# SOFTWARE REQUIREMENTS SPECIFICATION

# for

# MELT Chess

**Version 0.1**

**14. Juli 2021**

# Inhaltsverzeichnis

# 1 Einführung

## 1.1 Projektanforderungen

Entwickelt werden soll ein Schach Programm, welches es ermöglicht gegeneinander als auch gegen eine künstliche Intelligenz Schach zu spielen. Das Spiel soll dafür sowohl über eine graphische als auch über eine konsolenbasierte Benutzerschnittstelle verfügen. Das Spiel soll in englischer Sprache umgesetzt werden. Die Entwicklung soll sich dabei in drei Iterationen gliedern. In diesem Dokument sind die Anforderungen an die erste Iteration dargelegt.

## 1.2 Zielgruppe des Dokuments und weitere Resourcen

Dieses Anforderungsdokument richtet sich zum einen an die beteiligten Entwickler und dient zur Orientierung ob die Funktionalität des Projekt gemäß den Anforderungen umgesetzt wird, und zum anderen an die Kontaktperson(en) des Moduls um die Planung des Projekts zu überprüfen.

Einen weiteren Überblick bieten die Storycards, welche im Gitlab des Projekts zu finden sind und in Form von Issues umgesetzt werden.

# 2 Umfang des Projekts

## 2.1 Funktionale Anforderungen

Zum Umfang gehört in der 1. Iteration des Projekts die Bereitstellung einer Textbasierten Konsolen-Schnittstelle in der es möglich ist Mensch-gegen-Mensch Spiele zu spielen. Dazu ist die Umsetzung des Pakets model nötig, in dem der Zustand des Bretts sowie die Schachregeln[1] implementiert werden.

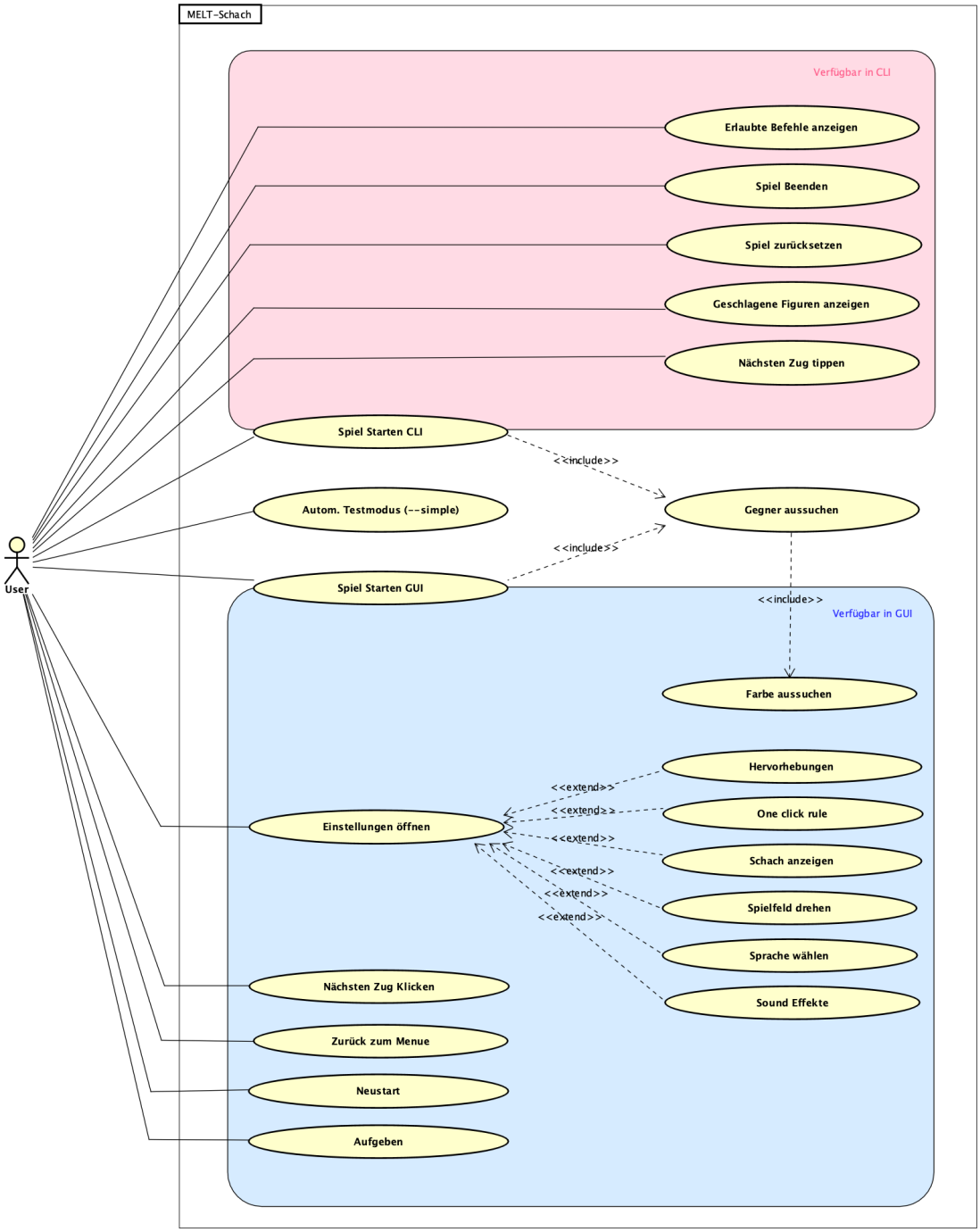
In der 2. Iteration wird die geschaffene Basis durch eine 2D-GUI unter Verwendung des JavaFX Moduls erweitert und eine rudimentäre Schach KI im Modul engine erstellt.

