**Project Chessboard**

Abschlussprojekt des Informatikgrundkurses Abitur 2025

Ein Bild, das Schachfigur, Hallensportarten, Schach, Brettspiel enthält.

KI-generierte Inhalte können fehlerhaft sein.

*Foto von Harun Benali, pexels*

Projektzeitraum: 23.01.2025 – 13.03.2025

Projektteilnehmer: Tobias Daust, Rino Lentz, Johanna Striebing

Betreuende Lehrkraft: Frau Börner

Inhaltsverzeichnis

[Teil 1 – Protokoll / Dokumentation 3](#_Toc192622154)

[Motivation 3](#_Toc192622155)

[Exkurs: Ein wenig Schachgrundlagen 3](#_Toc192622156)

[Anforderungsanalyse 4](#_Toc192622157)

[Grundanforderungen 4](#_Toc192622158)

[Mögliche Features 4](#_Toc192622159)

[Lastenheft 5](#_Toc192622160)

[Verteilung der Aufgaben 6](#_Toc192622161)

[Modellierung 7](#_Toc192622162)

[Protokollierung & Erläuterung der Projektteile (Dokumentation) 8](#_Toc192622163)

[Figur.java 8](#_Toc192622164)

[Move.java 9](#_Toc192622165)

[Positon.java 10](#_Toc192622166)

[Testphase 13](#_Toc192622167)

[Teil 2 – Reflexion 15](#_Toc192622168)

[Erläuterung verschiedener Entscheidungen 15](#_Toc192622169)

[Zur Informatikkurs-bezogenen grafischen Ausgestaltung der Spielfiguren 15](#_Toc192622170)

[Zum Überspringen von Figuren 15](#_Toc192622171)

[Zur Logik von Schach, Schachmatt und Patt 15](#_Toc192622172)

[Umsetzung des Software-Life-Zyklus 16](#_Toc192622173)

[Dokumentation des Arbeitsprozesses und Arbeitsanteile der Gruppenmitglieder 16](#_Toc192622174)

[Arbeitsphasen 16](#_Toc192622175)

[Reflexion des Arbeitsprozesses 18](#_Toc192622176)

[Inhaltliche Reflexion 18](#_Toc192622177)

[Allgemeine & methodische Reflexion 18](#_Toc192622178)

[Individuelle Reflexion 19](#_Toc192622179)

[Ausblick 20](#_Toc192622180)

## **Teil 1 – Protokoll / Dokumentation**

*In diesem Teil wird der Arbeitsprozess und das entstandene Projekt beschrieben. Dabei werden Grundlagen zum Verständnis bereitgestellt. Dieser Abschnitt beinhaltet keine Darstellung der Gruppenarbeit, sondern nur Aspekte der Prozessarbeit.*

### **Motivation**

Wir haben uns für diese Art des Projektes entschieden, weil wir uns einerseits – vermutlich offensichtlich – alle für Schach begeistern. Darüber hinaus haben wir entschieden, dass ein Schachspiel genau das richtige Maß an Komplexität besitzt, um es im angestrebten Zeitraum zu schaffen, es ist leicht justierbar, sodass wir es anpassen können, falls sich Schwierigkeiten einstellen sollten. Darüber hinaus bietet es verschiedene Arten von Aufgaben bzw. Bereichen, mit denen wir uns auseinandersetzen können. Dazu zählt die grafische Oberfläche und deren Konzeption sowie Implementierung, das eventuelle Angliedern einer Schachengine und ein mögliches Speichersystem für verschiedene Spieler und Spiele. Als Basis sollte allerdings erstmal ein einfaches, bedienbares „Schachbrett“ dienen, an das man bei Bedarf weitere Funktionen anfügen kann. Das Projekt bietet über den Unterricht hinaus viel Möglichkeit, weiterentwickelt zu werden – wir wollten gern etwas programmieren, was schlussendlich auch verwendet werden kann und als vorzeigbares Projekt dient.

### **Exkurs: Ein wenig Schachgrundlagen**

Um das Programm zu verstehen, soll an dieser Stelle etwas Hintergrundinformation zum Schach an sich gegeben werden. Hier werden auch die Regeln des Schachs kurz aufgeführt.

Schach entstand vermutlich im 6. Jahrhundert in Indien unter dem Namen „Chaturanga“; es beinhaltete dieselben Ideen und bereits ähnliche Figuren. Über Persien, wo es „Shatranj“ genannt wurde, gelangte es später nach Europa. Die „modernen“ Regeln des Schachs entstanden dann circa im 15. Jahrhundert in Spanien und Italien und 1886 fand die erste offizielle Schachweltmeisterschaft statt. Schach wird als Sport betrachtet, es gibt eigene Weltmeisterschaften, Wettbewerbe und ein Ratingsystem. Auch heute hat Schach seine Faszination nicht verloren, es zählt noch immer zu den beliebtesten Brettspielen. Serien wie „Queens Gambit“ lassen das Spiel kürzlich wieder massenhaft populärer werden.

Schach hat keine zufälligen Elemente. Auf 64 Feldern, die schwarz und weiß kariert spielen zwei Teams, schwarz und weiß mit jeweils der gleichen Anzahl Figuren gegeneinander. Jede Art von Figuren hat ihre eigene Art, sich auf dem Brett zu bewegen. Es gibt rund 10120 mögliche Spielverläufe – mehr als Atome im Universum.

