : Role) : void
+ check_port_input (input_string : String) : Boolean
<<enumeration>>
Role
RuleBook



Zwiebelarchitektur



Implementierung

Erläuterung besonders interessanter Implementierungsmerkmale

In diesem Abschnitt werden die besonders interessanten Implementierungsmerkmale des Spiels Super Superhirn vorgestellt.

Für die Implementierung des Non-Player Characters für die Rater Rolle wurde der erweiterte Knuth Algorithmus verwendet. Der Algorithmus bestimmt am Anfang sämtliche möglichen Permutationen der Codes und speichert diese ab. Im erweiterten Ansatz ist die Auswahl einer vermeintlich zufälligen Permutation nicht komplett zufällig. Zunächst werden die Permutationen herausgefiltert, die die meisten Farbdopplungen haben. AAAA ist nur eine Farbdopplung, AABB sind zwei Farbdopplungen. Anschließend werden aus dieser Menge sämtliche Variationen für den Fall berechnet (n^k, Stichprobe, da sich die Farben wiederholen und die Reihenfolge wichtig ist), dass ein komplett negatives Feedback eintritt. Ein negatives Feedback bedeutet, dass keine Farbe übereinstimmt, auch nicht an der falschen Stelle. Die Auswahl erfolgt basierend auf der Permutation mit dem geringsten Wert an möglichen Variationen bei einem negativen Feedback. Nach jedem Rateversuch werden alle Permutationen herausgefiltert, die nicht dem tatsächlichen Feedback entsprechen. Das bedeutet, wenn der Rateversuch mit AABB ein komplett negatives Feedback bekommt, werden alle Permutationen, wo entweder A oder B enthalten sind, gelöscht. Der Prozess wiederholt sich, indem die kleinste Variationsmenge bei einem erneuten negativen Feedback berechnet wird. Der erweiterte Ansatz verbessert die Effizienz und führt zu gezielten Rate Versuchen (vgl. Mathematische Strategie bei Mastermind, o.D., Tänzer et al., o.D, 4ff).

Im Board sind Observer implementiert, die den Non-Player Character benachrichtigen, wenn es Änderungen gibt. Abhängig von der Rolle des NPCs reagiert dieser auf Veränderungen im Board mit entsprechenden Eingaben.

Daneben wurden verschiedene Programmierprinzipien im Entwicklungsprozess konsequent angewendet (vgl. Salinger 2021f, 14). Redundanter Code wurde vermieden, um Klarheit und Wartbarkeit zu verbessern (Don’t Repeat Yourself). Grundsätzlich sollte der Code sauberer hinterlassen werden, als er vorgefunden wurde (Boy Scout Rule). Der Code wurde so einfach gehalten, wie es nur ging, damit die Verständlichkeit des Codes gegeben war (Keep it simple and stupid) und überflüssige Funktionalitäten wurden vermieden (You aren’t gonna need it).

Durch die Zwiebelarchitektur und der Trennung von Anliegen wurden klare Abgrenzungen geschaffen (Separation of Concern). Zudem wurde das Open Closed Principle verwendet, indem ein Software-Artefakt offen für Erweiterungen, aber geschlossen für Modifikationen sein sollte (vgl. Meyer 1997, 57ff).

Qualitätssicherung

Beschreibung des gewählten QS-Prozesses/der gewählten QS-Strategie

Von Beginn an basierte der Software-Entwicklungsprozess auf agilen Methoden und einem iterativen Vorgehen. Die Gesamtarbeit wurde in einzelne Arbeitsschritte aufgeteilt, die eine geringere Komplexität aufwiesen. Es forderte sowohl die engere Kommunikation im Team als auch die Fähigkeit, mit veränderten oder unklaren Anforderungen klarzukommen. Des Weiteren wurde Scrum in Kombination mit Linear verwendet, um den Prozess der Entwicklung zu dokumentieren und Aufgaben in Sprints aufzuteilen. Alle Ausarbeitungen waren einzusehen, und die Transparenz hat das eigenverantwortliche Arbeiten geprägt. Ebenfalls haben die Sprint-Retrospektiven dem Team ermöglicht, die Entwicklung des Projektes zu verfolgen und zu verbessern. Die Ausarbeitungen wurden hinsichtlich der aktuellen Anforderungen geprüft. Zusätzlich hat das Aufteilen der Aufgaben je nach ihrem Ausführungsgrad dazu geführt, dass über die gesamte Entwicklungszeit ein Überblick gegeben war.

Zudem wurden, wie im Kapitel der Implementierung genannt, unterschiedliche Programmierprinzipien im Entwicklungsprozess verwendet.

