Angewandte Informatik - HTW Berlin

Software Engineering 2

Mastermind

August 13, 2024

Students:	Dennis Forster	(586888)
	Daniil Serdyuk	(587626)
	Jaden Elijah Diodone	(587916)
	David Brundert	(588563)
	Christian Trefzer	(590062)
Supervisor:	Alexander Kramer	

Contents

List of Figures **2**

1 Einleitung **3**
1.1 Aufgabenbeschreibung . . . 3
1.2 Projektorganisation . . . 3

2 Requirements Engineering **4**
2.1 Funktionale Anforderungen . . . 4
2.1.1 Spielstart und Moduswahl . . . 4
2.1.2 Spielablauf . . . 5
2.1.3 Leaderboard . . . 5
2.1.4 Persistenz . . . 6
2.2 Nicht-funktionale Anforderungen . . . 6
2.2.1 Qualitätsanforderungen . . . 6
2.2.2 Rahmenbedingungen . . . 7
2.2.3 Eingabeaufforderung . . . 7
2.2.4 Darstellung . . . 8
2.2.5 Automatischer Löser . . . 8
2.3 Anforderungsänderung vom 7.6.2024 . . . 8
2.4 Anforderungsänderung vom 21.6.2024 . . . 8

3 Objektorientierte Analyse **10**
3.1 Analyseprozess (OOA) . . . 10
3.2 Use Case Diagramme nach Cockburn . . . 11
3.3 Klassendiagramm . . . 18
3.4 Sequenzdiagramm . . . 20
3.5 Zustandsdiagramm . . . 21
3.6 Qualitätsmanagement OOA . . . 21

4 Objektorientiertes Design **23**
4.1 Beschreibung des Designprozesses . . . 23
4.2 UI Mockups . . . 23
4.3 Klassendiagramm . . . 25
4.4 Zustandsdiagramm . . . 26
4.5 Sequenzdiagramm . . . 27
4.6 Architektur . . . 27
4.7 Entwurfsmuster . . . 27
4.8 Qualitätsmanagement . . . 28

5 Implementierung **30**
5.1 NavigationManager-Klasse . . . 30
5.2 Evaluation . . . 31
5.3 Server . . . 31
5.4 Rater-Bot . . . 32

6 Qualitätssicherung **33**
6.1 Qualitätssicherungsprozess . . . 33
6.2 Qualitätssicherungsmaßnahmen . . . 33
6.2.1 Richtigheit . . . 33

6.2.2 Zuverlässigkeit . . . 33
6.2.3 Änderbarkeit . . . 33
6.2.4 Sonstige Maßnahmen . . . 34

7 Fazit **35**
7.1 Beurteilung des Ergebnisses . . . 35
7.2 Beurteilung des Prozesses . . . 35
7.3 Mögliche Verbesserungen . . . 35

8 Anhang **36**
8.1 Fragenkatalog für das Interview mit dem Projektsponsor . . . 36
8.2 Ergebnisse des Interviews . . . 37
8.3 Anforderungen aus dem Projekt-Auftrags . . . 38
8.4 Letztendliche User-Interfaces . . . 39
8.5 Detailansicht Klassendiagramm OOD . . . 39

List of Figures

1 Kanban-Board zum Zeitpunkt der objektorientierten Analyse (OOA). . . . 4
2 Klassendiagramm . . . 19
3 Sequenzdiagramm . . . 20
4 Zustandsdiagramm . . . 21
5 UI Mockups . . . 24
6 Klassendiagramm der OOD-Phase. Vergrößerte Ansicht im Anhang verfügbar (8.5). . . . 25
7 Zustandsdiagramm einer willkürlichen Control-Klasse mit den von dieser implementierten Methoden. . . . 26
8 Sequenzdiagramm eines Tutorial-Aufrufs . . . 27
9 Klassendiagramm des fertigen Projektes. Klassen des UI in Cyan, der Controller in Gelb und der Models in Magenta. In der ersten Reihe wird deutlich, wie das Observer-Muster zwischen Model und UI umgesetzt wird. . . . 30
10 Auschnitte der letztendlichen User-Interfaces . . . 39
11 OOD Klassendiagramm . . . 40

1 Einleitung

Im Folgenden werden die Aufgabe des Projekts und die gewählte Projektorganisation beschrieben.

1.1 Aufgabenbeschreibung

Ziel des Projekts ist es, eine Abwandlung des Spiels "Super Superhirn" zu entwickeln. In dem Spiel gibt es maximal 8 verschiedene Farbcodes (2-8) und maximal 5 Steckplätze (4-5) zum Platzieren der Farben. Es gibt 2 Rollen: die Rolle des Kodierers und die des Raters. Der Kodierer überlegt sich eine Folge von 5 Farben, die der Rater erraten muss. Dabei hat der Rater maximal 12 Versuche. Wenn der Rater eine Farbkombination errät, bekommt er Feedback in Form von Farbpins, die besagen, wie viele Farben richtig erraten worden sind und sich an der falschen Position befinden (weiße Pins) und wie viele richtig erraten worden sind und sich an der richtigen Position befinden (schwarze Pins). Wenn kein Pin gesetzt worden ist, befindet sich keine der Farben in der Zielkombination. Die gesamte Spielhistorie ist über den Verlauf des Spiels sichtbar. Wenn 5 schwarze Pins (oder 4 in der einfacheren Variante) gesetzt worden sind, bevor die maximale Zuganzahl erreicht wurde, gewinnt der Rater, andernfalls der Kodierer. Als Spieler soll man entweder als Rater oder als Kodierer agieren können. Das Programm übernimmt dann die jeweils andere Rolle.

1.2 Projektorganisation

Zur Projektorganisation wurde ein Kanban-Board in Trello erstellt. Kanban ist deutlich flexibler als klassische Modelle, wie das Wasserfallmodell, da zu jeder Zeit neue Anforderungen als Tasks beliebig in das Kanban-Board aufgenommen werden können und nicht gänzlich zur vorherigen Phase zurückgesprungen werden muss. Gleichzeitig ist Kanban einfacher als andere agile Methoden, wie Scrum, da bspw. keine daily Standup-Meetings Teil des Modells sind. Es ist eine einfache Möglichkeit, sich über den aktuellen Stand des Projekts zu informieren, ohne tägliche Meetings zu halten. Es können außerdem Blockaden im Kanban-Board markiert werden. Dadurch sehen andere Teammitglieder, wann und wo es Probleme gibt und können dabei flexibel aushelfen. Gleichzeitig werden wir auch regelmäßige Meetings mit dem Team abhalten. Allerdings keine kurzen täglichen, sondern längere, die alle ein bis zwei Wochen stattfinden.

Das Kanban-Board beinhaltet bei uns die Phasen "Backlog", "To Do", "In Progress" und "Done". Das Backlog beinhaltet die Aufgaben, die für zukünftige Meilensteine relevant sind. Die Farbmarkierung für die Kanban-Tasks ist Blau für Dokumentation, Gelb für OOA (objektorientierte Analyse), Orange für OOD (objektorientiertes Design), Rot für Implementierung, Grün für Abschluss. Eine maximale Anzahl von Aufgaben innerhalb der jeweiligen Phasen wird von uns nicht festgelegt, um bei der konkreten Bearbeitung der Schritte flexibler zu sein.

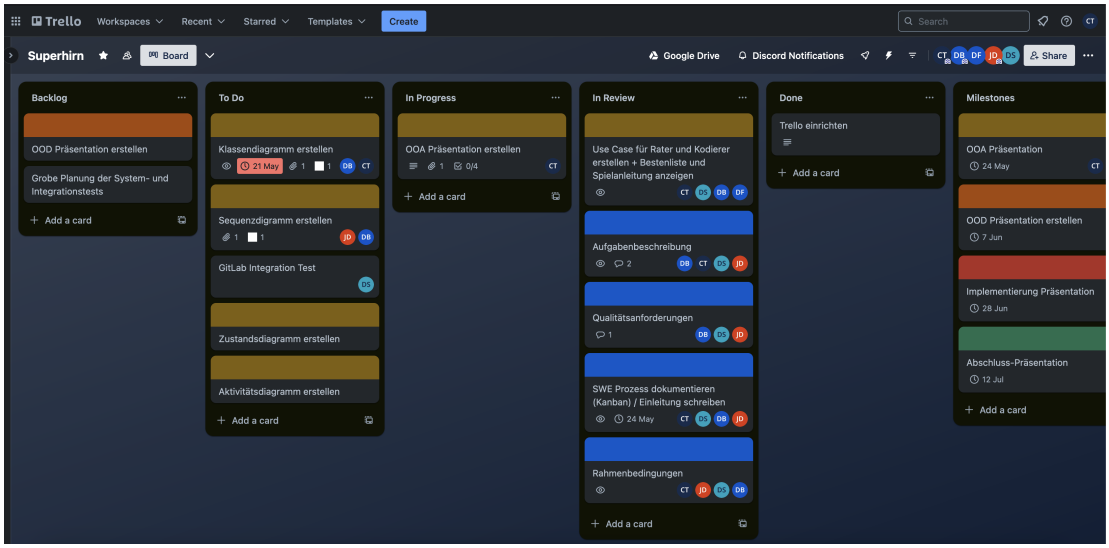


Figure 1: Kanban-Board zum Zeitpunkt der objektorientierten Analyse (OOA).

2 Requirements Engineering

Zur Identifizierung und Dokumentierung der Anforderungen zogen wir verschiedene Quellen heran: Die Materialien aus der Vorlesung und der Übung, Praxiserfahrungen mit Varianten des Spiels "Superhirn" sowie ein Interview mit dem Dozenten, der in der Rolle des Sponsors agiert.

Das Interview ist ein zentraler Bestandteil des Analyseprozesses. Um es effizient und zielgerichtet zu gestalten, wird im Vorfeld ein detaillierter Fragenkatalog erstellt. Der Fragenkatalog dient dazu, die relevanten Aspekte des Projekts gezielt anzusprechen und klare, präzise Antworten zu erhalten. Außerdem dient er zur Verifikation von Annahmen. Die gewonnenen Erkenntnisse aus dem Interview wurden anschließend ausgewertet und zusammengefasst.
Der Fragenkatalog kann im Anhang (8.1) eingesehen werden.

Zusätzlich befassten wir uns mit verschiedenen Abwandlungen des Spiels "Superhirn", wie zum Beispiel "Wordle", um ein besseres Verständnis für die Mechaniken und Herausforderungen des Spiels zu entwickeln. Diese praktische Erfahrung ermöglichte es, die Anforderungen realistischer und detaillierter zu formulieren.

Durch dieses Vorgehen konnten wir die Anforderungen methodisch erfassen und präziser formuliern. Dadurch hatten wir eine fundierte Grundlage für die Umsetzung des Projekts geschaffen.

2.1 Funktionale Anforderungen

2.1.1 Spielstart und Moduswahl

- Das Spiel soll eine Option bieten, ein neues Spiel zu starten.
- Der Spieler soll zwischen den Rollen "Rater" und "Kodierer" wählen können, wobei der Computer die jeweils andere Rolle übernimmt.
- Das Spiel soll die Möglichkeit bieten, die Spielregeln oder eine Hilfe-Funktion anzuzeigen.

- Das Spiel sollte über die Konsole gestartet werden können mit z.B. Eingabe von `python superhirn.py`.
- Es können Argumente mitgegeben werden, um direkt in einen Spielmodus zu starten, z.B. `--Tutorial`, `--Highscore`, `--Guesser`, `--Coder`.
- Argumente können frei gewählt werden und auch Abkürzungen verwendet werden, beides sollte nachvollziehbar sein.
- Bei `--Tutorial` und `--Highscore`-Argumenten soll nur das erste genommen werden.
- Zur Evaluation der Rateversuche sollen zwei Modi ausgewählt werden können: Das lokale Programm oder ein Online-Modus, in dem neue Rateversuche an einen externen Server übermittelt werden.