Ziel ist es, den gegnerischen König schachmatt zu setzen. Das bedeutet, ihn im nächsten Zug schlagen zu können, ohne, dass die gegnerische Seite etwas dagegen tun kann. Schach bedeutet, den gegnerischen König zu bedrohen, d.h. in im nächsten Zug Schach setzten zu können. Das gegnerische Team hat allerdings noch Möglichkeiten, es zu verhindern. Ein Patt ist erreicht, wenn es keinen legalen Zug mehr gibt, der König aber auch nicht im Schach steht. Die Partie ist dann unentschieden (Remis).

Die Figuren im Schach sind folgendermaßen:

* König (1× pro Spieler): Wichtigste Figur, kann ein Feld in jede Richtung ziehen.
* Dame (1×): Stärkste Figur, bewegt sich beliebig weit horizontal, vertikal und diagonal
* Türme (2×): Bewegen sich horizontal und vertikal
* Läufer (2×): Bewegen sich diagonal
* Springer (2×): Ziehen in L-Form und können über Figuren springen
* Bauern (8×): Bewegen sich nur vorwärts, schlagen diagonal und können durch „Bauernumwandlung“ in eine andere Figur verwandelt werden

Abgesehen davon gibt es noch

1. Die Rochade: Dabei wird der Turm und der König gleichzeitig bewegt. Eine Rochade ist nur auf bestimmten Feldern möglich und muss unter verschiedenen Voraussetzungen stattfinden, zum Beispiel, dass weder der beteiligte Turm noch König bisher gezogen worden sind, außerdem darf keines der „übersprungenen“ Felder oder die, auf denen die Figuren stehen, von Schach bedroht sein.
2. En passant: Eine besondere Schlagregel für Bauen am Anfang des Spiels. Wenn weiß einen Doppelzug mit einem Bauern macht und direkt neben einem schwarzen Bauern landet, darf dieser den weißen Bauern so schlagen, als ob er nur ein Feld gezogen hätte. Dieser Zug muss direkt auf den Doppelzug ausgeführt werden, sonst ist er nicht mehr möglich.

### **Anforderungsanalyse**

#### **Grundanforderungen**

Ausführbares Programm in Java, auf dem zwei Spieler auf einem digitalen Schachbrett gegeneinander Schach spielen können. Diese Spieler sitzen sich idealerweise gegenüber, während ein Tablet zwischen ihnen liegt. Sie können aber auch zum Beispiel nebeneinander vor einem Bildschirm sitzen.

Die Spieler sollen nur nach den Regeln des Schachs spielen können, das bedeutet:

* die weiße Partei fängt an
* die Spieler können nur abwechselnd ziehen
* die Figuren sind anfangs in der klassischen Form aufgestellt
* die Figuren können nur nach ihren spezifischen Bewegungen ziehen
* weitere Regeln wie Rochade und En Passant werden beachtet
* das Spiel erkennt Schach, Schachmatt und Patt und beendet ggf. das Spiel

Darüber hinaus soll das Schachbrett und seine Figuren grafisch ausgearbeitet sein.

#### **Mögliche Features**

Als erstes mögliches Feature soll es möglich sein, ein Spiel mit seinen Spielzügen zu speichern und später abrufen zu können. Daran angliedern könnte man auch eine Spielerdatenbank, die ein Spiel zwei Spielern eindeutig zuordnet. Diese Datenbank soll die Daten auch nach Beenden des Programmes speichern.

Ein weiteres Feature wäre es, eine Schachengine anzugliedern, mit der man ggf. spielen könnte.

Auch wäre es möglich, die grafische Oberfläche des Programmes weiter auszuarbeiten und so zum Beispiel ein Menü oder verschiedene Designs für Brett und Figuren auswählen zu können.

### **Lastenheft**

(Frontend)

* Programm in Java, eignes Fenster, anpassbar
* Schachbrett mit Benutzerschnittstelle in den Feldern (Mausinteraktion), Rückgabe der Felder für das Backend
* Modellierung der Figuren in Klassen; Rückgabe von Positionen und Art der Figur
* Modellierung des Brettes, mit Feldern (Buttons) die belegt oder unbelegt sein können
* Design des Brettes
* Design der Figuren

(Backend)

* Logik, die es erlaubt, dass Figuren nur nach ihrem spezifischen Muster ziehen können (nur Pferde können Figuren überspringen, alle Figuren können nur auf ihre bestimmten Felder ziehen)
* Logik, die die Regeln Rochade und En Passant erlaubt bzw. nicht erlaubt
* Logik, die ein Schach, Schachmatt und Patt erkennt und die Spieler im Spiel nur dementsprechend handeln lässt (wird der eigene König Schach gesetzt, gibt es eine deutlich geringere Anzahl an möglichen Zügen)
* Rückgabe, ob entsprechender Zug erlaubt ist, oder nicht bzw. Verhinderung des Zuges
* Speicherung der Züge bzw. Aktualisierung der auf dem Brett befindlichen Figuren und ihrer Position, da sich daran ein möglicher Zug festmacht