In der 3. Iteration soll die Anwendung um Netzwerkfähigkeit mit der Anwendung der Gruppe 2 erweitert werden. Zusätzlich soll eine Auswahl von möglichen Erweiterungen umgesetzt werden, welche in Summe einen Aufwand von mindestens 10 Einheiten aufweisen müssen. Die Auswahl mit möglicher Punktzahl lauten:

- Verbesserte KI mithilfe Min-/Max-Suche mit $\alpha/\beta$-Pruning (5)
- 3D-GUI (5)
- Eindimensionales Schach (5)
- Rückgängig-Machen von Zügen (3)
- Speichern/Laden von Spielen (3)
- Schachuhren (2)
- Zweisprachigkeit (2)
- Resizeable GUI (1)

[1] Nach den Regeln des Weltschachverbands (FIDE) in der deutschen Übersetzung von 2018

## 2.2 Usecase-Diagramm



## 2.3 Nicht-funktionale Anforderungen

### 2.3.1 Codingstyle, Metriken, Testabdeckung

Das Template verwendet die Plugins PMD und JaCoCo zur Generierung von Reports über Metriken, Codestyle und Testabdeckung. Es wird eine Testabdeckung des gesamten Codes von 90% Instruction Coverage erwartet, ausgenommen der GUI-Klassen. Im Template wird ein PMD-Regelsatz eingebunden. Diese Regeln sind für den Code und die Testfälle einzuhalten.

### 2.3.2 Überprüfungen

Es soll mindestens alle zwei Wochen ein Treffen mit dem zugewiesenen Tutor stattfinden. Neben den Deadlines der drei Iterationen gibt es außerdem noch folgende Termine:

- 23.04.2021: Abgabe Anforderungsanalyse, Vorgehensplan, Prüfung erfolgreiche Einrichtung der Infrastruktur
- 19.05.2021: Prüfung, ob auslieferbare Version vorliegt, die Anforderungen der ersten Iteration genügt mit automatisierten Tests und Theorie-Abfrage
- 16.06.2021: Prüfung, ob auslieferbare Version vorliegt, die Anforderungen der zweiten Iteration genügt mit Zwischenpräsentation
- 14.07.2021: Endabgabe mit Abschlusspräsentation

### 2.3.3 Abgabeformat

An den Schlusstagen der Iterationen, inklusive Endabgabe, muss der zu überprüfende/bewertende Stand mittels Tags (it1,it2,...) im git-Repository gekennzeichnet werden.

# 3 Vorgehensplan

Der Vorgehensplan wurde durch Storycards im Issuetracker von gitlab realisiert. Für die Dauer des Praktikums können diese hier[1] eingesehen werden. Im Folgenden befindet sich eine maschinell generierte Version aus dem csv Export von gitlab.

## 3.1 Storycard Issues

[1] https://projects.isp.uni-luebeck.de/melt-chess/melt-chess/-/issues

### Schnittstelle Konsole

id: #3 Milestone: 1. Iteration

Priorisierung A

Storypoints 5

Risiko high

Der Benutzer hat die Möglichkeit das Schachspiel gegen einen anderen Benutzer zu spielen. Dazu werden die Züge in der Konsole eingegeben.