2.1.2 Spielablauf

- Ein Spieler hat maximal 12 Rateversuche.
- Die Breite des zu erratenden Codes ist vier oder fünf Zeichen.
- Es gibt kein Zeitlimit.
- Als Rater: Der Spieler gibt Farbcodes ein, um den geheimen Code zu erraten.
- Als Kodierer: Der Spieler gibt den Farbcode ein, den der Computer erraten soll.
- Das System berechnet automatisch die Anzahl der richtigen Farben und Positionen und gibt entsprechendes Feedback.
- Bei Spielende soll der richtige Code angezeigt werden.
- Nach Spielende soll der Spieler die Wahl haben, ein weiteres Spiel mit den gleichen Einstellungen zu starten oder ein neues Spiel zu beginnen.
- Die Eingabe erfolgt über eine Eingabezeile.
- Spiel kann jederzeit mit `Q` beendet werden (Aufruf von Highscore und Tutorial jederzeit optional, bei Aufruf während des Spieles muss in den vorherigen Zustand zurückgesprungen werden).
- Ergebnis wird angezeigt, wenn keine Lösung gefunden ist.
- Rateversuche können wiederholt werden (z.B. zweimal Eingabe desselben Codes, gilt dann beide Male als Versuch).
- Formal falsche Eingaben führen zu einer Fehlermeldung, Eingabe kann aber korrigiert werden (kein Fehlversuch). Fehlermeldung gibt Hinweis auf korrekte Eingabe.

2.1.3 Leaderboard

- Nach gewonnenem Spiel kann der Spieler seinen Namen eingeben.
- Es soll ein Leaderboard geben, das nur menschliche Spieler einträgt.
- Bei gleicher Anzahl von Versuchen entscheidet die Zeit über die Platzierung im Leaderboard.
- Computerrater erscheint NICHT auf dem Leaderboard.

- Anzeige der besten zehn (10) Spieler.
- Mehrfachnennung möglich.

2.1.4 Persistenz

- Das System soll lokale Speicherung ermöglichen, um den Spielstand und das Leaderboard zu sichern.
- Highscore soll als Datei persistent gespeichert werden.
- Laden der Highscores beim Spielstart.
- Datenformat der Speicherung kann frei gewählt werden, sollte aber nachvollziehbar sein.

2.2 Nicht-funktionale Anforderungen

2.2.1 Qualitätsanforderungen

- Speicherverbrauch ist keine relevante Anforderung (sollte keine Hunderte von MB verbrauchen/belegen).
- Spiel sollte keine erkennbaren Verzögerungen z.B. beim Bildaufbau oder Reaktion auf Eingaben haben.
- Sprache: Deutsch.
- Textverständnis ab 12 Jahren.
- Python-Version: letzte Long-Term-Support-Version (derzeit 3.12.x).
- Fehlerfreiheit in der Durchführung.
- Ausführung von Source-Code, kein Binary.
- Erweiterbarkeit.
- Der Code soll so strukturiert und dokumentiert sein, dass er leicht zu ändern und zu warten ist.
- Codekommentierung: Englisch.
- Das Spiel soll auf jedem Betriebssystem laufen, das Python 3 unterstützt.
- Das System soll reaktionsschnell sein und keine wahrnehmbaren Verzögerungen bei der Benutzerinteraktion aufweisen.
- Wenn der Spieler die Rolle Rater hat, soll das System immer korrekt antworten.
- Das System soll robust gegen Fehleingaben sein und entsprechende Fehlermeldungen anzeigen.
- Es soll keine Fehler geben, die zu Programmabstürzen führen.
- Das Spiel muss objektorientiert in der letzten Long-Term-Support-Version von Python 3 implementiert werden.
- Das Spielfeld soll min. 13-14 Zeichen x 20 Zeilen umfassen.
- Nach jedem Spielzug baut sich das Spielbrett auf der Konsole neu auf.

- Farben können mehrfach verwendet werden, aber es müssen mindestens 2 verschiedene Farben zu Codeerzeugung genutzt werden.
- Das Spiel soll allgemeinverständlich und ab 12 Jahren geeignet sein.

2.2.2 Rahmenbedingungen

- Farbkodierung ist die folgende: 1=Rot, 2=Grün, 3=Gelb, 4=Blau, 5=Orange, 6=Braun, 7=Weiß (Code sowie Bewertung), 8=Schwarz (Code sowie Bewertung).
- Es soll sich nicht um eine Web-Applikation handeln, es muss auf der Konsole spielbar sein.
- Implementierung in deutscher Sprache.
- Entwurf und Analyse müssen den Prinzipien und Methoden aus SE 1 folgen.
- Die Modellierungssprache ist UML 2.5.
- Der Code, die Dokumentation und die Qualitätssicherung sind unter Git zu verwalten und unter der GitLab Instanz der HTW zu hosten.
- Der Name des Projekts auf GitLab ist `WiSe23-24_Superhirn_12`.
- Der Dozent muss Owner des Projekts auf GitLab sein.
- Bibliotheken sind erlaubt, solange sie keinen direkten Einfluss auf die Spiellogik haben (Numpy ist erlaubt).
- Codefragmente Dritter dürfen nicht eingebaut werden.
- Aller Code muss mit Pair Programming erstellt werden. Insbesondere muss aus jedem Commit hervorgehen, welche beiden Entwickler beteiligt waren.
- Der Kodierer-Bot muss zufällige Farbkombinationen setzen.
- Der Kodierer-Bot soll keine Codes mit nur einer Farbe erstellen.
- Der Rater-Bot darf keine perfekten Züge bzw. Spiele machen, sondern benötigt ein gewisses Ausmaß an Unschärfe.
- Es gibt keine Auswahl des Schwierigkeitsgrades.
- Der Spielstart kann über die übergebenen Parameter beim Ausführen des Programms erfolgen (dazu zählt Spielmodus und Tutorialflag).

2.2.3 Eingabeaufforderung

- Dateiname hat keine Vorgabe, sollte aber nachvollziehbar sein.
- Spiel sollte in einem eigenen Fenster starten mit für das Spielfeld angebrachter Größe.
- Auf Veränderung der Fenstergröße muss nicht reagiert werden (kann aber).
- Bestätigen von Eingaben auf der Eingabezeile mit ENTER.
- Eingabe (Q, T, H etc.) ist case-insensitive.
- Eingabe vom Code nacheinander ohne Trenn- und Leerzeichen (z.B. `12345[Enter]`).

2.2.4 Darstellung

- Es wird immer das komplette Spielfeld dargestellt (auch nicht gespielte Zeilen).
- Legende mit der Farbcodierung während des Spielens immer sichtbar.
- Verwendung von Farben zur Anzeige OPTIONAL (geht nicht in die Bewertung ein).
- Das Spielfenster zeigt die Startzeit an.
- Anzeige der laufenden Spielzeit OPTIONAL.
- Nach Spielende Anzeige der Delta-Zeit im Interface.
- Es kann mit Pop-Up-Fenster als Feedback und Eingabe gearbeitet werden, es reicht aber auch die Arbeit mit einer Eingabezeile und einer Frage-, Feedback-Zeile ober/unterhalb der Eingabezeile.

2.2.5 Automatischer Löser

- Automatische Lösung soll in der Darstellung nachvollziehbar sein (z.B. 0,5 – 1,0 Sekunden pro Zeile).
- Lösungsalgorithmus soll einfache Unschärfe haben (PC soll mit Wahrscheinlichkeit auch einmal falsch raten).
- Keine Einbindung von Machine Learning in den Lösungsalgorithmus, numpy kann verwendet werden.

2.3 Anforderungsänderung vom 7.6.2024

Freie Wahl der Anzahl der Farben

- Es ist möglich, zu Beginn jedes Spiels festzulegen, mit wieviel Farben gespielt werden soll. Min= 2 / Max = 8. Wählt man z.B. 3, gibt es die Farben 1,2,3.

Freie Wahl der Stellen eines Codes

- Es ist möglich, zu Beginn jedes Spiels festzulegen, wieviel Stellen der Code haben soll: 4 oder 5

2.4 Anforderungsänderung vom 21.6.2024

Online Modus

- Das Spiel soll darauf vorbereitet werden, auch mit anderen über das Internet gespielt zu werden.
- Erster Schritt dazu: Ein Rater (egal ob ein menschlicher Rater oder der Computer als Rater) bekommt seine Bewertungen über das Netz.
- Hierzu muss konfigurierbar sein, ob die Rolle des Codierers vom laufenden Programm oder einem anderen über das Internet erreichbaren Programm eingenommen wird.
- Hierzu muss es möglich sein, eine IP-Adresse und einen Port hinterlegen zu können.
- Soll ein über das Internet erreichbares Programm die Rolle des Codierers einnehmen, kommt auch der zu ratende Code von diesem Programm.

Regeln für den Datenaustausch zwischen Programm und externem Codierer-Bot

- Die Kommunikation erfolgt einzig über HTTP Post.
- Als Content-Type wird für die Kommunikation ausschliesslich application/json verwendet (JavaScript Object Notation).
- Natürlich abgesehen von Client- oder Server-Fehlern wie z.B. 404 (Not Found).
- Die JSON-Nachrichten haben für Requests genau dasselbe Format, wie für Responses.

3 Objektorientierte Analyse

3.1 Analyseprozess (OOA)

Im Rahmen der objektorientierten Analyse (OOA) werden die funktionalen und nichtfunktionalen Anforderungen an das System systematisch untersucht und modelliert. Ziel dieses Prozesses ist es, ein tieferes Verständnis für die Anforderungen zu gewinnen und eine klare und präzise Visualisierung der verschiedenen Systemaspekte zu ermöglichen.

Der Analyseprozess beginnt mit der Identifikation und Definition aller relevanten Use Cases. Diese Use Cases werden anhand des Cockburn Templates[?] dokumentiert, welches eine strukturierte und einheitliche Darstellung der Anwendungsfälle gewährleistet. Durch die detaillierte Beschreibung der Interaktionen zwischen den Akteuren und dem System konnten wir die funktionalen Anforderungen präzisieren und die Benutzerperspektive ausarbeiten.

Im weiteren Verlauf des Analyseprozesses wurden die identifizierten Use Cases verwendet, um ein statisches Objektmodell in Form eines Klassendiagramms anzufertigen. Dies umfasst die Modellierung von Klassen, deren Beziehungen zueinander sowie die Definition der wichtigsten Methoden und Attribute. Diese Use-Cases teilen sich auf in "must-haves" und "nice-to-haves". Wir legen im weiteren Verlauf des Projekts den meisten Fokus auf die must-haves, da diese für ein funktionales Programm zwingend erforderlich sind.

3.2 Use Case Diagramme nach Cockburn

Use Case #1 (must-have)	Spiel als Rater starten
Goal in Context	Der Spieler möchte ein neues Spiel "Super Superhirn" als Rater starten.
Preconditions	Anwendung wurde gestartet. Der Spieler befindet sich im Hauptmenü.
Success End Conditions	Das Spiel wird im Rater-Modus gestartet.
Failed End Conditions	Das Spiel startet gar nicht oder im falschen Modus.
Primary Actor	Spieler
Trigger	Konsolenanfrage des Spielers, ein Spiel als Rater starten zu wollen.