Zusammenführen der Programme / Kommunikation der Schnittstellen

Arbeiten mit GitHub

Paralleles Schreiben der Dokumentation

Planen und durchführen der Testphase

### 

### **Verteilung der Aufgaben**

Rino: alle im Lastenheft vermerkten Punkte zu Frontend, Mitarbeit im Backend sobald fertig

Tobias: alle im Lastenheft vermerkten Punkte zu Backend

Johanna: gemeinsame Modellierung der Backend-Logik mit Tobi, Beraten des Frontends mit Rino, paralleles Schreiben der Dokumentation und Überblick & Planung des Arbeitsprozesses

Alle: Testphase und Anpassen des Programmes danach (jeder in den Bereichen, für die er selbst zuständig war)

### **Modellierung**

Anfängliche Modellierung der Klassen

|  |  |
| --- | --- |
| Klasse: Figur | |
| position | Positon |
| typ | string |
| team | string |
| erster\_zug | boolean |
| Move\_Vorschläge | array |
|  |  |
| move | Funktion , abhängig von typ |
| getPosition | Funktion |
| getTyp | Funktion |
| setPosition | Funktion |
| setTeam | Funktion |
| getTeam | Funktion |
| getMoveVorschläge | Funktion |
| setMoveVorschläge | Funktion |

Diese Modellierung wurde im späteren Programmieren weiter angepasst, da wir verschiedene Aspekte, wie bspw. die Umrechnung der Koordinaten noch nicht bedacht hatten; außerdem wurden teilweise andere Bezeichnungen verwendet; nicht alles wurde umgesetzt; vieles wurde anders umgesetzt.

Diese Modellierung diente vor allem als Grundlage für die Entwicklung der Programmlogik.

|  |  |
| --- | --- |
| Klasse: Button | |
| position | Position |
| Figur\_Existet | boolean |
| Figur\_Type | Figur, abhängig von Figur\_Existent |
| Angeklickt | Boolean |
| Team | Integer |
|  |  |
| actionPerformed | Funktion |

|  |  |
| --- | --- |
| Klasse: Position | |
| Reihe | Int |
| Spalte | Int |
|  |  |
| setReihe | Funktion |
| getReihe | Funktion |
| setSpalte | Funktion |
| getSpalte | Funktion |

|  |  |
| --- | --- |
| Klasse: Spiel | |
| Züge | Array |
| Figuren | Array |
|  |  |
| addZug | Funktion |
| getSpiel | Funktion |
| startSpiel | Funktion |
|  |  |

### **Protokollierung & Erläuterung der Projektteile (Dokumentation)**

*An dieser Stelle sollen zentrale Aspekte des Codes erklärt werden. Es soll insbesondere auf die Objektorientierung, die grafische Ausarbeitung sowie die Logik, die das Spielen nach Schachregeln ermöglicht, geachtet werden. Es wird nicht auf alle Abschnitte des Codes eingegangen, da dies erstens den Rahmen dieser Dokumentation sprengen würde und der Code zweitens auch umfassend dokumentiert ist. Die hier erklärten Abschnitte sollen eher ein Grundverständnis des Codes herstellen.*

1. **Erläuterung des Backend-Codes**

#### **Figur.java**

**Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

KI-generierte Inhalte können fehlerhaft sein.**line 7-19 → definiert die Klasse Figur nach dem Klassendiagramm

ab line 22 → definiert die Funktion move(); diese fügt die zur liste move\_vorschlage, die Positionen hinzu, welche nach Typ der Figur (unterscheidend nach „bauer“, „turm“, „pferd“, „laufer“, „dame“ und „könig“) und anhand ihrer aktuellen Position möglich sind; es handelt sich um die theoretischen Platzierungsmöglichkeiten auf einem leeren Brett, d.h. überspringen von Figuren, belegte Positionen oder „pinnen“ wird hier nicht beachtet

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

ab line 101 → definieren von wichtigen Funktionen, um Teams, Positionen usw. zu setzen und abzurufen

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

#### **Move.java**

Ein Bild, das Text, Schrift, Reihe, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.→ die getFigur() Methode sucht in der alle\_Figuren Liste nach einer bestimmten Figur, auf einer Position; wird also zur Bestimmung der Figur auf einer Position genutzt

→ die eingrenzen\_move() Methode filtert aus allen möglichen Positionen (festgelegt in move()) ungültige Positionen heraus

→ unter dem Kriterium, dass diese Postionen nicht mehr auf dem Brett liegen

Ein Bild, das Text, Schrift, Screenshot, Algebra enthält.

KI-generierte Inhalte können fehlerhaft sein.

→ unter dem Kriterium, dass auf der betreffenden Position schon eine Figur steht

Ein Bild, das Text, Schrift, Reihe, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.

#### **Positon.java**

→ diese vollständige Datei dient dazu, die Klasse Position zu definieren; Annahmeargumente sind dabei Spalte und Reihe; es gibt Funktionen Spalte & Reihe zu setzen und abzurufen

**Uebersetzter.java**

insgesamt: → beinhaltet zwei Funktionen, die Schachkoordinaten (z.B. b3 oder h7) in normale Koordinaten (also x und y, z.B. 2, 3 oder 5, 7) umwandelt; wichtig ist hierbei, dass die normale Zahlen-Koordinatenbezeichnung auf dem Brett von bei null anfängt

→ wandelt Zahlen-Koordinaten in Schachkoordinaten; Übergabeparameter: zwei Integer x und y;

Ein Bild, das Text, Schrift, Screenshot, Algebra enthält.