**Abgeschlossen wenn**

- ☑ Die Benutzereingabe wird korrekt geparsed (Tests für “parseUserMoveInput“ bestanden)
- ☑ Gameloop ist etabliert (User Input einlesen, Schachbrettposition ausgeben, ...)
- ☑ Ausgabe der geschlagenen Figuren auf den Befehl “beaten“
- ☑ Ausgabe wenn ein Spieler im Schach steht
- ☑ Die Anforderungen an die Ausgabe bei gesetztem “
- ☑ Fehleingabe wird mit “!Invalid move“ quittiert
- ☑ Gültige Eingabe wird mit “!¡Eingabe¿“quittiert
- ☑ Ein ungültiger Zug wird mit “!Move not allowed“quittiert
- ☑ Schachbrett wird nach jedem Zug geprintet
- ☑ Matt oder Unentschieden wird angezeigt und dadurch wird das Spiel beendet
- ☑ Es gibt einen Befehl, um das Spiel zu beenden
- ☑ Nach einer falschen Eingabe wird auf eine neue Eingabe gewartet

### Erstellen der Models

id: #4 Milestone: 1. Iteration

Priorisierung A

Storypoints 31

Risiko high

Erstellen der Klassen im Paket `models` welche von den anderen Paketen `cli`, `engine` und `gui` benutzt werden

**Abgeschlossen wenn**

- ☑ Alle blockierenden Storycards sind abgeschlossen

### Erstellen Board Klasse

id: #5 Milestone: 1. Iteration

Priorisierung B

Storypoints 2

Risiko low

Die `Board` Klasse im Paket `model` kapselt eine Position auf dem Brett, sowie die Information welcher Spieler am Zug ist. Die Position wird dabei als Array mit 64 Integerwerten zwischen 0 und 23 kodiert.

**Abgeschlossen wenn**

- ☑ Die Klasse repräsentiert eine Position auf dem Brett
- ☑ Ein neues Objekt kann durch übergeben eines FEN
- ☑ Ein neues Objekt kann durch die “makeMove“ Methode aus einem bestehenden Objekt erzeugt werden
- ☑ Die Methoden “getPiecePositionsFor“ und “getPieceAt“ bestehen alle mithilfe von FEN

### Erstellen Move Klasse

id: #6 Milestone: 1. Iteration

Priorisierung B

Storypoints 1

Risiko low

Die `Move` Klasse im Paket `model` soll einen Container für einen Spielzug verwirklichen. Sie merkt sich dazu von und zu welchem Feld der Zug gemacht wird, sowie eine Flag die besondere Umstände des Zugs (Rochade, Figurentausch, ...) beschreibt.

**Abgeschlossen wenn**

- ☑ Die Informationen zu einem Zug werden erfolgreich kodiert
- ☑ Die toString Methode gibt eine String repräsentation des Zuges gemäß den Anforderungen des Konsolen Clients zurück

### Erstellen der MoveGenerator Klasse

id: #7 Milestone: 1. Iteration

Priorisierung C

Storypoints 13

Risiko high

Die `MoveGenerator` Klasse im Paket `model` soll mögliche Züge zu einer gegebenen Position unter berücksichtigung der Schachregeln generieren. Die werden sowohl von der GUI zum hervorheben der möglichen Felder beim bewegen einer Figur, als auch von der Engine bei der Suche nach dem nächsten Zug verwendet.

**Abgeschlossen wenn**

- ☑ Der Generator kann die verschiedenen Laufrichtungen zu den unterschiedlichen Schachfiguren berechnen
- ☑ Die "generate" Methoden bestehen sinnvolle Tests (Siehe die Flags in der Klasse "Move": werden auch diese Züge gefunden?)

### Erstellen der Piece Klasse

id: #8 Milestone: 1. Iteration

Priorisierung B

Storypoints 2

Risiko low

Die Klasse `Piece`im Paket `model` soll die geplanten statischen Methoden zur Extraktion der kodierten Informationen zur Verfügung stellen. Typ und Farbe einer Schachfigur wird dabei als Integer zwischen 0 und 23 kodiert.