Description	**Step**	**User Action**	**System**
	1	Spieler sendet Konsolenbefehl zum Start des Spiels als Rater	System sendet Tutorialanfrage gemäß Use Case #8
	2	Spieler nimmt Tutorialanfrage an	System sendet Spieler zu Tutorialscreen für Rater
	3	Spieler sendet Anfrage, zurück zu navigieren	System sendet Spieler zum Spiel und startet dieses
Subvariations	**Step**	**User Action**	**System**
	2	Spieler lehnt Tutorialanfrage ab	System sendet Spieler zum Spiel und startet dieses

Use Case #2 (must-have)	Spiel als Rater spielen
Goal in Context	Der Spieler möchte ein neues Spiel "Super Superhirn" als Rater spielen.
Preconditions	Der Spieler hat das Spiel gemäß Use Case #1 gestartet.
Success End Conditions	Der Spieler trägt seinen Namen in die Bestenliste ein.
Failed End Conditions	Spieler löst Fehler aus, der nicht durch das System behandelt wird.
Primary Actor	Spieler
Trigger	Success End Condition von Use Case #1

Description	**Step**	**User Action**	**System**
	1	/	System generiert Farbcode und wartet auf Rateversuch
	2	Spieler gibt einen Rateversuch ab	System vergleicht mit richtiger Kombination und gibt entsprechende Pins zurück
	3	Spieler gibt weitere Rateversuche ab	System vergleicht mit richtiger Kombination und gibt entsprechende Pins zurück
	4	Spieler errät richtige Kombination	System fragt nach Namen
	5	Spieler trägt Namen ein	System trägt Namen, Zuganzahl und Zeit in Bestenliste ein
Subvariations	**Step**	**User Action**	**System**
	4	Spieler rät eine falsche Kombination und hat keine Züge mehr übrig	System bricht das Spiel ab und zeigt die richtige Kombination

Use Case #3 (must-have)	Spiel als Kodierer starten
Goal in Context	Der Spieler möchte ein neues Spiel "Super Superhirn" als Kodierer starten.
Preconditions	Anwendung wurde gestartet. Der Spieler befindet sich im Hauptmenü.
Success End Conditions	Das Spiel wird im Kodierer-Modus gestartet.
Failed End Conditions	Das Spiel startet gar nicht oder im falschen Modus.
Primary Actor	Spieler
Trigger	Konsolenanfrage des Spielers, ein Spiel als Kodierer starten zu wollen.

Description	**Step**	**User Action**	**System**
	1	Spieler sendet Konsolenbefehl zum Start des Spiels als Kodierer	System sendet Tutorialanfrage gemäß Use Case #8
	2	Spieler nimmt Tutorialanfrage an	System sendet Spieler zu Tutorialscreen für Kodierer
	3	Spieler sendet Anfrage, zurück zu navigieren	System sendet Spieler zum Spiel und startet dieses
Subvariations	**Step**	**User Action**	**System**
	2	Spieler lehnt Tutorialanfrage ab	System sendet Spieler zum Spiel und startet dieses

Use Case #4 (must-have)	Spiel als Kodierer spielen
Goal in Context	Der Spieler möchte ein neues Spiel "Super Superhirn" als Kodierer spielen.
Preconditions	Das Spiel wurde gemäß Use Case #3 gestartet.
Success End Conditions	System errät richtige Kombination oder bricht das Spiel ab.
Failed End Conditions	Spieler löst Fehler aus, der nicht durch das System behandelt wird.
Primary Actor	Spieler
Trigger	Success End Condition von Use Case #3

Description	**Step**	**User Action**	**System**
	1	/	System wartet auf Eingabe des Farbcodes
	2	Spieler stellt Farbcode ein	System gibt Rateversuch ab
	3	Spieler gibt automatisch entsprechende Pins zurück	System gibt neue Rateversuche ab
	4	Spieler gibt automatisch entsprechende Pins zurück	System errät richtige Kombination
Subvariations	**Step**	**User Action**	**System**
	4	Es ist der letzte Zug und der Spieler gibt automatisch eine Pinkombination zurück, die nicht dem richtigen Farbcode entspricht	Das System hat es nicht geschafft, die Kombination rechtzeitig zu erraten, und bricht das Spiel ab

Use Case #5 (must-have)	Bestenliste einsehen
Goal in Context	Der Spieler möchte sich die Bestenliste bereits gespielter Spiele anschauen.
Preconditions	Anwendung wurde gestartet. Der Spieler befindet sich im Hauptmenü.
Success End Conditions	Die Bestenliste wird angezeigt.
Failed End Conditions	Die Bestenliste wird nicht angezeigt.
Primary Actor	Spieler
Trigger	Konsolenanfrage des Spielers, die Bestenliste einsehen zu wollen.

	Step	**User Action**	**System**
Description	1	Spieler sendet Konsolenbefehl zum Einsehen der Bestenliste	System öffnet Bestenliste

Use Case #6 (must-have)	Spielanleitung einsehen
Goal in Context	Der Spieler möchte sich die Spielanleitung anschauen.
Preconditions	Anwendung wurde gestartet. Der Spieler befindet sich im Hauptmenü.
Success End Conditions	Die Spielanleitung wird angezeigt.
Failed End Conditions	Die Spielanleitung wird nicht angezeigt.
Primary Actor	Spieler
Trigger	Konsolenanfrage des Spielers, die Spielanleitung einsehen zu wollen.

	Step	**User Action**	**System**
Description	1	Spieler sendet Konsolenbefehl zum Einsehen der Spielanleitung	System öffnet Spielanleitung

Use Case #7 (must-have)	Zum Hauptmenü navigieren
Goal in Context	Der Spieler möchte zurück zum Hauptmenü, ausgehend von Spielanleitung, Bestenliste, Spielende.
Preconditions	Der Spieler befindet sich im Fenster der Spielanleitung, der Bestenliste oder er befindet sich am Ende eines Spiels (Success End Condition von Use Case #4 bzw. #2).
Success End Conditions	Das Hauptmenü wird angezeigt.
Failed End Conditions	Das Hauptmenü wird nicht angezeigt.
Primary Actor	Spieler
Trigger	Konsolenanfrage des Spielers, zurück zum Hauptmenü navigieren zu wollen.

	Step	**User Action**	**System**
Description	1	Spieler sendet Konsolenbefehl zum Navigieren zum Hauptmenü	System öffnet Hauptmenü

Use Case #8 (must-have)	Tutorialanfrage anzeigen
Goal in Context	Hat der Spieler sich für eine Rolle entschieden, so kriegt er eine Anfrage, ob er das Tutorial sehen will.
Preconditions	Der Spieler hat sich für eine Rolle entschieden.
Success End Conditions	Die Tutorialanfrage wird angezeigt.
Failed End Conditions	Die Tutorialanfrage wird nicht angezeigt.
Primary Actor	Spieler
Trigger	Der Spieler hat sich für eine Rolle entschieden.

	Step	**User Action**	**System**
Description	1	/	System sendet Tutorialanfrage

Use Case #9 (must-have)	Hauptmenü anzeigen		
Goal in Context	Sobald der Spieler die Anwendung startet, wird das Hauptmenü angezeigt.		
Preconditions	Der Spieler hat die Anwendung gestartet.		
Success End Conditions	Das Hauptmenü wird angezeigt.		
Failed End Conditions	Das Hauptmenü wird nicht angezeigt.		
Primary Actor	Spieler		
Trigger	Der Spieler hat die Anwendung gestartet.		
Description	**Step**	**User Action**	**System**
	1	Spieler startet Anwendung	System zeigt Spieler Hauptmenü an

Use Case #10 (nice-to-have)	Spielanleitung in Spiel anzeigen		
Goal in Context	Der Spieler soll im Spiel die Anleitung für seine Rolle aufrufen können.		
Preconditions	Der Spieler befindet sich im Spiel.		
Success End Conditions	Die Spielanleitung für die entsprechende Rolle des Spielers wird angezeigt.		
Failed End Conditions	Es wird die falsche oder keine Spielanleitung angezeigt.		
Primary Actor	Spieler		
Trigger	Der Spieler hat im Spiel eine Anfrage an das System gesendet, das Tutorial für seine Rolle einzusehen.		
Description	**Step**	**User Action**	**System**
	1	Der Spieler hat im Spiel eine Anfrage an das System gesendet, das Tutorial für seine Rolle einzusehen	System pausiert das Spiel und zeigt Spieler die Spielanleitung an

Use Case #11 (nice-to-have)	Von Spielanleitung zurück zum Spiel navigieren
Goal in Context	Der Spieler soll nach Success End Condition von Use Cse #10 wieder zurück zum Spiel navigieren können.
Preconditions	Der Spieler hat Success End Condition von Use Case #10 erreicht und ist nicht zu einem anderen Screen navigiert.
Success End Conditions	Das Spiel wird entpausiert und dem Spieler wieder angezeigt.
Failed End Conditions	Das Spiel wird dem Spieler nicht angezeigt oder das Spiel wird nicht entpausiert.
Primary Actor	Spieler
Trigger	Der Spieler hat in der Spielanleitung seiner Rolle eine Anfrage an das System gesendet, zurück zum Spiel zu navigieren.

	Step	**User Action**	**System**
Description	1	Der Spieler hat in der Spielanleitung seiner Rolle eine Anfrage an das System gesendet, zurück zum Spiel zu navigieren	System entpausiert das Spiel und zeigt Spieler das Spiel an

3.3 Klassendiagramm

Als nächster Schritt gilt es, ein Klassendiagramm zu entwerfen. Den Anforderungen entsprechend haben wir für diesen Grobentwurf die folgenden Klassen identifiziert und anschließend in Form eines Klassendiagramms in Beziehung gestellt:

- Spieler
- Spiel
- Hauptmenü
- Zug
- ZugHistory
- Code
- Bewertung
- Leaderboard
- Eintrag
- Tutorial

Der Spieler kann über das Hauptmenü ein Spiel starten. Dieses enthält den Code, die Zughistorie und bietet die Möglichkeit, ein Tutorial anzuzeigen. Weiterhin wird nach dem Spiel ggf. das Ergebnis im Leaderboard in Form eines Eintrags gespeichert, worauf auch vom Hauptmenü zugegriffen werden kann.

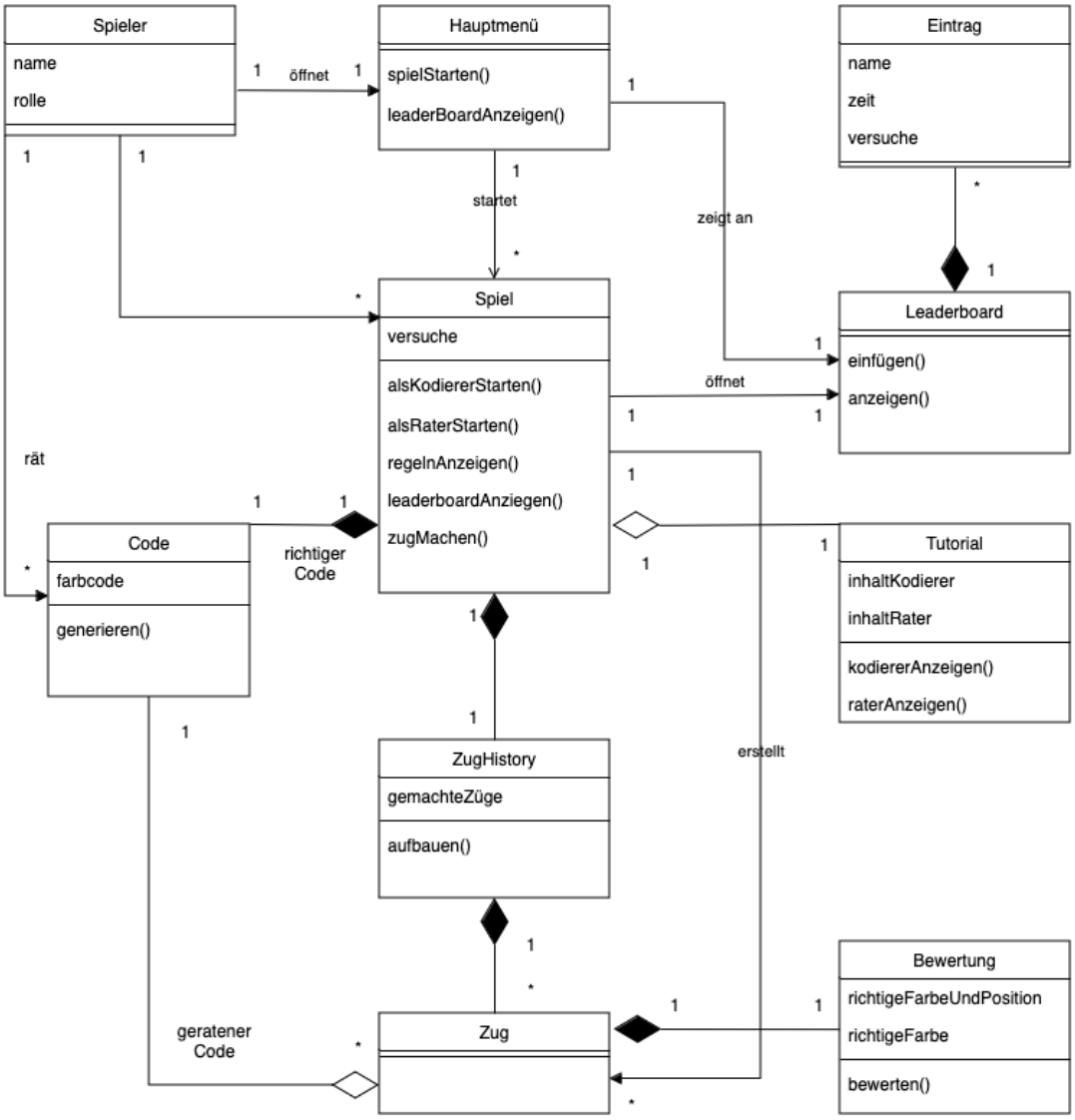


Figure 2: Klassendiagramm

3.4 Sequenzdiagramm

Weiterhin haben wir ein Sequenzdiagramm entworfen, um den Use Case #1 (Spiel als Rater starten) im Kontext des Klassendiagramms genauer darzustellen.
Hier sind die Lebenslinien der Klassen erkennbar. Während Klassen wie das Tutorial erst durch Nutzeranfrage erstellt und anschließend wieder geschlossen werden, existiert das Leaderboard dauerhaft.