KI-generierte Inhalte können fehlerhaft sein.

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.→ wandelt Schachkoordinaten in Zahlen-Koordinaten; Übergabeparameter: string

**Main\_Schach.java**

***Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.***→ verschiedene Figuren werden in Startposition nach schwarzem und weißem Team hinzugefügt; hier z.B. die Türme

Ein Bild, das Text, Screenshot, Schrift, Reihe enthält.

KI-generierte Inhalte können fehlerhaft sein.→ eigentliches Aufrufen / Ausführen des Programmes mit der grafischen Oberfläche

1. **Erläuterung des Frontend-Codes**

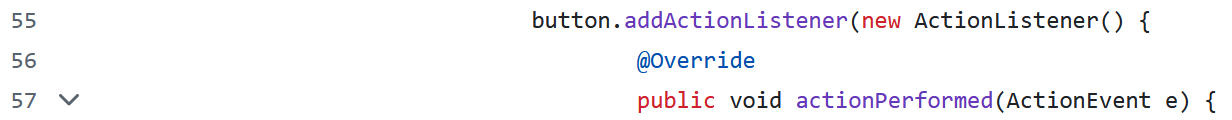
**SchachGUI.java**

*Diese Datei wird trotz ihrer Relevanz und Komplexität nicht in allen Aspekten ausführlich aufgearbeitet, da sie besonders gut und verständlich kommentiert ist.*

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

KI-generierte Inhalte können fehlerhaft sein.→ diese Datei möglich die grafische Darstellung des Spiels; dabei sind alle 64 Felder Buttons die abwechselnd gefärbt sind und mit einer Klasse definiert sind, die ihren Namen und ihren Status als „angeklickt“ als Boolean darstellt,

→ Ausführung nach einem Programmklick (in diesen Teil ist eingebettet, wie die Züge gezogen werden



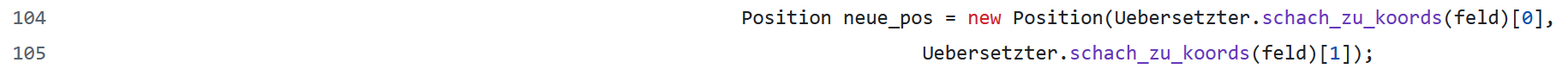
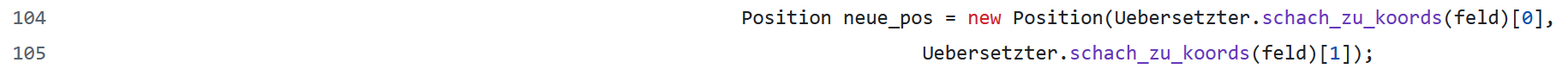
Ein Bild, das Text, Screenshot, Schrift enthält.

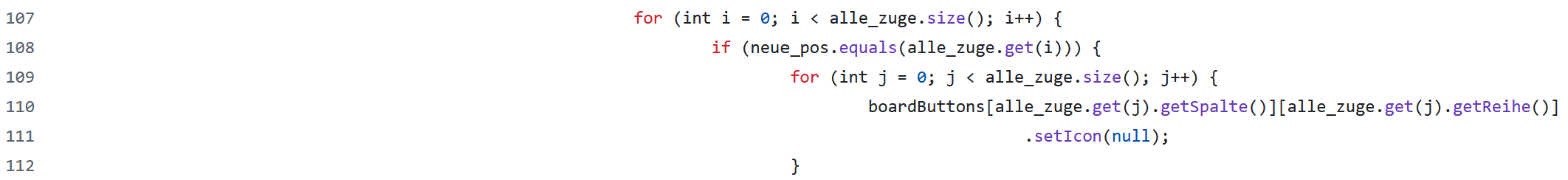
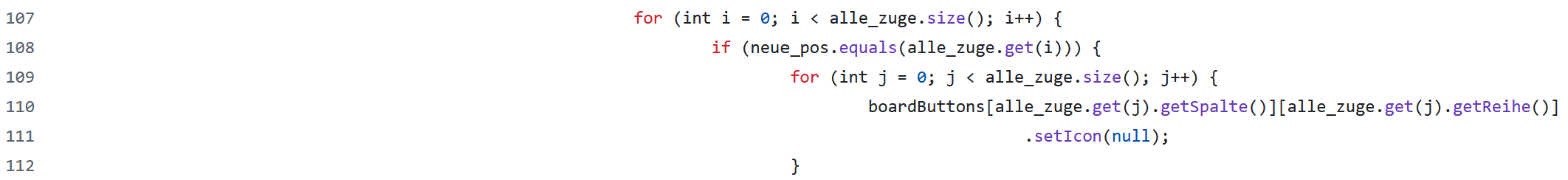
KI-generierte Inhalte können fehlerhaft sein.→ falls noch keine Figur ausgewählt (angeklickt) wurde, wird eine Figur an der entsprechenden Position gesucht und geprüft, ob sie auch die richtige Farbe hat

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.→ falls es eine ausgewählte Figur gibt, so werden alle möglichen Züge angezeigt

### 

→ das angeklickte Feld wird als neue\_pos gespeichert

→ dann wird überprüft ob es sich um einen erlaubten Zug handelt

→Ein Bild, das Text, Schrift, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein. falls ja, wird die Figur grafisch verschoben und das Brett neu gezeichnet (reprint()) und die Werte (wer gerade spielt, die ausgewählte Figur etc.) aktualisiert

Ein Bild, das Text, Schrift, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.