**Abgeschlossen wenn**

- ☑ Die statischen Methoden zur Rückgabe der in einem Integer Wert codierten Informationen bestehen alle Tests.
- ☑ Die toString Methode gibt die vorher festgelegten Symbole korrekt wieder

### Erstellen der 2D GUI

id: #9 Milestone: 2. Iteration

Priorisierung E

Storypoints 34

Risiko high

Zum planen GUI für die Folgenden Schritte weitere Issues anlegen und mit #9 verlinken: Design, Architektur, Implementierung

**Abgeschlossen wenn**

- ☑ Klare Vorstellung wie die GUI aussehen soll (Design)
- ☑ Die Architektur wurde geplant
- ☑ Die GUI wurde implementiert

### Erstellen der Engine

id: #10 Milestone: 2. Iteration

Priorisierung A

Storypoints 13

Risiko high

Erstellen der Klasse `Engine` im Paket `engine`. Die Engine berechnet zu einer `Board`-Instanz einen neuen Zug.

**Abgeschlossen wenn**

- ☑ Die Engine ist in der Lage ein paar einfache ausgewählte Schachprobleme zu lösen
- ☑ Die Engine kann sinnvolle nächste Züge für ein PVPC Spiel generieren

### Erstellen MoveValidator Klasse

id: #11 Milestone: 1. Iteration

Priorisierung D

Storypoints 8

Risiko high

Der `MoveValidator` soll die vom `MoveGenerator` berechneten Züge auf gültigkeit überprüfen. Da die erlaubten Bewegungsrichtungen bereits in `MoveGenerator` korrekt implementiert sind, müssen hauptsächlich die Sonderregeln die zu schach des Königs führen beachtet werden.

**Abgeschlossen wenn**

- ☑ Der König kann nicht in schach bewegt werden
- ☑ Gefesselte eigene Figuren können den König nicht in Schach setzen
- ☑ Ist ïm schachïmplementiert, wurde das generieren von Rochadezügen in "MoveGenerator" verhindert. Edit: Die Züge werden zwar generiert, aber vom "MoveValidator" abgelehnt.

### Erstellen der Game Klasse

id: #12 Milestone: 1. Iteration

Priorisierung E

Storypoints 3

Risiko low

Die `Game` Klasse soll den Gesamtzustand einer Partie kapseln, aber hauptsächlich für die beiden GUI Module relevant sein.

**Abgeschlossen wenn**

- ☑ Funktion für neues Spiel starten"
- ☑ Verwaltet alle nötigen Objekte einer Partie und bietet ein einfaches Interface für die GUI Module
- ☑ Erkennt Schachmatt und Patt (#13 )

### Schachmatt und Patt

id: #13 Milestone: 1. Iteration

Priorisierung E

Storypoints 2

Risiko low

Es müssen die Fälle Patt (ein Spieler hat keine möglichen Züge) und Schachmatt (ein Spieler hat keine möglichen Züge und steht im Schach) implementiert werden

**Abgeschlossen wenn**

☑ Partie kann in Patt enden

☑ Partie kann durch schachmatt gewonnen/verloren werden

### Testen von model und cli

id: #14 Milestone: 1. Iteration

Priorisierung A

Storypoints 8

Risiko high

Testen der Konsolenschnittstelle und damit auch des `model` Pakets mittels der bereitgestellten checker App.

**Abgeschlossen wenn**

☑ checker App läuft und kann MELT Chess ausführen

☑ Bugs und nicht bestandene Tests sind behoben

### Erstellen MenuController

id: #15 Milestone: 2. Iteration

Priorisierung C

Storypoints 13

Risiko high

Der ‘MenuController‘ interpretiert die Aktionen des Benutzers, die dieser über die [‘MenuView‘](#16) ausführt.

**Abgeschlossen wenn**

☑ Der Benutzer kann einen von drei Spielmodi wählen: "Game against AI / Spiel gegen KI", "Game against second player / Spiel gegen zweiten Spieleröder NNetwork game / Netzwerkspiel". Der gewählte Modus muss sich grafisch abheben.

☑ Der Benutzer kann eine von zwei Farben der Spielfiguren wählen: weiß oder schwarz. Die gewählte Farbe muss sich grafisch abheben.