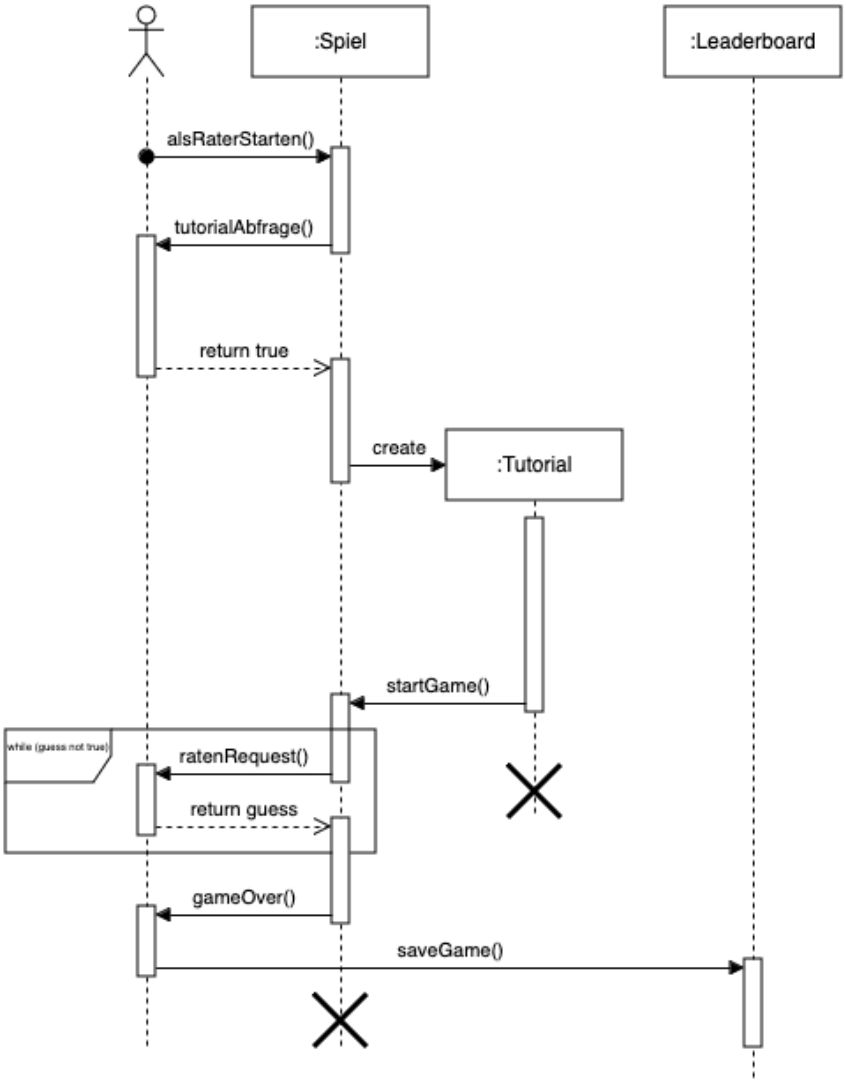


Figure 3: Sequenzdiagramm

3.5 Zustandsdiagramm

Zuletzt haben wir ebenfalls ein Zustandsdiagramm entworfen, um die Navigation zwischen den Klassen in Verbindung mit deren Zuständen darstellen zu können.

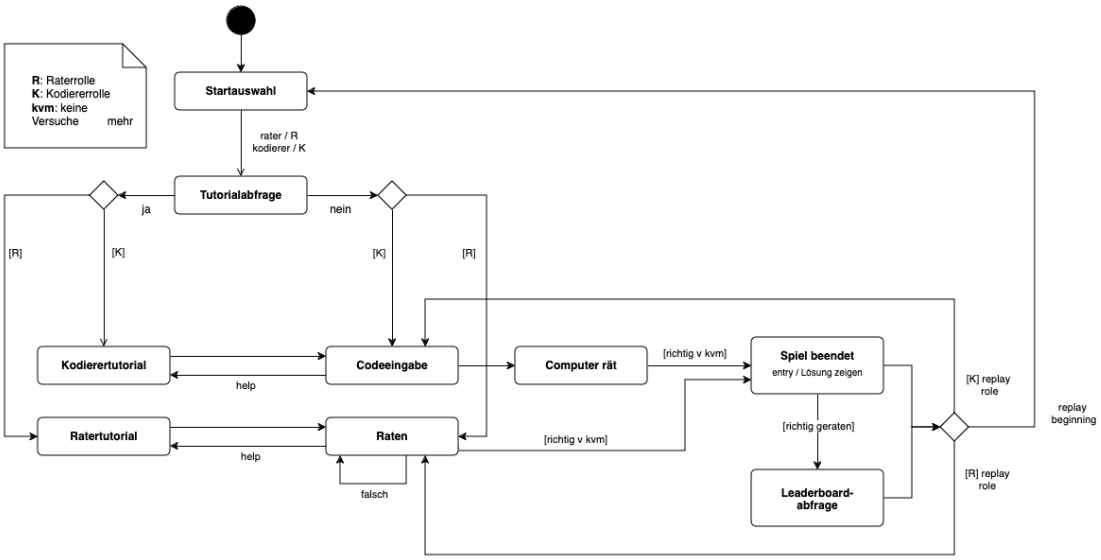


Figure 4: Zustandsdiagramm

Hierbei lässt sich erkennen, wie die Navigation mit den verschiedenen Rollen funktioniert und wie das Spiel beendet wird.

3.6 Qualitätsmanagement OOA

Die Analysephase eines Softwareprojekts ist entscheidend für das spätere Gelingen des gesamten Projekts. In dieser Phase werden die Anforderungen ermittelt, dokumentiert und analysiert. Ein effizientes Qualitätsmanagement sorgt dafür, dass die erfassten Anforderungen vollständig, konsistent und eindeutig sind. Im Rahmen unseres Projekts "Mastermind" wurden verschiedene Maßnahmen zur Sicherstellung der Qualität in der Analysephase implementiert:

Anforderungsanalyse und -dokumentation: Wir haben umfangreiche Anforderungen sowohl funktionaler als auch nicht-funktionaler Natur gesammelt. Diese wurden in einem strukturierten Prozess mit dem Projektsponsor besprochen und verifiziert.

Use Case Analyse: Um die Anforderungen besser zu verstehen und zu veranschaulichen, wurde eine Use Case Analyse nach Cockburn durchgeführt. Diese Methode half dabei, die Interaktionen zwischen dem System und seinen Benutzern zu definieren und sicherzustellen, dass alle relevanten Szenarien abgedeckt sind.

Modellierung und Diagramme: Für die Analyse und Dokumentation wurden verschiedene UML-Diagramme erstellt, einschließlich Klassendiagrammen, Sequenzdiagrammen und Zustandsdiagrammen. Damit konnten wir die Struktur und das Verhalten des Systems gut visualisieren und fehlende Klassen identifizieren.

Review und Validierung: Alle erstellten Dokumente und Diagramme wurden regelmäßigen Reviews unterzogen, um sie mit den Anforderungen abzugleichen. Diese Reviews wurden

in den wöchentlichen Meetings durchgeführt, an denen das gesamte Team beteiligt war. Feedback aus diesen Meetings wurde systematisch erfasst und in die weitere Entwicklung integriert.

Kontinuierliche Verbesserung: Der Qualitätsmanagementprozess war flexibel gestaltet, sodass Anpassungen und Verbesserungen basierend auf den Erkenntnissen aus den Reviews und den Fortschritten im Projektverlauf vorgenommen werden konnten.

Konstruktive Qualitätssicherung: Zur konstruktiven Qualitätssicherung wurde das Prozessmodell Kanban gewählt. Kanban ermöglicht eine flexible und kontinuierliche Anpassung der Aufgaben und Prozesse, was besonders in der Analysephase von Vorteil ist.

Durch diese Maßnahmen konnten wir nach der Analysephase eine solide Basis für die nachfolgenden Phasen bilden.

4 Objektorientiertes Design

In diesem Abschnitt wird das Klassendiagramm der OOA zu einem Entwurf für die Implementierung weiterentwickelt.

4.1 Beschreibung des Designprozesses

Die Klassen Spiel, Leaderboard, Tutorial und Hauptmenü werden in Anlehnung an das MVC Pattern, welches in der QS bereits vorausgewählt wurde, jeweils in View, Control und Model wie Observer (falls die View einen Zustandswechsel erfahren kann) aufgespalten. Die Klasse Spieler fällt durch MVC zur Hälfte weg. Für einen menschlichen Spieler gibt es für die Views Input-Listener, die die Spieler Methoden handhaben, indem sie die Logik in der jeweiligen Control der View aufrufen. Dafür gibt es aber Bot Klassen für den Rater und den Kodierer, auf die die Spiellogik zugreifen kann. Die Eintrag Klasse aus dem alten Klassendiagramm wird praktisch übernommen. Der Code ist keine eigene Klasse, sondern ist nun eine Variable in der Control des Spiels. Die Zug History befindet sich nun im Model des Spiels und beinhaltet bis zu 12 Züge, jeweils bestehend aus Rateversuchen und dazugeörigen Evaluationen. Abgesehen davon gibt es nun auch die angeforderte Persistenz für das Leaderboard, in Form von LeaderboardWriter und LeaderboardReader. Um zwischen Views zu navigieren wurde das Klassendiagramm außerdem um einen NavigationManager erweitert. In dem Abschnitt (4.3) wird das Diagramm aufgezeigt und es wird noch genauer auf die komplexeren Bestandteile des Diagramms eingegangen. Dort wird außerdem die geplante Logik des Spielablaufs beschrieben, für beide Rollen.

4.2 UI Mockups

UI Mockups sind ein essenzieller Bestandteil des Designprozesses und dienen als visuelle Blaupause für die Benutzeroberfläche der Anwendung. Sie helfen dabei, das Layout und die Interaktionsmöglichkeiten frühzeitig zu visualisieren und geben dem Entwicklungsteam sowie dem Sponsor eine konkretere Vorstellung von der zu entwickelnden Software.

Zu Beginn haben für die zentralen Sichten Hauptmenü und Spielfeld UI Mockups erstellt. Da sich diese als sehr nützlich erwiesen, wurden iterativ weitere Mockups für das Leaderboard und das Tutorial ergänzt. Dabei wurden folgende Schwerpunkte gesetzt:

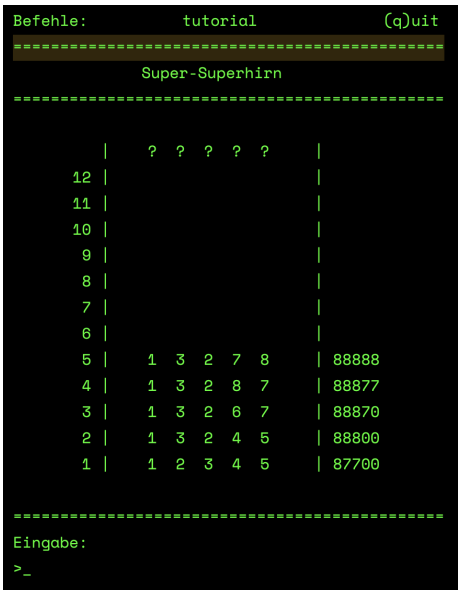
Klarheit und Einfachheit: Das Interface wurde so gestaltet, dass es für den Benutzer intuitiv, übersichtlich und leicht verständlich ist. Dabei wurde ein minimalistischer Ansatz gewählt. Die UI-Elemente sind auf das nötigste reduziert und alle relevanten Informationen und Nachrichten kurz und prägnant formuliert. Die Fenstergröße von nur 56x27 (Höhe x Breite) Zeichen folgt dem schlanken Design.