### **Testphase**

Unsere Testphase hat vorerst zu einem großen Teil separat in Front- und Backend stattgefunden. Die grafische Oberfläche des Spielbretts mit den klickbaren Buttons konnten wir somit schon im Zwischenstand zeigen. Auch das Backend wurde selbstverständlich während seiner Entwicklung immer wieder getestet. Am relevantesten für uns war deswegen die Zusammenführung der beiden Teile. Dass sie funktioniert, war uns von vornherein klar, ebenso natürlich, welche Logiken (sprich Regeln) wir nicht implementieren könnte. Insofern waren wir nicht auf große Überraschungen eingerichtet, die sich auch nicht eingestellt haben.

Wir haben das Programm gemeinsam getestet. Bei den Tests, die nach dem Zusammenführen stattgefunden haben, wurde unter Anderem überprüft:

* Ob am Anfang alle Figuren am richtigen Platz stehen
* Ob Figuren anklickbar sind
* Ob Figuren gezogen werden können
* Ob Figuren anders als ihre spezifischen Regeln gezogen werden können
* Ob nur schwarz und weiß nur abwechselnd ziehen können
* Ob weiß anfängt
* Ob die Figuren die richtige Grafik haben
* Ob das Brett die richtige Grafik aufweist
* Ob Figuren geschlagen werden können
* Ob geschlagene Figuren vom Brett verschwinden

Es haben sich nach diesem Testen bei uns keine Fehler herausgestellt; was vor allem auf unsere ausführlichen Tests während der Entwicklung zurückzuführen ist. Etwaige Anpassungen hätten wir bilateral ausgeführt, jeder hätte an dem Projektteil justiert, den er selbst mitgeschrieben / konzeptioniert hatte.

## **Teil 2 – Reflexion**

*In diesem Teil wird das Produkt sowie der Arbeitsprozess, der in Teil 1 beschrieben wurde, analysiert, reflektiert und eingeordnet. Darüber hinaus wird die Gruppenarbeit selbst (z.B. das Verteilen der Aufgaben) dargestellt.*

### **Erläuterung verschiedener Entscheidungen**

#### **Zur Informatikkurs-bezogenen grafischen Ausgestaltung der Spielfiguren**

Dass die Mitschüler\*innen des Kurses in unserem Programm verarbeitet wurden, ist gewiss sicher eines der ersten Dinge, die ins Auge fallen. Für uns – auch im Sinne der abschließenden Präsentation – tangiert es nicht die Seriosität des Projektes (zumal weitere Möglichkeiten, das Brett nach eigenen Wünschen grafisch anzupassen geplant sind), sondern ist eher als Hommage an den Kurs und die Zeit, die wir mit ihm verbracht haben, zu sehen. Wir hoffen sehr, dass sich der eine oder andere in seinen Positionen wiederfindet.

#### **Zum Überspringen von Figuren**

Das Pferd ist die einzige Figur, dass andere Figuren überspringen kann – im Umkehrschluss bedeutet dies, dass alle anderen Figuren es nicht können sollen. Bei uns wird die Anzahl der möglichen Züge für jede Figur in einer Liste gespeichert. Diese Liste wird dann abgeglichen, ob auf den betreffenden Feldern Figuren stehen. Um ein überspringen zu verhindern, müsste man die Zugrichtung kalkulieren. Es wäre beispielsweise egal, ob hinter dem Läufer eine Figur auf den möglich zu beziehenden Feldern steht, wenn er eigentlich nach vorn ziehen möchte. Ist der Weg zu diesem Feld frei, darf die Verstellung in die andere Richtung keine Rolle spielen. Diese Logik zu implementieren ist uns einerseits zu spät klargeworden und auch wenn wir inzwischen Lösungsansätze dafür gefunden haben, reicht doch die Zeit nicht mehr aus, sie zu implementieren.

#### **Zur Logik von Schach, Schachmatt und Patt**

Diese Regeln zu implementieren ist ebenfalls zeitlich nicht mehr möglich gewesen. Prinzipiell soll es so implementiert werden, dass man die möglichen Züge abfragt, und dann anhand der aktuellen Positionen der Figuren auf dem Brett erfragt, ob der König unter den Möglichen „schlagbaren“ Figuren ist. Dazu kommen, wenn man diese Logik implementiert aber auch eine Reduzierung der möglichen Züge: wird der eigene König ins Schach gesetzt, muss er wegbewegt oder die Bedrohung durch Schlagen oder Blocken unterbrochen werden. Auch das „pinnen“ von Figuren, das heißt, dass eine Figur nicht wegbewegt werden kann, weil sonst der König im Schach steht und im nächsten Zug geschlagen werden könnte haben wir noch nicht ausarbeiten können – hierbei hat die Zeit auch nicht mehr für Lösungsansätze gereicht.