Analytische und konstruktive Qualitätssicherung wurden ebenfalls während der Arbeit am Projekt Super Superhirn durch unterschiedliche Praktiken gewährleistet. Die konstruktive Qualitätssicherung ermöglicht es, Fehler frühzeitig zu erkennen, oder diese von Anfang an zu vermeiden, indem sichergestellt wird, dass sich der Erstellungsprozess an die definierten Qualitätsanforderungen hält. Das Prozessmodell gehört bereits zu einem Ansatz der Qualitätssicherung (vgl. Salinger 2023b). Zudem erlaubt das Erstellen der Use Case Diagramme, Sequenzdiagramme und Moduldiagramme einen klaren Überblick über die Struktur und Interaktionen des Systems zu gewinnen. Auf Basis der Diagramme wurden darüber hinaus Prototypen erstellt, die es ermöglichen, unterschiedliche Designideen

auszuprobieren und potenzielle Designfehler frühzeitig zu erkennen und zu beheben. Darüber hinaus wurde GitLab als Plattform für die Versionskontrollen eingesetzt. Unterschiedliche Branches wurden verwendet, um das gleichzeitige Arbeiten an verschiedenen Funktionen zu erlauben und die Fehlerbehebung zu erleichtern. Sollte eine weitere Anforderung des Spiels umgesetzt werden, wurde ein neuer Branch eröffnet. War die Arbeit an einem Branch beendet, wurde dieser durch Code-Reviews überprüft, bevor ein Merge (das Zusammenführen in den Hauptzweig) ausgeführt wurde. Änderungen und Implementationen müssen also dem Qualitätsstandard entsprechen. Die Qualität des Codes wurde mit Hilfe der Softwarerichtlinien bewertet. Der Quellcode musste einheitlich verfasst werden. Da das Projekt mit Python verfasst ist, hat sich das Projekt an die Style Guide Python Enhancement Proposal 8 (PEP8) gehalten (vgl. Salinger 2023c, 7ff). Zum Beispiel müssen Namenskonventionen eingehalten werden.

Die analytische Qualitätssicherung ist produktorientiert und basiert darauf, bereits entstandene Mängel aufzudecken. Dafür werden Teilprodukte nach ihrer Fertigstellung auf Qualität überprüft (vgl. Salinger 2023d, 6ff). In Super Superhirn wurde Test Driven Development eingesetzt, um schon im Voraus Unit Tests für die künftigen Module zu erstellen. Die einzelnen Funktionen und Methoden konnten damit isoliert getestet werden, um sicherzustellen, dass sie wie erwartet arbeiten. Ein Integrationstest prüfte die Wechselwirkung zwischen den verschiedenen Modulen und Schichten.
Zum Testen der Server-Komponente wurde privat mithilfe von Flask ein entfernter Server erstellt. Dieser kann Requests mit JSON’s, die den Anforderungen entsprechend erstellt wurden, als Inhalt annehmen, zufälligen Code erstellen nach den gesendeten Parametern und, in Abhängigkeit vom übersendeten Rateversuch, ein Feedback an die Server-Komponente in der eigentlichen App zurückgeben. Der Server lief im gleichen Netz und die Verbindung zwischen App und Server wurde mit Hilfe von zwei verschiedenen Maschinen erstellt.

Während der gesamten Implementation wurde Pair Programming verwendet. Mindestens zwei Gruppenmitglieder haben zusammen am Code gearbeitet.

Die Kombination von analytischen und konstruktiven Qualitätsmaßnahmen erlaube es, frühzeitig potenzielle Probleme zu erkennen, um somit die Qualität der Software kontinuierlich zu verbessern. Die iterative Arbeit im Verlauf der Entwicklung von Super Superhirn war der Ausgangspunkt der kontinuierlichen Optimierung und Anpassung.

Fazit

Die Entwicklung des Spiels Super Superhirn unter Anwendung unterschiedlicher Qualitätssicherungspraktiken hat zu einem erfolgreichen Ergebnis geführt. Die implementierten Features, wie zum Beispiel der erweiterte Knuth Algorithmus, oder das Anwenden von verschiedenen Programmierprinzipien, hat dazu beigetragen, dass das Spiel gemäß den Anforderungen durchgespielt werden kann. Der NPC Rater und NPC Codierer erfüllen ihre Aufgaben und reagieren auf das Feedback des Spielers bzw. auf das Feedback des Servers (im Fall des NPC Raters). Die Verwendung von Scrum mit dem agilen Projektmanagement-Tool Linear hat die Planung und Aufgabenverwaltung verbessert.