Benutzerfreundlichkeit: Durch die Verwendung von UI Mockups konnten verschiedene Designalternativen getestet und bewertet werden. Dies half dabei, eine Benutzeroberfläche zu entwickeln, die sowohl funktional als auch ästhetisch ansprechend ist. Feedback von allen Teammitgliedern wurde gesammelt und in die Mockups integriert, um die Benutzerfreundlichkeit weiter zu optimieren.

Kontinuierliche Verbesserung: Die UI Mockups wurden in einem iterativen Prozess entwickelt, bei dem regelmäßig Anpassungen vorgenommen wurden, basierend auf dem Feedback des Teams und der Stakeholder. Dies stellte sicher, dass das endgültige Design den Anforderungen und Erwartungen aller Beteiligten entsprach. Die abgebildeten UIs zeigen einen der ersten Prototypen.

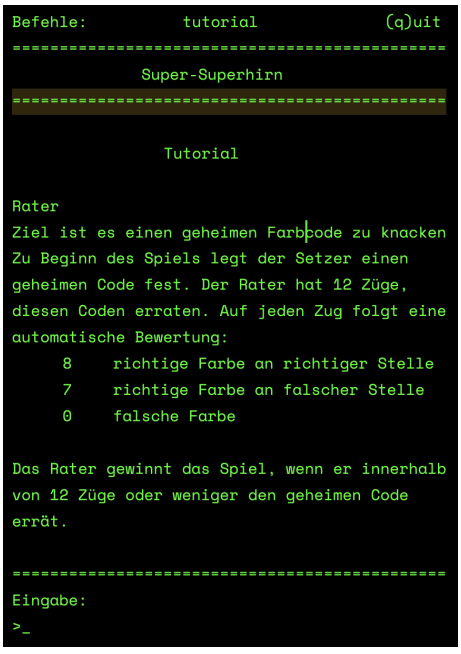
Durch den Einsatz der UI Mockups konnten wir den Grundstein für eine gute Nutzererfahrung und Anwenderfreundlickeit schaffen.

(a) Hauptmenü



(b) Spielfeld

(c) Leaderboard



(d) Tutorial

Figure 5: UI Mockups

4.3 Klassendiagramm

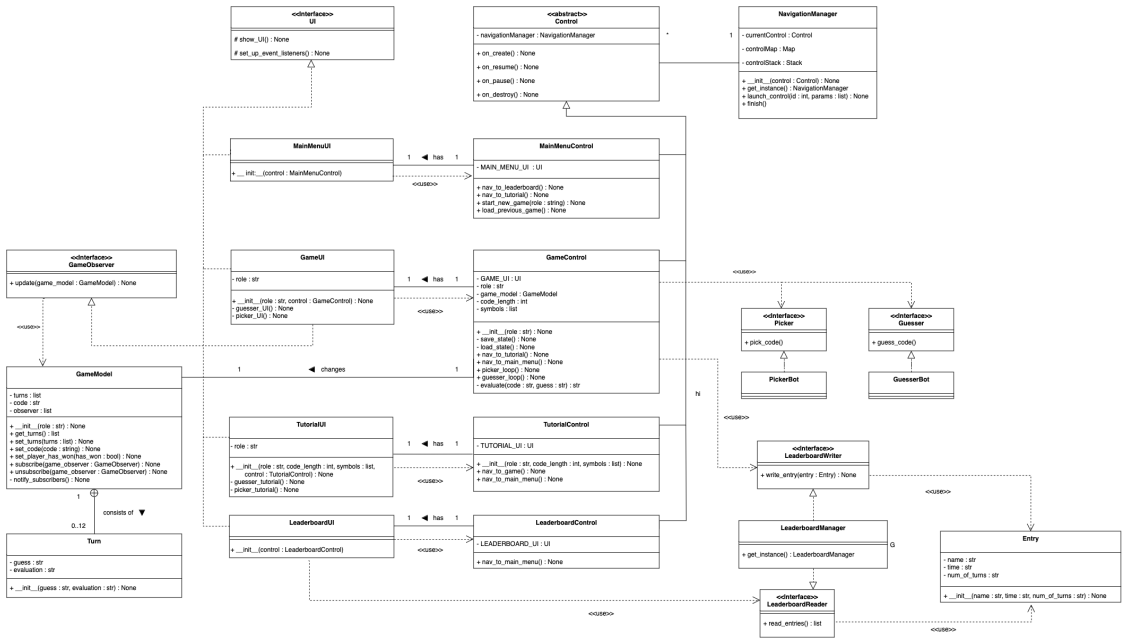


Figure 6: Klassendiagramm der OOD-Phase. Vergrößerte Ansicht im Anhang verfügbar (8.5).

NavigationManager hat get_instance(), um nur eine einzige Instanz der eigenen Klasse im Programm zurückzugeben. navigate_to() ändert die momentane Control. Dadurch wird auch eine Instanz der entsprechenden UI erstellt, wobei die Methoden show_UI() und set_up_event_listeners() der UI aufgerufen werden. show_UI() soll in unserem Fall die Konsole clearen und neu zeichnen. Die Events werden nach set_up_event_listeners() von der Control gehandhabt.

TutorialUI erhält von TutorialControl die Rolle des Spielers. Sie kann der Rolle des Kodierers oder der des Raters entsprechen. Das Tutorial hat dann Methoden, um unterschiedliche Views abhängig von der Rolle anzuzeigen. Sollte keine Rolle angegeben werden, so werden beide Tutorials untereinander angezeigt. Bei den Eventlisteners hängen die Navigationsmöglichkeiten auch von der Rolle ab.

Spielt der Spieler als Kodierer, so wird über den Konstruktor der GameUI die Methode picker_loop() von GameControl aufgerufen. In dieser muss der Spieler dazu aufgefordert werden, einen Code anzugeben. Dies muss mit der set_code() Methode des GameModels geschehen, damit die UI daraufhin aktualisiert wird. Daraufhin wird ein Thread gestartet, wo der Guesser Bot kontinuierlich rät und bewertet wird. Die Züge des Bots werden auch an das GameModel gegeben, um die UI zu informieren und zu aktualisieren. Letztendlich gewinnt der Bot oder er verliert. Das wird auch an das GameModel gegeben, über set_player_has_won(), um die UI dazu aufzufordern, die richtige EndMessage anzuzeigen. Danach wird picker_loop() beendet.

Spielt der Spieler als Rater, so wird über den Konstruktor der GameUI die Methode guesser_loop() von GameControl aufgerufen. In dieser setzt ein Picker Bot den zu erratenden Code, was das GameModel und darüber auch die UI beinflusst. Der Spieler wird dann mehrfach dazu angehalten, zu raten oder woanders hinzunavigieren. Sobald das

Spiel endet, wird geprüft, ob der Spieler gewonnen hat und eine entsprechende Message über die UI ausgegeben, nachdem das Model angepasst wurde.

Navigation, die das Spiel unterbricht, sorgt dafür, dass das Model und andere relevante Objekte und Variablen in einem File gespeichert werden, so dass das Spiel nach Rücknavigation rekonstruiert werden kann.

Bei Beginn eines neuen Spiels durch start_new_game() in MainMenuControl oder durch Nutzeraufforderung nach Ende des Spiels, wird das File, das den Spielstand beinhaltet, geleert.

4.4 Zustandsdiagramm

Figure 7: Zustandsdiagramm einer willkürlichen Control-Klasse mit den von dieser implementierten Methoden.

Dieses Diagramm verdeutlicht die Funktionweise der Navigation in unserem Programm. Jede Control-Klasse muss diese Methoden beinhalten, damit der Navigation Manager die Controls kreieren, dem Stack hinzufügen und aus dem Stack wieder entfernen kann. Ein beispielhafter Ablauf dieser Funktionalität ist in dem nachfolgenden Sequenzdiagramm zu erkennen.

4.5 Sequenzdiagramm

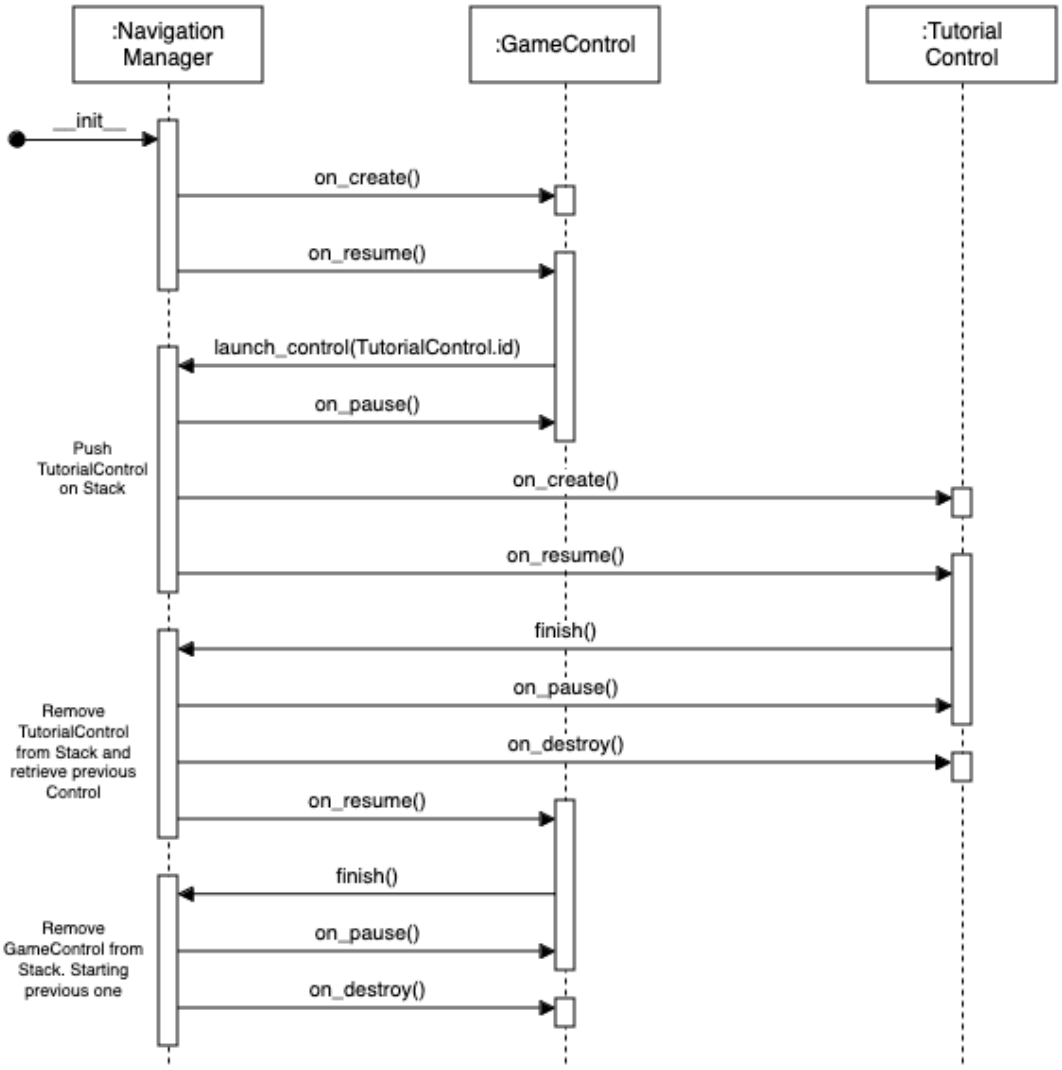


Figure 8: Sequenzdiagramm eines Tutorial-Aufrufs

4.6 Architektur

Mastermind folgt dem Publisher-Subscriber Architekturstil für unbuffered Communication. Das ermöglicht eine lose Kopplung zwischen den verschiedenen Komponenten der Anwendung durch ereignisgesteuerte Kommunikation. Hierbei agieren Publisher - in unserem Fall das Model - die Ereignisse oder Nachrichten erzeugen, und Subscriber - hier die View -, die auf diese Ereignisse hören und entsprechend reagieren können. Diese Architektur unterstützt Flexibilität, Erweiterbarkeit und gute Wartbarkeit, da Änderungen an einem Teil der Anwendung minimale Auswirkungen auf andere Teile haben.