### **Umsetzung des Software-Life-Zyklus**

Die Arbeit an dem Projekt lässt sich am ehesten an einem Wasserfallmodell erklären; wir haben zu Anfang Ziele festgelegt und Aufgaben verteilt, die dann von den Teammitgliedern recht selbstständig abgearbeitet wurden. Wir hatten keine sog. „Sprints“ oder Ähnliches – für uns hat sich das aufgrund der unterschiedlichen Zeiten, zu denen schlussendlich am Projekt gearbeitet wird, nicht gut realisieren lassen. Dadurch, dass wir mit GitHub gearbeitet haben, war allen jeweils der aktuelle Stand der Software bekannt und wir konnten gegenseitig unsere Fortschritte verfolgen. Die Unterrichtsstunden wurden damit je nach aktuellem Bedarf entweder für das Programmieren oder für das gemeinsame Planen / Absprachen genutzt. Viel der Arbeit wurde jeweils allein zu Hause geleistet. Dennoch haben wir erstmal stringent an unseren verteilten Aufgaben gearbeitet.

In den eigenständigen Arbeitsphasen sind natürlich zyklische Elemente zu erkennen, wie man, wenn man das eigene Programm immer wieder testet und mit dem Lastenheft bzw. mit den genauen Anforderungen abgleicht. Genauso haben wir irgendwann eine gewisse Routine in unserer Arbeit gefunden – jede Stunde, die wir im Rahmen des Unterrichts zusammengekommen sind, haben wir gemeinsam besprochen, woran wir gerade arbeiten, was realisierbar ist und was nicht. Man kann also insgesamt sagen, dass wir uns an für uns passenden Elementen aus beiden Möglichkeiten bedient haben.

### **Dokumentation des Arbeitsprozesses und Arbeitsanteile der Gruppenmitglieder**

#### **Arbeitsphasen**

|  |  |  |  |
| --- | --- | --- | --- |
| *Tätigkeit* | | | *Beteiligte Personen* |
| Allgemeine Ideenfindung (Suche nach Gemeinsamkeiten, verwerfen von unpassenden Ideen aufgrund von Komplexität etc.) | | | Rino, Tobi, Johanna |
| Entscheidung für ein Thema | | | Rino, Tobi, Johanna |
| Allgemeine Aufgabenverteilung (z.B. → Verantwortung für paralleles Protokollieren liegt bei Johanna) | | | Rino, Tobi, Johanna |
| Ausarbeitung Anforderungsanalyse | | | Rino, Tobi, Johanna |
| Ausarbeitung des Lastenhefts | | | Rino, Tobi, Johanna |
| Verteilung der Aufgaben | | | Rino, Tobi, Johanna |
| Einstieg in die Arbeitsumgebung GitHub | | | anleitend: Tobi; Rino, Johanna |
| Festlegung einer Deadline (ab diesem Zeitpunkt nur noch Testphase) | | | Rino, Tobi, Johanna |
| Rino | Tobi | Johanna | |
| Reflexion einer alten ähnlichen Ausarbeitung zu einem Schachbrett | Inspiration durch bereits bestehende Projekte auf GitHub für Umsetzung, Brainstorming (gemeinsam mit Johanna) | Dokumentation des Arbeitsprozess | |
| Schwachstellen- / Effizienzanalyse | Entwerfen der Klassendiagramme (gemeinsam mit Johanna, Beratung durch Rino) | Kommunikation über Programmschnittstellen, Fristen & aktuelle Arbeitsstände | |
| Umprogrammieren des bereits vorhandenen Codes, derweil Testen und Verbessern | Konzeptionierung der verschiedenen Spiel-Logiken (gemeinsam mit Johanna), Fokussetzung in Anbetracht der Zeit | Aufarbeiten des Codes nach Wichtigkeit, Kommentierung in der Dokumentation | |
| Ausarbeitung von Klassen die im Frontend angesiedelt sind z.B. Buttons (gemeinsam mit Johanna & Tobi) | Syntaktische Ausarbeitung der Spiel-Logiken | Konzeptionierung der Dokumentation (Fokus bei der Kommentierung des Codes, Auswahl der getroffenen Entscheidungen etc.) | |
| Schaffen von Schnittstellen mit dem Backend (gemeinsam mit Tobi) | Reflexion und effiziente Anpassung (mit Johanna), leichte Überarbeitung der Klassen | Aufsetzen des Layouts und der Methoden für die Dokumentation | |
| Ausarbeiten der Figuren mit Gimp |
| → Endprodukt: funktionsfähige Benutzeroberfläche, grafisch ausgearbeitete Figuren, Möglichkeit Figuren durch Klicken zu verschieben & zu schlagen | → Endprodukt: objektorientierte, logische Funktionsbasis des Programms | → Endprodukt: Dokumentation | |
|  |  |  | |
| Testen des Programms | | Rino, Tobias, Johanna | |
| (Dokumentation der Tests) | | Johanna | |
| Individuelle Reflexion | | Rino, Tobias, Johanna | |
| Gemeinsame Abnahme von Produkt und Dokumentation | | Rino, Tobias, Johanna | |

Zum Verständnis der Arbeitsphasen: Wir haben an unseren Aufgabenteilen viel getrennt gearbeitet und auch zu Hause einige Zeit in das Projekt investiert. Die Unterrichtsstunden waren für uns vor allem auch ein Platz, um uns über den aktuellen Stand auszutauschen und neue Aufgaben zu verteilen. Wann genau welche Arbeit passiert ist, lässt sich nicht ganz genau rekonstruieren. Wir geben deshalb keine festen Daten und Zeitpunkte an.