☑ Die Taste SStart game / Spiel startenßtartet das Spiel gegen die KI, gegen den zweiten Spieler oder startet das Netzwerkspiel, je nachdem, was der Benutzer ausgewählt hat

☑ Die Taste SSettings / Einstellungen"öffnet das Einstellungsmenü

### Erstellen MenuView

id: #16 Milestone: 2. Iteration

Priorisierung A

Storypoints 8

Risiko high

Zeigt dem Benutzer das Hauptmenü an.
**Abgeschlossen wenn**

- ☑ Enthält drei Tasten zur Auswahl des Spielmodus: "Game against AI / Spiel gegen KI", "Game against second player / Spiel gegen zweiten Spieleründ NNetwork game / Netzwerkspiel". Die ausgewählte Taste muss sich grafisch abheben.‘handleRadioButtonAIOnAction‘handleRadioButtonPVPOnAction‘handleRadioButtonNetOnA
- ☑ Enthält zwei Tasten zur Auswahl der Farbe der Spielfiguren: weiß und schwarz. Die ausgewählte Taste muss sich grafisch abheben.‘handleRadioButtonColorBlackOnAction‘handleRadioB
- ☑ Enthält Spielstarttaste SStart game / Spiel starten„handleButtonStartOnAction‘
- ☑ Enthält eine Taste SSettings / Einstellungenßum Öffnen der Einstellungen‘handleButtonSettingsOnAc

### Erstellen SettingsView

id: #17 Milestone: 2. Iteration

Priorisierung A

Storypoints 13

Risiko high

Zeigt dem Benutzer das Einstellungen an.
**Abgeschlossen wenn**

- ☑ Enthält eine Taste SSave / Speichernßum Speichern der Einstellungen.‘handleButtonSaveOnAction‘
- ☑ Enthält Taste Cancel / Abbrechenßum Verlassen der Einstellungen.‘handleButtonCancelOnAction‘
- ☑ Enthält eine Label mit dem Text Ïnterface language / Interfacesprache".
- ☑ Enthält zwei Radio Buttons mit möglichen Sprachen.‘handleRadioButtonEnOnAction‘handleRadioBu
- ☑ Enthält ein Kontrollkästchen mit dem Text Rotate the chessboard after each move / Das Schachbrett nach jedem Zug drehen".‘handleCheckboxFlipBoardOnAction‘

- ☑ Enthält ein Kontrollkästchen mit dem Text Öne
- ☑ Enthält ein Kontrollkästchen mit dem Text SShow that the player is in check / Zeigen, dass der Spieler in Schach ist".'handleCheckboxShowInCheckOnAction'
- ☑ Enthält ein Kontrollkästchen mit dem Text SShow possible moves when selecting a character / Mögliche Züge bei Auswahl einer Figur anzeigen".'handleCheckboxShowPossibleMovesOnA

### Erstellen SettingsController

id: #18 Milestone: 2. Iteration

Priorisierung C

Storypoints 13

Risiko high

Der 'SettingsController' interpretiert die Aktionen des Benutzers, die dieser über die ['SettingsView'](#17) ausführt.

**Abgeschlossen wenn**

- ☑ Der Benutzer kann die Sprache über das Auswahlmenü ändern.
- ☑ Mit der Taste SSave / Speichern"kann der Benutzer die geänderten Einstellungen speichern.
- ☑ Mit der Taste Cancel / Abbrechen"kann der Benutzer die geänderten Einstellungen verwerfen und das Einstellungsfenster verlassen.
- ☑ Der Benutzer kann das Kontrollkästchen verwenden, um Rotate the chessboard after each move / Das Schachbrett nach jedem Zug drehenßu aktivieren oder zu deaktivieren.
- ☑ Der Benutzer kann das Kontrollkästchen verwenden, um Öne
- ☑ Der Benutzer kann das Kontrollkästchen verwenden, um SShow that the player is in check / Zeigen, dass der Spieler in Schach istßu aktivieren oder zu deaktivieren.
- ☑ Der Benutzer kann das Kontrollkästchen verwenden, um SShow possible moves when selecting a character / Mögliche Züge bei Auswahl einer Figur anzeigenßu aktivieren oder zu deaktivieren.

### Erstellen Settings Klasse

id: #19 Milestone: 2. Iteration

Priorisierung

Storypoints

Risiko S

Speichert den Status der aktuellen Einstellungen und stellt sie zur Abfrage bereit.
**Abgeschlossen wenn**

- ☑ Die Klasse besitzt alle nötigen Felder für die verschiedenen Einstellungsmöglichkeiten
- ☑ Die Klasse kann serialsiert werden

### Erstellen TextManager Klasse

id: #20 Milestone: 2. Iteration

Priorisierung B

Storypoints 8

Risiko low

Steuert die Ausgabe des gewünschten Sprachpakets in Abhängigkeit von der aktuellen Sprache
**Abgeschlossen wenn**

- ☑ Andere Klassen können durch den 'TextManager' die Beschriftungen von Labels ändern lassen