4.7 Entwurfsmuster

In Mastermind kommen hauptsächlich zwei Entwurfsmuster zum Einsatz: MVC und Singleton. Mit der zweiten Änderung der Anforderungen wurde die Software um eine Client-

Server Struktur erweitert.

Das MVC-Entwurfsmuster wird angewendet, um eine klare Trennung der Verantwortlichkeiten zu erreichen:

Separation of Concerns: MVC trennt die Anwendung in drei Hauptkomponenten: das Model, die View und den Controller. Dadurch werden Datenverarbeitung, Benutzeroberfläche und Steuerungslogik voneinander entkoppelt.

Erleichterte Wartung und Erweiterung: Durch die klare Struktur ist es einfacher, Änderungen vorzunehmen oder neue Funktionen hinzuzufügen, ohne andere Teile der Anwendung zu beeinträchtigen.

Wiederverwendbarkeit und Flexibilität: Die Wiederverwendbarkeit von Komponenten wird verbessert, da jede Schicht unabhängig entwickelt und getestet werden kann.

Parallelentwicklung: Entwickler können gleichzeitig an verschiedenen Teilen der Anwendung arbeiten, da die Trennung der Verantwortlichkeiten Kollisionen reduziert.

Bessere Testbarkeit: Jede Komponente (Model, View, Controller) kann separat getestet werden, was zu effizienteren Tests und höherer Codequalität führt.

Das Singleton-Entwurfsmuster wird verwendet, um sicherzustellen, dass nur eine Instanz einer Klasse existiert:

Kontrollierter Zugriff auf eine einzige Instanz: Singleton ermöglicht den Zugriff auf eine globale Instanz einer Klasse, was nützlich ist, wenn genau eine Instanz einer Ressource benötigt wird.

Globale Zugänglichkeit: Die Instanz kann überall in der Anwendung abgerufen werden, was den Zugriff auf gemeinsame Ressourcen oder Konfigurationen erleichtert.

Diese Entwurfsmuster unterstützen die Architektur von Mastermind, indem sie klare Strukturen und effiziente Lösungen für spezifische Designanforderungen bereitstellen.

4.8 Qualitätsmanagement

Nach der Designphase sollen die geplante Architektur und das Design die Anforderungen vollständig erfüllen und eine solide Grundlage für die Implementierung bieten. Ein strukturiertes und methodisches Vorgehen ist hier besonders wichtig, da Fehlentscheidungen im Design zu einem späteren Zeitpunkt schwer zu korrigieren sind. Folgende Maßnahmen wurden umgesetzt:

Architekturreviews: Die entworfene Softwarearchitektur wurde in 2 Teammeetings diskutiert. Diese Reviews halfen dabei, Schwachstellen in der Architektur zu identifizieren. Außerdem hatte dadurch jedes Teammitglied ein umfassendes Verständnis des entwickelten Designs.

Design Patterns: Bei der Entwicklung der Software wurden bewährte Design Patterns verwendet. Diese Muster bieten erprobte Lösungen für häufig auftretende Probleme und tragen zur Robustheit und Flexibilität des Designs bei. Beispiele für eingesetzte Muster sind das Singleton, MVC und Observer Pattern.

UI-Mockups: Zur Validierung des UI-Designs wurden mehrere Mockups erstellt. Dadurch konnten wir vom Projektsponsor wertvolles Feedback einholen. Zum Beispiel konnte ermittelt werden, dass der Sponsor eine einheitliche Größe des Konsolenfensters wünscht.

In iterativen Meetings konnten wir diese neuen Anforderungen agil umsetzen und die Nutzerfrendlichkeit erhöhen.

Dokumentation und UML-Diagramme: Zur Dokumentation wurden UML-Diagramme angefertigt und wichtige Designentscheidungen schriftlich festgehalten. Diese Dokumentation dient als Referenz für die Implementierungsphase und als Nachschlagwerk für Wartung und Erweiterungen.

5 Implementierung

Im Schritt vom objektorientierten Design zur tatsächlichen Implementierung haben am fundamentalen Aufbau des Programms kaum Änderungen stattgefunden. Die Grundlegende Struktur bildet nach wie vor das MVC-Pattern mit Integration der Observer-Struktur zwischen Klassen der UI und der Models - wie in Abbildung (9) sichtbar.

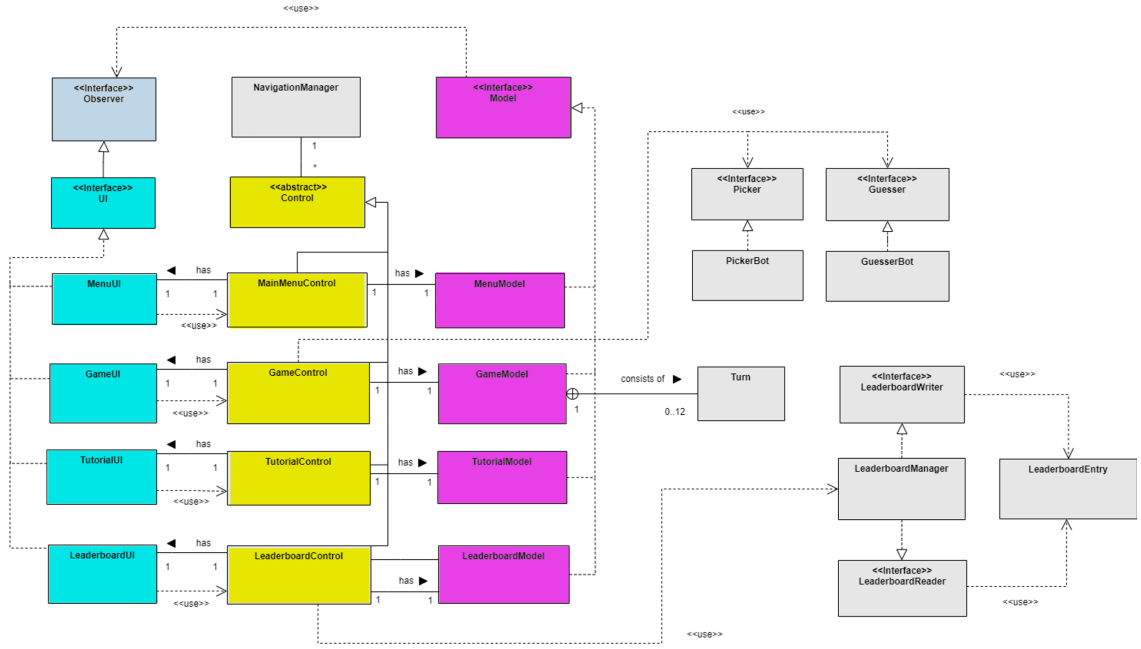


Figure 9: Klassendiagramm des fertigen Projektes. Klassen des UI in Cyan, der Controller in Gelb und der Models in Magenta. In der ersten Reihe wird deutlich, wie das Observer-Muster zwischen Model und UI umgesetzt wird.

Die vollzogenen Änderungen betreffen einzelne Methoden der Klassenstruktur. So hat sich die Struktur des NavigationManagers von den in Abschnitt (4.3) beschrieben Methoden wie folgt verändert.
Außerdem wurden die User-Interfaces fortlaufend den neuen Anforderungen angepasst. Ausschnitte der finalen Interfaces finden sich ebenfalls im Anhang (8.4).

5.1 NavigationManager-Klasse

Eine navigate_to()-Methode hat es nicht in die finale Version des Programms geschafft. Lediglich die auch im Sequenzdiagramm (8) sichtbaren Methoden "launch_control(control_id, params)", "finish(params)" und die ergänzte Methode "to_first_control" finden sich im fertigen Programm wieder. Letztere ergänzt den NavigationManager um die Funktion mit einem Funktionsaufruf zur ersten Control zurückzukehren und diese zu aktivieren nachdem alle anderen Control-Klassen vom Stack entfernt werden. Zudem haben die beiden zuvor schon vorhandenen Methoden Argumente erhalten, die es (launch_control) ermöglichen einer Control beim Start Parameter zu übergeben sowie (finish) eine indirekte Kommunikation zwischen den Control-Klassen realisieren. Wird eine Control-Instanz über den Aufruf der finish-Methode vom Stack entfernt, so werden potenziell übergeben Argumente beim Aktivieren der schlafenden Control auf dem Stack darunter übergeben.

5.2 Evaluation

Die Evaluationsmethode evaluate() in game_logic.py nimmt die Parameter guess: int und actual_code: int und gibt ein Objekt der Klasse Turn zurück, welcher den Rateversuch und die Evaluation enthält, also die Anzahl der schwarzen und weißen Pins, jeweils.

Zuerst wird gezählt, wie oft die Farben in dem zu ratenden Code vorkommen, indem ein dictionary mit Farben und der dazugehörigen Anzahl im zu ratenden Code erstellt wird. Dann wird die Anzahl der schwarzen Pins berechnet, indem über den zu ratenden Code iteriert wird und an jeder Stelle des Codes überprüft wird, ob der Rateversuch an der selben Stelle gleich ist. Ist dies der Fall, wird die Anzahl der schwarzen Pins inkrementiert und wir merken uns in black_pin_set_at die Positionen, für die schon schwarze Pins gesetzt wurden, damit wir diese nicht nochmal für weiße Pins mitzählen. Dann wird nochmal über den zu ratenden Code iteriert, um die weißen Pins zu berechnen. Dafür wird geprüft, ob für die Position nicht schon ein schwarzer Pin gesetzt wurde und ob die Farbe an der entsprechenden Position bei dem Rateversuch noch im zu ratenden Code an einer Stelle vorkommt. Ist dies der Fall, wird die Anzahl der weißen Pins inkrementiert und die verbleibende Anzahl der geratenen Farbe im zu ratenden Code wird dekrementiert. Letztlich wird der daraus resultierende Turn zurückgegeben.

Die Methode wurde bezogen auf Randbedingungen und unterschiedlichste Kombinationen zwischen aufzutrenden weißen und schwarzen Pins umfangreich getestet. Außerdem wurde getestet, was passiert, wenn die schwarzen Pins sich am Ende des Rateversuchs befinden und ob sich wiederholende Farben zu Fehlerfällen führen.

5.3 Server

Zur Evaluation von Rateversuchen anderer Geräte wurde ein Flask Server gebaut. Flask ist ein Express ähnliches Framework mithilfe dessen sich leicht eine API für das handhaben von HTTP Methoden auf unterschiedlichen Routen erstellen lässt. Die einzige Route ist '/'. Sie kann POST Requests abfangen und bearbeitet diese mit der evaluate() Methode des Evaluationsservers. Dies ist in add_routes() definiert. Die evaluate() Methode selbst nimmt das JSON in dem gesendeten request auf und extrahiert die relevanten Daten davon heraus. Im Fall von fehlenden Feldern wird der Statuscode 400 mit der Fehlernachricht "error: Missing field" mit dem entsprechenden Schlüssel des fehlenden Feldes zurückgesendet. Andernfalls wird geprüft, ob es sich um ein neues Spiel handelt, also ob die gesendete gameid 0 ist. Dann wird das Spiel mit der Methode new_game() mit einer entsprechenden gameid für das Spiel ausgestattet und das Spiel erhält einen zu ratenden Code mit dem in __setter_bot gesetzten Bot. Es wird ein Dictionary über die Spiele und zugehörige zu ratende Codes in __actual_code_dict geführt, so dass mehrere Spiele quasiparallel laufen können. Ist die gameid nicht 0, wird die gesendete gameid übernommen. Der Zug des Spielers wird dann anhand des für die gameid hinterlegten Codes mit der Evaluationsmethode aus game_logic.py evaluiert. Das Ergebnis wird zusammen mit den anderen Feldern wieder zurück an den Client geschickt, mit Statuscode 200, als JSON. Der Server läuft mit self.app.run(host='0.0.0.0', port=5000) auf allen privaten und öffentlichen IP Adressen der lokalen Maschine, sowie auf dem local host unter Port 5000.