### **Reflexion des Arbeitsprozesses**

#### **Inhaltliche Reflexion**

Unsere Ansprüche, wie sie zu anfangs im Lastenheft formuliert wurden konnten wir nur in Teilen umsetzen. Das liegt unter anderem daran, dass wir erst in Prozess des Programmierens Probleme gemerkt haben und beispielsweise unsere Klassen von der anfänglichen Modellierung abweichend daran angepasst haben. Dass es so kommen wird, war uns von Anfang an klar, und wir haben unsere Arbeitsweisen daran angepasst, indem wir viel auch von zu Hause ausgearbeitet haben.

Wir haben uns von den Basics, also wie die Figuren insgesamt auf einem leeren Brett ziehen könnten weitergearbeitet zu den Logiken, jeweils nur abwechselnd ziehen zu können oder Figuren zu schlagen. Dabei hätten wir retrospektiv von Anfang an klarmachen müssen, in welcher Reihenfolge wir die Schritte bearbeiten, da sie aufeinander aufbauen. Dennoch haben wir diesen Teil stark intuitiv umgesetzt, worauf man sich bei größeren Projekten jedoch nicht wieder verlassen sollte.

Auch sind uns während der Arbeitsphasen zwei weitere Dinge klar geworden: zunächst, wie gering die insgesamt Zeit für das Projekt eigentlich war und dass unsere eigenen Ansprüche daran angepasst werden müssen; sowie zweitens, dass je mehr Logiken in das Programm implementiert werden, desto rechenintensiver wird es auch. Bei unserer aktuellen Umsetzung gibt es ein Array von möglichen Zügen für jede Figur, aus dem dann bei Bedarf aussortiert wird. Je mehr Regeln implementiert werden und, desto länger ein Spiel dauert, desto mehr muss berechnet werden. Dadurch wird das Programm langsam, doch wir stehen noch nicht an dem Punkt, unsere aktuelle Umsetzung effizienter zu gestalten, da wir noch nicht alle Regeln implementieren konnten.

#### **Allgemeine & methodische Reflexion**

Vom Projekt bleibt schlussendlich nicht nur das einigermaßen fertig programmierte Schachbrett selbst, sondern auch viel Erfahrung in der Zusammenarbeit zurück. Insgesamt kann gesagt werden, dass wir viel über Software-Development und dessen Planung gelernt haben. Die Kommunikation hat sich als der unbekannteste, neuste Teil herausgestellt, da wir nicht wussten, wie wir unsere Arbeit teilweise aufeinander abstimmten sollten. Auch wenn wir zu Anfang unsere Ziele klar definiert und Aufgaben verteilt hatten, gab es doch zeitweise Schwierigkeiten zu wissen, wer wann woran arbeitet und welche Teile gerade aufeinander aufbauen und miteinander verknüpft werden sollten. Allerdings haben wir das dann gut mit verschiedenen Fristen umsetzten können. Es hat sich als extrem hilfreich herausgestellt, sich gegenseitig über den Code auszutauschen, um auf neue Umsetzungsideen zu kommen.

In der Programmier-Praxis konnten wir außerdem von den Lehrinhalten, die wir uns über die Semester erarbeitet hatten, profitieren. Insbesondere von der Projektorientierung; auch wenn dieses Semester schon etwas länger her war.

Insgesamt hat es sich als sehr effektiv erwiesen, dass wir die Aufgaben nach unseren unterschiedlichen Fähigkeiten verteilt haben. Damit waren wir nicht nur schneller, sondern hatten auch eher Spaß an unserem Projekt. Bestimmte Abschnitte wie die Modellierung oder das Einarbeiten in GitHub haben wir vollständig zusammen gemeistert. Die Arbeit in GitHub müssen wir insgesamt positiv hervorheben – dafür, dass wir es uns mit unseren unterschiedlichen Vorstellungen und Wissensständen selbst beigebracht haben, hat es überraschend effektiv funktioniert.

#### **Individuelle Reflexion**

Außerdem soll an dieser Stelle individuell dokumentiert werden, was die einzelnen Projektteilnehmer aus der Arbeit inhaltlich und methodisch gelernt haben, wie sie diese erworbenen Kenntnisse in Zukunft anwenden können und was sie am Projekt gern noch geändert hätten.

**Johanna**

Ich empfand es als willkommene Abwechslung, dass wir nach drei Semestern mit eher theoretischem Schwerpunkt auch ein vollständig eigenes Projekt programmieren konnten. Natürlich war dabei sehr nützlich, vorher im Unterricht die Phasen der Softwareentwicklung durchgegangen zu sein. Ich habe inhaltlich einiges über Java gelernt, eine Programmiersprache, in der ich noch nicht sehr viel Erfahrung hatte (eher JavaScript). Das Ganze zusammen mit Tobi umzusetzen, indem wir beide zusammen an der Logik des Codes arbeiten können, auch wenn ich mich mit der Syntax nicht genau auskenne, hat sich als überraschend produktiv herausgestellt. Trotzdem weiß ich jetzt auch, wie anspruchsvoll es ist, Softwareprojekte zu koordinieren. Dadurch, dass ich mit einer Art Kommunikationsschnittstelle betraut war und zeitgleich auch die Dokumentation schreiben wollte, war es oft ich, die Fristen (durch-)setzen musste. Insgesamt bin ich sehr dankbar für die Erfahrung und rechne damit, auch in einem späteren Studium davon profitieren zu können – auch, wenn wir beispielsweise GitHub ein wenig anders als an der Uni genutzt haben. Aber gerade diese Unterschiede haben mir neue Arbeitsweisen aufgezeigt. Ich hätte unfassbar gern noch eine Schachengine an das Projekt angefügt – aber diese Schnittstelle zu schaffen, ist sehr viel komplizierter und zeitaufwändiger, als wir es uns anfangs vorgestellt haben. Vielleicht ist es ein zukünftiges Projekt.