### Erstellen NetworkConnectView

id: #21 Milestone: 2. Iteration

Priorisierung E

Storypoints 8

Risiko low

Die 'NetworkConnectView' gibt dem Benutzer die Möglichkeit, die Daten für die Netzwerkverbindung einzugeben.
**Abgeschlossen wenn**

- ☑ Es gibt das Textfeld ‘textfieldIPAddress‘, in dem die IP
- ☑ Es gibt das Textfeld ‘textfieldPort‘, in dem das entsprechende Port eingegeben werden kann. Dieses Textfeld ist mit dem Label ‘lblPort‘ Port”versehen.
- ☑ Es gibt das Label ‘lblConnectionFail‘ Connection Error, check IP
- ☑ Es gibt den Button ‘handleConnectSaveOnAction‘ Connect/”Verbinden”.
- ☑ Es gibt den Button ‘handleButtonCancelOnAction‘ Cancel/Äbbrechen”.

### Erstellen NetworkConnectController

id: #22 Milestone: 3. Iteration

Priorisierung

Storypoints

Risiko D

Der ‘NetworkConnectController‘ interpretiert die Aktionen des Benutzers, die dieser über die ‘NetworkConnectView‘ ausführt.

**Abgeschlossen wenn**

- ☑ Bei Klick auf ‘btnConnect‘ wird das ‘lblConnectionFail‘ auf ünsichtbar”geändert und ein Verbindungsaufbau entsprechend der Nutzereingabe versucht.
- ☑ Schlägt die Verbindung fehl, wird die Sichtbarkeit von ‘lblConnectionFail‘ auf ßichtbar”geändert.
- ☑ Gelingt die Verbindung, so wird die ‘GameView‘ als aktuelle ‘Scene‘ gesetzt und das Spiel wird gestartet.
- ☑ Bei einem Klick auf ‘btnCancel‘ wird die ‘MainMenueView‘ als aktuelle ‘scene‘ gesetzt.
- ☑ Die eigene IP

### Erstellen GameView

id: #23 Milestone: 2. Iteration

Priorisierung A

Storypoints 21

Risiko high

Die ‘GameView‘ ist die View, in der der Nutzer das eigentliche Spiel spielen kann, hier kann er Züge eingeben und das Spielfeld sehen.

**Abgeschlossen wenn**

- ☑ Es gibt den Button ‘handleButtonSettingsOnAction‘ SSettings/Ëinstellungen”.
- ☑ Es gibt den Button ‘handleButtonResetOnAction‘ Reset/SZurücksetzen”.
- ☑ Es gibt den Button ‘handleButtonSurrenderOnAction‘ SSurrender/Äufgeben”.
- ☑ Es gibt den Button ‘handleButtonMenueOnAction‘ Main Menue/”Hauptmenü”.
- ☑ Es gibt das Label ‘lblHistory‘ darin werden die ausgeführten Züge aufgelistet. Dieses Label sollte scrollbar sein.
- ☑ Es gibt ein Canvas ‘canvasField‘, das die Musterung des Schachfeldes zeigt.
- ☑ Es gibt die Möglichkeit, die Grafik der aktuellen Aufstellung der Figuren auf dem ‘canvasField‘ zu zeigen.
- ☑ Es gibt das Label ‘lblCheckMessage‘, das im Initialzustand unsichtbar ist. Hier werden Informationen wie Check!/SSchach!öder Check Mate/SSchachmattängezeigt.
- ☑ Es gibt das Label ‘lblInfo‘, das anzeigen soll, welche Farbe am Zug ist. Das muss nicht zwingend ein Label sein, sonder könnte auch als Farbfeld realisiert werden.

### Erstellen GameController

id: #24 Milestone: 2. Iteration

Priorisierung C

Storypoints 13

Risiko high

Der ‘GameController‘ interpretiert die Aktionen des Benutzers, die dieser über die [GameView](#23) ausführt.

**Abgeschlossen wenn**

- ☑ Der Button ‘btnSettings‘ setzt die ‘SettingsView‘ als aktuelle ‘Scene‘.
- ☑ Der Inhalt des Labels ‘lblHistory‘ wird nach jedem Zug entsprechend aktualisiert.
- ☑ Der Button ‘btnMainMenue‘ setzt die ‘MainMenueView‘ als aktuelle ‘Scene‘.
- ☑ Der Button ‘btnReset‘ setzt das Spiel zurück, also ein neues Spiel im aktuellen Modus wird gestartet. Das sollte wahrscheinlich aber im NetworkGame nicht möglich sein.