Der Server wurde über Postman umfangreich getestet. Postman ist eine Software, die es ermöglicht, HTTP Methoden an Endpunkte zu schicken und die Antworten zurückzubekommen.

5.4 Rater-Bot

Für die Implementierung eines automatischen Rater-Bots haben wir uns an verschiedenen bestehenden Strategien zur Lösung des ursprünglichen Spiels "Mastermind" orientiert und so erweitert, dass diese mit variablen Längen der Codes und Anzahl an Farben umgehen können.
Letztendlich entstanden ist eine Version der least-worst-case-Strategie. Bei dieser wird vor jedem Rateversuch für jede noch mögliche Kombination überprüft, wie groß die Menge der im Anschluss bestehenden Kombinationen gänzlich ohne Überschneidung mit dem potenziellen Rateversuch wäre. Es wird als Rateversuch letztendlich die Kombination gewählt, für die die beschriebene Menge (die worst cases) eine minimale Größe hat - daher "least-worst-case-Strategie".
Der Bot pflegt außerdem eine Liste aller nach wie vor möglichen Codes. Erhält dieser Feedback durch eine Evaluation des Rateversuchs, werden aus der Liste alle nun nicht mehr möglichen Kombinationen entfernt, sodass diese solange schrumpft, bis nur noch die richtige Kombination übrig geblieben ist.
Es ist außerdem möglich, für die Berechnung des nächsten Rateversuchs auch Kombinationen zuzulassen, bei denen es sich logisch nicht mehr um die Lösung handeln kann. Dies kann unter Umständen zu einem höheren Informationsgewinn durch die Evaluation führen, da solche "illegalen" Rateversuche unter Umständen eine größere Anzahl noch möglicher Codes auszuschließen vermögen.

6 Qualitätssicherung

Folgend der Qualitätssicherungsprozess und die Qualitätssicherungsmaßnahmen, die momentan im Projekt zum Einsatz kommen.

6.1 Qualitätssicherungsprozess

- Der Qualitätssicherungsprozess und dessen Maßnahmen werden dem ganzen Team mitgeteilt.
- Änderungen der QS geschehen möglicherweise nach Absprache nach Meetings.
- Es werden wöchentlich am Mittwoch Meetings durchgeführt, in denen verifiziert wird, ob die funktionalen und nicht-funktionalen Anforderungen erfüllt werden. An diesen Meetings sind alle im Team beteiligt. Diese beinhalten die Phasen: 1. Vorstellung der Projektergebnisse, 2. Probleme gemeinsam lösen oder in das Backlog schreiben, 3. Zuweisung der Tasks im Kanban Board, die zur nächsten Woche gelöst werden sollen.

6.2 Qualitätssicherungsmaßnahmen

Folgend werden die Qualitätssicherungsmaßnahemn für die relevanten Qualitätsmerkmale aufgelistet.

6.2.1 Richtigheit

- Es werden umfangreiche Tests für alle nicht-trivialen Methoden durchgeführt.
- Die Zwischenergebnisse werden regelmäßig dem gesamten Team vorgestellt. Das Team kann dann Einwände gegen Modelle, Implementierungen etc. einwerfen. Insbesondere wird vor jeder Präsentation geprüft, ob die Anforderungen des Kunden für die jeweilige Phase erfüllt worden sind.

6.2.2 Zuverlässigkeit

- Die Tests müssen eine umfangreiche Fehlerbehandlung und Randszenarien abdecken.

6.2.3 Änderbarkeit

- Es wird MVC oder ein ähnliches Pattern verwendet, um die UI austauschbar zu machen und die Anwendung im allgemeinen besser wartbar zu machen.
- Es werden wo immer sinnvoll entsprechende Design/Architecture Patterns und Architekturstile verwendet, um die Lesbarkeit zu erhöhen.
- Jede Klasse, Methode, Funktion und jedes Modul erhält aussagekräftige docstrings.
- Komplexe Codezeilen werden am besten vermieden oder so kommentiert, dass sie einfach verständlich werden.
- Gewünschte Parametertypen und Rückgabetypen werden mit dokumentiert, als Type Annotations.
- Jedes Modul sollte ein separates Problem lösen.
- Es wird sich an den PEP 8 Style Guide gehalten.

- Der explizitiste Code ist zu bevorzugen.
- Es wird nur eine Anweisung pro Codezeile ausgeführt.
- Ausnahmen oder das zurückgeben von None oder False sollte bei irregulären Ablauf so früh wie möglich geschehen.
- Statt dem austauschen zweier Werte über eine temporäre Variable sollten Tupel verwendet werden.
- Private Methoden werden vorne beim Namen mit Unterstrich gekennzeichnet.
- Ignorierung einer Variable erfolgt durch __.
- Erstellen einer Liste der Länge n mit selben Inhalt geschieht durch [gewünschter Inhalt] * n.
- Erstellen einer Liste der Länge n von Listen geschieht durch list = [[] for __ in range(n)].

6.2.4 Sonstige Maßnahmen

- Die Berechnungszeit beider Bots sollte eine halbe Sekunde pro Zug nicht überschreiten.
- Es wäre Sinnvoll, Personen außerhalb des Teams das Spiel vor der Abnahme spielen zu lassen, um die UI auf Verständlichkeit zu überprüfen.
- Insbesondere sollten Personen, die das Spiel noch nicht kennen, vor der Abnahme maximal mithilfe des Tutorials das Spiel verstehen.

7 Fazit

7.1 Beurteilung des Ergebnisses

Insgesamt haben wir ein zuverlässiges und robustes Programm entwickelt, was allen notwendigen Anforderungen entspricht. Es besteht allerdings auch weiterhin Raum zur Verbesserung, indem einige der optionalen Anforderungen umgesetzt werden, wie die Verwendung von Farben in der Anzeige anstatt Nummern oder die Implementierung von Flags für die Initialisierung des Spiels, um direkt von der Konsole aus einen bestimmten Spielmodus starten zu können.

7.2 Beurteilung des Prozesses

Das genutzte Kanban Board half immens bei der Visualisierung der Aufgaben und des Workflows. Es wurde auch sichtbar, wenn Aufgaben noch niemand Verantwortlichen hatten.

Bezogen auf den Qualitätssicherungsprozess wurden, anders als abgesprochen, Merges mit dem Main-Branch vor den Code Reviews durchgeführt. Der Code wurde stattdessen kontinuierlich erweitert und immer Mittwochs bei Erneuerungen reviewt. Zeittechnisch war es nicht möglich, umfangreiche automatisierte Tests zu schreiben. Stattdessen wurden Tests hauptsächlich adhoc und nur für komplexe und schwer manuell überprüfbare Funktionalitäten, wie der Evaluierung, geschrieben. Diese sind dann auch entspreched umfangreich, um Zuverlässigkeit zu gewährleisten. Die Präsentation der Zwischenergebnisse ist, wie die Review des neuen Codes, wöchentlich am Mittwoch geschehen. Nicht jede Klasse, Methode, Funktion oder Modul hat einen umfangreichen Docstring in der tatsächlichen Implementierung erhalten. Module und einfache Methoden wurden weggelassen. Allgemein ist der Code allerdings noch gut dokumentiert. Es gibt, anders als vereinbart, keine einheitliche Docstring-Konvention in der Implementierung. Type Annotations wurden allerdings gemacht. Jedes Modul löst ein seperates Problem und es wurde kein grober Verstoß gegen PEP 8 gefunden. Man hat sich auch an einheitliche Namenskonventionen und idiomatische Programmierung gehalten. Die Berechnungszeit der Bots ist ausreichend schnell für einen fluiden Spielfluss. Spieler außerhalb des Teams gab es zum Zeitpunkt dieser Ausarbeitung nicht. Das vorgegebene Pair Programming wurde in den Commit Nachrichten auf Github dokumentiert und mit fast keinen Ausnahmen durchgezogen. In einigen Fällen wurden inbesondere kleinere Fehlerkorrekturen eigenständig ohne Pair-Programming integriert. Größere Commits allerdings wurden immer in Pair-Programming entwickelt und dokumentiert.

7.3 Mögliche Verbesserungen

In Zukunft sollte es stärker auf die Einhaltung von Code Reviews vor Merges mit dem Main-Branch kommen, damit Qualitätsmakel sich weniger schnell in das Produkt einschleichen können. Außerdem sollte mehr Acht auf die Ausarbeitung zur Qualitätssicherung, zumindest bei Dingen wie gemeinsamen Konventionen für Kommentare, gelegt werden. Weiterhin war der wöchentliche Rhythmus der Meetings in Teilen nicht ausreichend. Es wäre sinnvoll gewesen, in kürzeren Abständen (Täglich oder alle 2 Tage) den akutellen Stand kurz zu klären und so Schwierigkeiten schneller erfassen zu können und den Wöchentlichen Termin für eine ausführlichere Planung und Diskussion beizubehalten. Ebenfalls ist wär es eine Überlegung Wert gewesen, nach Test-Driven Development vorzugehen. Teilweise waren die Tests nur eine Nachüberlegung, wohingegen dieses Vorgehen geholfen hätte.

8 Anhang

8.1 Fragenkatalog für das Interview mit dem Projektsponsor

1. Können Sie uns bitte Ihre Vision für das Endprodukt beschreiben?
2. Welche konkreten Funktionen oder Features erwarten Sie in der Mastermind-Implementierung?
3. Welche Zielgruppe oder Benutzer sollen mit diesem Spiel angesprochen werden?
4. Welche Plattformen sollen unterstützt werden (z.B. Desktop, Web, mobile)?
5. Sind zukünftige Erweiterungen oder Weiterentwicklungen des Spiels geplant?
6. Konkretisierung der Regeln? Sei die zu erratende Kombination `2256`
   - Ist bei einem Guess von `1122` die Antwort 2x weiß oder 1x weiß?
   - Ist bei einem Guess von `1123` die Antwort 1x weiß oder 2x weiß?
   - Ist bei einem Guess von `1221` die Antwort 1x schwarz, 1x weiß oder 1x schwarz?
7. Darf man an bestimmten Stellen Neuronale Netze oder andere Tools des Machinellen Lernens verwenden (Fragt sich natürlich wo die helfen können, aber falls man die braucht)
8. Soll es verschiedene Schwierigkeitsstufen geben oder soll das Programm immer die bestmögliche Kombination guessen / setzen?
   - keine verschiedenen Stufen!
   - es soll eine Unschärfe geben, also nicht immer in minimaler Anzahl (5) Guesses lösen.
9. Darf Codierer-Bot komplizierte Businesslogik enthalten oder soll die zu erratende Kombination vollständig zufällig sein?
10. Soll man die Anzahl der zu erratenden Stellen auswählen können (statt festen 5)?
11. Sollen beide Rollen an einem Gerät ausgeführt werden? / Sollen zwei Geräte über einen Server kommunizieren?
12. Müssen Highscores für verschiedene Player persistiert werden? Soll die Persistierung nur lokal sein? Soll danach gefragt werden, ob die Runde gespielt werden soll?
13. Soll es ein leaderboard geben?
14. Welche Informationen sollen persistiert werden (Name, Highscore, Zeit)?
15. Sollen die Spieler gespeichert werden? Frühere Spiele einsehen? (alle Guesses)
16. Soll es ein zeitliches Limit für jeden Zug / für das gesamte Spiel geben? Auch wenn das Programm guesst?
17. Wer soll Eingaben machen können? Nur User oder auch ein anderes Programm
18. Wird ein Login gefordert?
19. Welche Aktionen soll ein Spieler, der Codierer ist, machen können? (nur Eingabe der zu eratenden Konbination, oder noch weitere Parameter?)
20. Sollen falsche Eingaben wiederholt werden können? (vllt Warnung?)