Der agil und iterativ gestaltete Entwicklungsprozess erlaubt es dem Team, flexibel auf Anforderungsänderungen und Anpassungen zu reagieren. Die Arbeit war in Sprints strukturiert, wodurch die Fortschritte sehr gut nachvollziehbar waren. Zudem hat die enge Kommunikation im Team dazu beigetragen, dass die Arbeitsergebnisse regelmäßig kontrolliert wurden. Dadurch war der Entwicklungsprozess stets zielgerichtet.

Im Rückblick auf das Projekt, gäbe es einige Stellen der Optimierung. Die stark begrenzte Zeit, als auch die Arbeiten in anderen Modulen, hat die Arbeitsaufteilung und Ausarbeitung erschwert. Zum Beispiel konnte nicht immer Zeit für das Pair Programming gefunden werden, weswegen die Arbeit sich nach hinten verschoben hat. Des Weiteren hätte das Überarbeiten der alten Diagramme frühzeitig eingeplant werden müssen. Da sich die Anforderungen während der Ausarbeitung verändert haben, mussten auch die Diagramme verändert werden. Insgesamt hätte die Planung der Arbeit noch weiter verbessert werden können.

Quellen

Eales, A., & Eales, A. (2005). The observer pattern revisited. *Educating, Innovating & Transforming: Educators in IT: Concise paper*.

Marbach, D. (2015). *Die Zwiebelarchitektur und ihre Vorzüge.* [online] https://entwickler.de/software-architektur/die-zwiebelarchitektur-und-ihre-vorzuge [abgerufen am 20.12.2023]

Mathematische Strategie bei Mastermind (o.D.). Meinstein. [online] https://meinstein.ch/math/mathematische-strategie-bei-mastermind/ [abgerufen am 20.12.2023]

Meyer, B. (1997). *Object-oriented software construction* (Vol. 2, pp. 331-410). Englewood Cliffs: Prentice hall.

Salinger, S. (2021a). *Folien Einheit 3: Lebenszyklus und Prozessmodelle* [Vorlesungsfolien]. Hochschule für Wirtschaft und Technik. https://moodle.htw-berlin.de/mod/resource/view.php?id=1401018

Salinger, S. (2021b). *Folien Einheit 6: Objektorientierte Analyse I / Anforderungserhebung* [Vorlesungsfolien]. Hochschule für Wirtschaft und Technik. https://moodle.htw-berlin.de/mod/resource/view.php?id=1401026

Salinger, S. (2021c). *Folien Einheit 8: Objektorientierte Analyse III / Objektmodell* [Vorlesungsfolien]. Hochschule für Wirtschaft und Technik. https://moodle.htw-berlin.de/mod/resource/view.php?id=1401034

Salinger, S. (2021d). *Folien Einheit 10: Objektorientiertes Design I / Modularisierung* [Vorlesungsfolien]. Hochschule für Wirtschaft und Technik. https://moodle.htw-berlin.de/mod/resource/view.php?id=1401043

Salinger, S. (2021e). *Folien Einheit 11: Objektorientiertes Design II / Architektur* [Vorlesungsfolien]. Hochschule für Wirtschaft und Technik. https://moodle.htw-berlin.de/mod/resource/view.php?id=1401047

Salinger, S. (2023a). *Folien Übung 4* [Vorlesungsfolien]. Hochschule für Wirtschaft und Technik. https://moodle.htw-berlin.de/mod/resource/view.php?id=1516737

Salinger, S. (2023b). *Folien Einheit 1: Softwarequalität* [Vorlesungsfolien]. Hochschule für Wirtschaft und Technik. https://moodle.htw-berlin.de/mod/resource/view.php?id=1516642

Salinger, S. (2023c). *Folien Einheit 3: Konstruktive Qualitätssicherung 1* [Vorlesungsfolien]. Hochschule für Wirtschaft und Technik. https://moodle.htw-berlin.de/mod/resource/view.php?id=1516643

Salinger, S. (2023d). *Folien Einheit 7,8 und 9: Analytische Qualitätssicherung 1,2 und 3* [Vorlesungsfolien]. Hochschule für Wirtschaft und Technik. https://moodle.htw-berlin.de/mod/resource/view.php?id=1516647

Schwaber, K., & Sutherland, J. (2011). The scrum guide. *Scrum Alliance*, *21*(1), 1-38.

Tänzer, N., & Fiedler, F. (o.D.). Praktikumsbericht im Rahmen des Praktikums Paralles Programmieren: Mastermind. Hamburg: Universität Hamburg, Fachbereich Informatik.

Linear. (s.d.). Linear: Agile Projektmanagement-Tool. Abgerufen von https://linear.app/