- ☑ Das Label ‘lblInfo‘ wird im Fall von Schach (wenn das in den Einstellungen so konfiguriert ist) mit dem Text Check!/SSchach!äuf sichtbar gesetzt und beim nächsten Zug aber wieder auf unsichtbar gesetzt.
- ☑ Das Label ‘lblCheckMessage‘ wird im Fall von Schachmatt mit dem Text Check Mate, game over!/SSchachmatt, Spiel vorbei!äuf sichtbar gesetzt.
- ☑ Das Label ‘lblInfo‘ zeigt immer, welche Farbe am Zug ist. Das muss nicht zwingend ein Label sein, sonder könnte auch als Farbfeld realisiert werden.
- ☑ Der ‘GameViewController‘ kann sich von der Klasse ‘BoardGraphics‘ die grafische Umsetzung des aktuellen Aufstellung holen und diese in dem ‘canvasField‘ über der Musterung des Schachfeldes anzeigen. Das muss nach jedem Zug passieren.
- ☑ Der ‘GameViewController‘ achtet darauf ob und wo in das Canvas ‘canvasField‘ geklickt wird. Wenn dort was passiert, verarbeitet er das Event entsprechend.

### Erstellen GraphicsManager

id: #25 Milestone: 2. Iteration

Priorisierung B

Storypoints 8

Risiko high

Die ‘GraphicsManger‘-Klasse Verwaltet die grundsätzliche Verwendung von allen verwendeten Grafiken. Nur auf diesem Weg soll ein Zugriff auf unsere Grafiken möglich sein.

**Abgeschlossen wenn**

- ☑ Andere Klassen können bestimmte Grafiken anfordern.

### Erstellen BoardController

id: #26 Milestone: 2. Iteration

Priorisierung C

Storypoints 21

Risiko high

Die ‘BoardController‘-Klasse kontrolliert, was passiert, wenn auf das Schachfbrett geklickt wird. Sie benutzt dabei ‘BoardModel‘, um relevante Informationen zu erhalten und kann auch Informationen in ‘BoardModel‘ verändern.

**Abgeschlossen wenn**

☑ Aus einem übergebenen Cursor

☑ Speichert im ‘BoardModel‘, welches genaue Feld angeklickt wurde.

☑ Kann das ‘BoardModel‘ fragen, ob bereits ein genaues Feld angeklickt wurde.

☑ Kann das ‘SettignsModel‘ fragen, ob es erlaubt ist, ein bestimmtes Feld anzuklicken.

☑ Kann aus einem ‘Board‘ die Positionen (Koordinaten) für Graphiken bestimmen und das Ergebnis im ‘BoardModel‘ speichern. Das gilt für die Grafiken von den Figuren, für die geschlagenen Figuren und für die Grafiken der Markierungen.

### Erstellen BoardModel

id: #27 Milestone: 2. Iteration

Priorisierung B

Storypoints 8

Risiko high

Hier werden Informationen über das gerade sichtbare Schachbrett gespeichert. Dazu gehören die Positionen von den Grafiken, welche Figur gerade ausgewählt ist etc.

**Abgeschlossen wenn**

☑ speichert ob eine Figur ausgewählt ist.

☑ speichert welche Figur ausgewählt ist.

☑ speichert eine Liste der geschlagenen Figuren als ‘ImageView‘

### Alpha Beta Pruning

id: #28 Milestone: 3. Iteration

Priorisierung A

Storypoints 6

Risiko low

In der Engine soll die Suche nach neuen Zügen via ‘minimax‘-Algorithmus durch alpha-beta-pruning beschläunigt werden.

**Abgeschlossen wenn**

☑ minimax wurde durch alphabetapruning erweitert

☑ Mit GUI getestet

☑ feature switch und minimax Algorithmus entfernen

☑ Resolve pmd violations

### Network Game implementieren

id: #29 Milestone: 3. Iteration

Priorisierung E

Storypoints 21

Risiko high

Das Netzwerkspiel soll in GUI und CLI möglich sein.
**Abgeschlossen wenn**

☑ Zur Kommunikation zwischen zwei Spielen werden die bekannten Befehle aus der CLI genutzt.

☑ Nach einem Zug des Spielers (ggf auch zu Beginn) wird auf einen Zug des Netzwerkgegners gewartet.

☑ r̈esignẅird als Befehl beim Aufgeben oder Abbrechen des Spiels verwendet.