21. Was sind die "Spielsteine"? Zahlen, Buchstaben oder Farben?
22. Gibt es Anforderungen bezüglich...
    - der maximalen Berechnungszeit?
    - des verwendeten Speicherplatzes?
    - weiterer Einschränkungen?

8.2 Ergebnisse des Interviews

Funktionale Anforderungen

- Single Player only - Ein Nutzer pro Maschine, Spieler 2 ist immer der Computer
- Möglichkeit zum Clearen des Spielfelds und Eingabe der Farben
- Auswahl des Spielmodus über Argumente, mit Nachfrage bei fehlenden Argumente
- Tutorial (nach der Spielmodus-Auswahl) mit den Spielregeln. Optional: Tutorial/Help als Parameter
- Spielstart über Parameter, mit Nachfrage bei fehlenden Parametern
- Auswahl der Rollen Rater und Codierer vor Spielbeginn
- Der Spielablauf soll automatisiert sein, das heißt der Codierer macht während des Spiels keine Eingabe. Die einzelnen Schritte sollen nachvollziehbar ausgegeben werden. Zeitintervall zwischen den Zügen ca 0,5s
- Bei Spielende: Auswahl ein weiteres Spiel mit den gleichen Einstellungen oder ein neues Spiel zu starten
- Highscore / Leaderboard: Nach dem Spiel soll der Name des Spielers eingetragen werden können. Dabei soll der Name des letzten Spielers vorgeschlagen werden. Bei gleicher Anzahl von Versuchen entscheidet die Zeit.
- Die Länge des Codewortes sind 5 Stellen und es gibt 8 verschiedene Farben(Änderung der Anforderung möglich, Codewortlänge besser variable gestalten)
- Lösbarkeit des Spiels mit Unsicherheit, keine Schwierigkeitsgrade
- Zeit-Tracking für Highscore, kein Zeitlimit für Rate-Versuche
- Keine Rate-Einschränkung: Bereits ausgeschlossene Farben können wieder erfragt werden

Nicht-funktionale Anforderungen

- Kompatibilität mit jedem Betriebssystem, das Python 3 unterstützt
- Verwendung der letzten Long-Term-Support-Version von Python
- Berechnungszeit soll nicht laggy wirken
- Nutzung von Numpy erlaubt, Implementierung eines neuronalen Netzes in Numpy möglich
- Codierer-Bot ohne Businesslogik
- Allgemeinverständlich in deutscher Sprache, ab 12 Jahren geeignet

- Der Code soll in englischer Sprache geschrieben sein
- Der Algorithmus soll korrekt sein (Widerspruch zur Unschärfe bei Lösbarkeit?)
- Lokale Persistenz die über den Gameloop hinaus geht
- Spielfeldgröße: 13-14 Zeichen x 20 Zeilen

8.3 Anforderungen aus dem Projekt-Auftrags

Funktionale Anforderungen

- Nach jedem Spielzug baut sich das Spielbrett auf der Konsole neu auf.

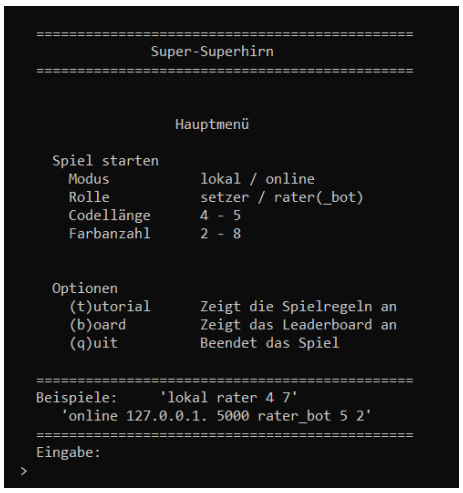
Nicht-funktionale Anforderungen

- Das Spiel muss auf der Konsole/im Terminal zu spielen sein.
- Das Spiel muss objektorientiert in Python 3 implementiert werden.

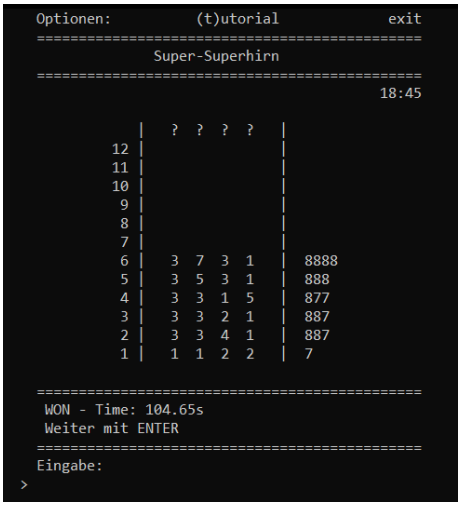
Operative Anforderungen

- Team aus 4-5 Personen
- Entwurf und Analyse müssen den Prinzipien und Methoden aus SE 1 folgen
- Als Modulierungssprache muss UML 2.5 eingesetzt werden.
- Aller Kode (inkl. dem der Qualitätssicherung) ist mit Git zu verwalten und unter der GitLab-Instanz der HTW zu hosten: `https://gitlab.rz.htw-berlin.de`.
- Der Name des Projektes lautet: WiSe23-24_Superhirn_¡12¿.
- Jedes Teammitglied muss hier als Owner eingetragen sein.
- Außerdem muss der Dozent als Owner (ohne Ablaufdatum) eingetragen sein.
- Auch die Dokumentation, inklusive der Projektbeschreibung ist im Projekt via Git zu verwalten. Werden Zwischenversionen gefordert, müssen diese termingerecht über Git (bzw. das GitLab der HTW) zur Verfügung stehen (die Projektbeschreibung als PDF).
- Der Einsatz von Bibliotheken ist erlaubt, solange es sich nicht um solche handelt, die direkt die Spiellogik bzw. den Spielablauf adressieren.
- Codefragmente Dritter (z.B. aus Internetquellen) dürfen nicht eingebaut werden bzw. werden als Plagiate bewertet.
- Aller Code muss in Form von Paarprogrammierung erstellt werden. Aus jedem Commit muss hervorgehen, welche beiden Entwickler beteiligt waren.
- Fokus auf Richtigkeit (Funktionalität), Zuverlässigkeit (insbesondere Fehlertoleranz/Robustheit) und Änderbarkeit/Wartbarkeit (insbesondere Modifizierbarkeit/ Erweiterbarkeit und Stabilität).

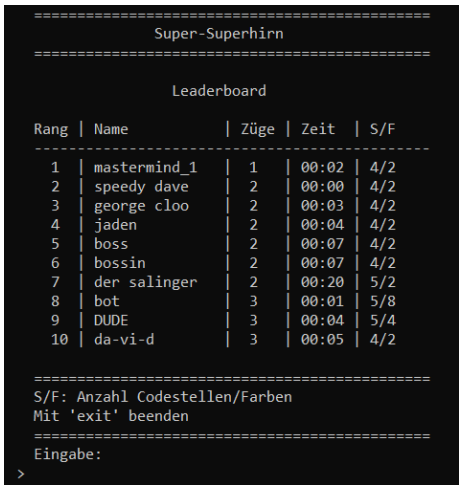
8.4 Letztendliche User-Interfaces



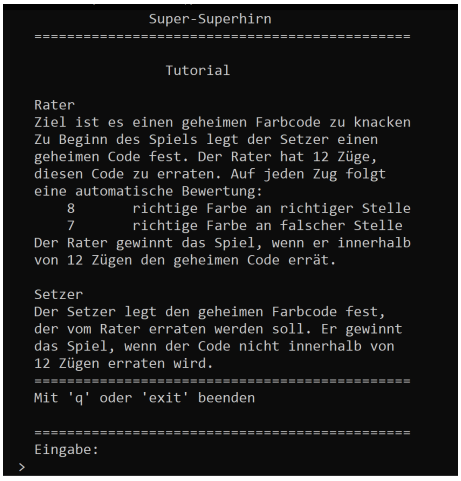
(a) Hauptmenü



(b) Spielfeld



(c) Leaderboard



(d) Tutorial

Figure 10: Auschnitte der letztendlichen User-Interfaces

8.5 Detailansicht Klassendiagramm OOD

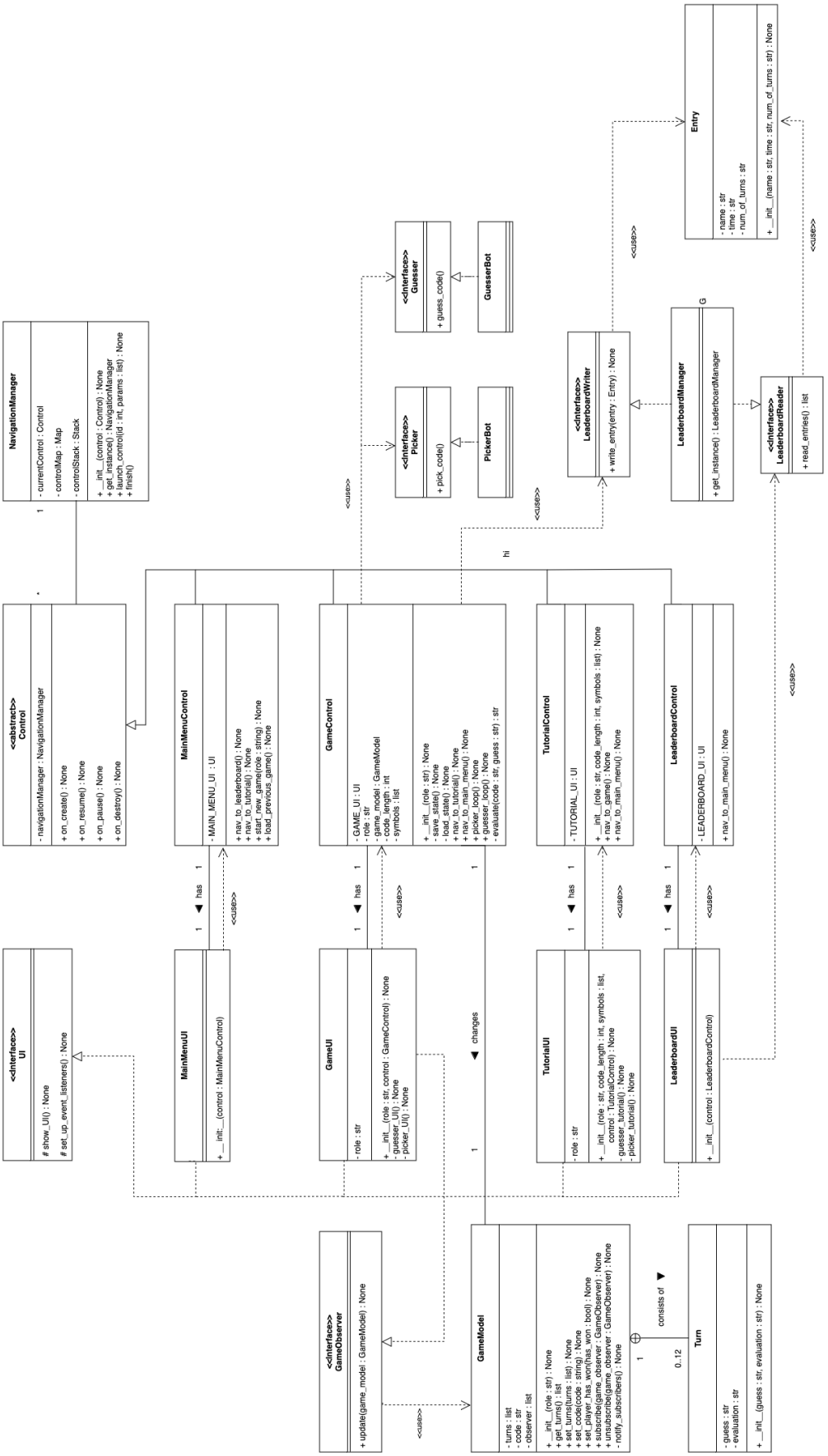


Figure 11: OOD Klassendiagramm