**Rino**

Während des Projekts im Informatikunterricht, bei dem ich überwiegend programmiert und speziell für die Gestaltung des GUI unseres Schachspiels verantwortlich war, habe ich wertvolle Erfahrungen gesammelt. Die Arbeit an der Benutzeroberfläche hat mir gezeigt, wie essenziell ein gut durchdachtes und intuitives Design für das Gesamtprojekt ist. Es gab Momente, in denen die Integration verschiedener GUI-Elemente und die Abstimmung mit der Spiel-Logik herausfordernd waren, doch genau diese Hürden haben meinen Lernprozess maßgeblich bereichert.

Besonders schätzte ich die Möglichkeit, meine Programmierkenntnisse praxisnah einzusetzen und zu erweitern. Das Projekt hat mich gelehrt, wie wichtig es ist, strukturiert vorzugehen und auch in stressigen Phasen den Überblick zu behalten. Durch regelmäßige Abstimmungen mit den Teammitgliedern konnte ich nicht nur mein eigenes Wissen einbringen, sondern auch von den Erfahrungen meiner Kollegen profitieren. Diese Zusammenarbeit hat gezeigt, wie wertvoll eine offene Feedbackkultur und klare Kommunikation in der Teamarbeit sind.

Rückblickend bin ich stolz auf das Erreichte und sehe das Projekt als einen wichtigen Schritt in meiner persönlichen und fachlichen Weiterentwicklung. Die Herausforderungen, die ich überwinden musste, haben mein Verständnis für Softwareentwicklung vertieft und mir gezeigt, dass kontinuierliches Lernen und Ausprobieren der Schlüssel zum Erfolg sind.

**Tobias**

Während des Prozesses, das Schachprogramm zu entwickeln, habe ich eine Menge gelernt, vor allem im Bereich der Backend-Logik. Es war unglaublich bereichernd, mich so intensiv mit den inneren Mechanismen des Spiels auseinanderzusetzen. Mit der Unterstützung von Johannes konnte ich tiefer in die Logik eintauchen und Probleme aus neuen Blickwinkeln betrachten. Die Zusammenarbeit hat den gesamten Prozess nicht nur effektiver, sondern auch inspirierender gemacht.

Am meisten Spaß hatte ich, wenn ich auf scheinbar unlösbare Hürden gestoßen bin und diese Schritt für Schritt knacken konnte. Es war ein besonderer Moment, als der Code endlich funktionierte und die Logik des Programms aufging. Solche Erfolgserlebnisse sind einfach unschlagbar.

Die Herausforderungen waren allerdings auch zahlreich. Besonders knifflig war es manchmal, den Überblick über komplexe Abläufe und Zusammenhänge zu behalten. Aber mit Geduld, viel Debugging und vor allem durch den Austausch mit Johannes habe ich einen Weg gefunden, jedes Hindernis zu überwinden. Dieser Prozess hat mir gezeigt, wie wichtig Ausdauer und eine offene Herangehensweise sind.

Insgesamt hat mir dieses Projekt nicht nur geholfen, technisches Wissen zu vertiefen, sondern auch meine Problemlösungsfähigkeiten und Teamkompetenzen zu stärken. Es war eine aufregende Reise, und ich bin stolz darauf, was ich dabei erreicht habe.

### **Ausblick**

Auch wenn das Projekt für Außenstehende schon sehr weit entwickelt wirken mag, ist es für uns keineswegs abgeschlossen. Es steht außer Frage, dass Gruppenmitglieder auch nach Abschluss des Projektes für den Rahmen des Unterrichts weiter daran arbeiten und es weiterentwickeln werden. Dabei spielen vor allem die Features eine Rolle, die den zeitlichen Rahmen des Projektes, auch wenn zuhause viel Arbeit getan wurde, einfach gesprengt hätten.

Zu den Dingen, die noch implementiert werden, zählt zum Beispiel, dass nur das Pferd Figuren überspringen kann. Über Möglichkeiten der Umsetzung haben wir im Team schon diskutiert, auch wenn es nicht mehr bis zur Abgabe umgesetzt wird. Auch ist die Angliederung einer Schachengine ein großer Wunsch, und damit verbunden natürlich auch ein anschauliches Menü, das Speichern von Spielern und Spielen. Vielleicht gibt es die Möglichkeit, die Spiele in Verbindung mit externen Programmen auswerten zu lassen und mit Statistiken zu veranschaulichen. Außerdem gab es bei uns allen den Wunsch nach mehr Designs für Spielfeld und Figuren.