### Speichern/Laden implementieren

id: #31 Milestone: 3. Iteration

Priorisierung

Storypoints

Risiko E

Erstellen die Klassen ‘Saving‘ und ‘SavingManager‘ im Paket ‘util‘. Die Klasse ‘Saving‘ enthält das Spielobjekt und die Historie der Züge. Die Klasse ‘SavingManager‘ speichert und lädt das Spiel.
**Abgeschlossen wenn**

☑ Enthält das Spielobjekt.

☑ Enthält die Historie der Züge._‘SavingManager‘:_

☑ Sucht nach Speicherdateien in einem angegebenen Verzeichnis.

☑ Erzeugt einen Standardnamen basierend auf dem aktuellen Datum.

☑ Erzeugt eine Datei und speichert das Speicherobjekt(‘Saving‘) darin.

☑ Lädt das Spiel aus der ausgewählten Speicherdatei.

### Resizeable GUI Feature implementieren

id: #32 Milestone: 3. Iteration

Priorisierung

Storypoints

Risiko D

Das GUI-basierte Frontend lässt sich jederzeit auf eine beliebige Größe ziehen.
**Abgeschlossen wenn**

☑ Reagieren auf Änderungen der Größe des Spielfensters.

☑ Aktualisieren die Größe von Text, Symbolen und anderen Elementen, wenn die Größe des Fensters geändert wird.

### Zweisprachigkeit implementieren

id: #34 Milestone: 3. Iteration

Priorisierung

Storypoints

Risiko V

Von jeder Stelle im Spiel aus kann zwischen zwei Sprachen gewechselt werden.
**Abgeschlossen wenn**

☑ Erstellen ‘TextManager‘ Klasse.

☑ Fügen .properties

☑ Übernehmen den Text aus der gewünschten .proteries

☑ Ändern das Locale.

# 4 Anhang

## 4.1 Tabellarische Anforderungsanalyse

| Klassifikation | Priorität | Kategorie | Kriterium | Iteration |
|---|---|---|---|---|
| funktional | must | Hauptfunktion | gegen eine KI spielen | 2 |
| funktional | must | Hauptfunktion | gegen einander spielen | 1 |
| funktional | must | Schnittstelle | graphisch | |
| funktional | must | Schnittstelle | konsolenbasiert | 1 |
| funktional | must | Sprache | Englisch | |
| funktional | must | Züge | nicht zulassen ungültiger Züge | 1 |
| funktional | must | GUI | 2D | 2 |
| funktional | should | Features | Verbesserte KI mithilfe Min-/Max-Suche mit α/β\alpha/\betaαα/β-Pruning (5) | 3 |
| funktional | should | Features | 3D-GUI (5) | 3 |
| funktional | should | Features | Eindimensionales Schach (5) | 3 |
| funktional | should | Features | Rückgängig-Machen von Zügen (3) | 3 |
| funktional | should | Features | Speichern/Laden von Spielen (3) | 3 |
| funktional | should | Features | Schachuhren (2) | 3 |
| funktional | should | Features | Zweisprachigkeit (2) | 3 |
| funktional | should | Features | Resizeable GUI (1) | 3 |
| funktional | must | Hauptfunktion | Netzwerkspiel | 3 |
| nicht-funktional | must | Codequalität | Einhaltung der Metriken | |
| nicht-funktional | must | Codequalität | Einhaltung Style-Convention | |
| nicht-funktional | should | Codequalität | Kommentierung des Codes | |
| Nebenbedingung | must | Testabdeckung | Abdeckung von 90 % des Codes (ohne GUI Klassen) durch JUnit-Tests | |
| nicht-funktional | must | Dokumentation | Stets aktualisierte Anforderungsdokumentation via Story-Cards | |
| nicht-funktional | must | Dokumentation | Stets aktualisierte Dokumentation der Architektur unter Zuhilfenahme von UML-Diagrammen (Klassen-, Objekt-, Sequenz-, Zustand-) | |
| nicht-funktional | must | Dokumentation | Stets aktualisierte Javadoc-Dokumentation des Programms | |
| nicht-funktional | must | Dokumentation | Stets aktualisierte Dokumentation der Programmverwendung (Bedienungsanleitung) | |
| Nebenbedingung | must | Sprache | Java, Version 11 | |
| Nebenbedingung | must | Bibliothek | JavaFX | |
| Nebenbedingung | could | Bibliothek | externe Bibliotheken für die Umsetzung 3D-GUI | |
| Nebenbedingung | must | Version | Versionsverwaltung in Git | |
| Nebenbedingung | must | Build-Managment | Maven | |