Schach

Modul: IN212 OOP2  
Autor: Sebastian Berchtold

Stefan Hafner

Version: 1.0

Datum 02.03.2024

Inhaltsverzeichnis

[1 Applikations Übersicht 1](#_Toc160288510)

[1.1 Layout 1](#_Toc160288511)

[2 Technische umsetzung der Themen 4](#_Toc160288512)

[2.1 Enum 4](#_Toc160288513)

[2.2 Abstrakte Klasse 5](#_Toc160288514)

[2.3 Vererbung 5](#_Toc160288515)

[2.4 Collections 6](#_Toc160288516)

[2.5 Serialisierung 7](#_Toc160288517)

[2.6 Logger 8](#_Toc160288518)

[2.7 Unit Test 9](#_Toc160288519)

[2.8 Exceptions 9](#_Toc160288520)

[3 Fazit 9](#_Toc160288521)

# Applikations Übersicht

## Layout

Layout:

* Wir haben uns für ein schlankes Layout entschieden, das mit dem Scene Builder erstellt wurde.
* Das Layout wurde mit einem AnchorPane erstellt und hat eine feste Größe.
* Für das Schachbrett verwenden wir ein GridPane, in das für jedes Feld ein StackPane erstellt wird, um Zugriff auf jedes Feld zu erhalten.
* Die richtigen Schachkoordinaten wurden auf der Seite eingebaut.

Funktionalitäten:

* Benutzer können Spielnamen speichern.
* Ein neues Spiel kann durch Drücken der Schaltfläche "New Game" gestartet werden, wobei Weiß am Zug ist.
* Mit "Save Game" ist es möglich, einen Zwischenstand zu speichern, und mit "Load Game" kann ein Spiel geladen werden.
* Um zu verhindern, dass versehentlich ein neues Spiel gestartet wird, haben wir eine Schaltfläche "Reset" eingebaut. Diese macht den "New Game" Button wieder sichtbar für die Bedienung.

Ein Bild, das Screenshot, Text, Quadrat, Schach enthält.

Automatisch generierte Beschreibung

Durch Drücken von "Load Game" oder "Save Game" öffnet sich ein FileChooser, der direkt auf den Speicherort navigiert, an dem das Programm ausgeführt wird.

Ein Bild, das Text, Computer, Software, Screenshot enthält.

Automatisch generierte Beschreibung

Zusätzlich zu den Schachkoordinaten haben wir dickere Linien implementiert, um deutlich zu machen, welcher Spieler am Zug ist. Durch diese visuelle Kennzeichnung ist es für die Benutzer leichter erkennbar, welcher Zug als nächstes gemacht werden muss. Dies trägt zur Benutzerfreundlichkeit und zur besseren Verständlichkeit des Spiels bei.

Ein Bild, das Screenshot, Schachfigur, Cartoon, Schach enthält.

Automatisch generierte Beschreibung

Um die Benutzererfahrung beim Bewegen der Figuren zu verbessern, haben wir einen Glow-Effekt implementiert. Wenn eine Figur ausgewählt wird, werden alle möglichen Schritte mit einem sanften Glow-Effekt hervorgehoben. Dieser Effekt macht die Felder, auf die die Figur navigiert werden kann, etwas heller, was die Orientierung und das Verständnis für mögliche Züge erleichtert. Diese visuelle Rückmeldung trägt dazu bei, dass die Spieler schnell und intuitiv erkennen können, wohin sie ihre Figuren bewegen können.  
  
Ein Bild, das Cartoon, Schachfigur enthält.

Automatisch generierte Beschreibung

Im Falle eines Gewinns wird eine große Benachrichtigung über den Bildschirm angezeigt, die den Sieg des Spielers verkündet. Nachdem diese Nachricht erscheint, wird die Bewegung von Figuren deaktiviert, um das Spiel zu beenden. Um ein neues Spiel zu starten oder ein gespeichertes Spiel zu laden, muss der Benutzer den Reset-Button drücken und anschließend entweder "New Game" wählen oder ein zuvor gespeichertes Spiel laden. Diese klare Trennung zwischen einem abgeschlossenen Spiel und dem Beginn eines neuen Spiels sorgt für eine reibungslose Spielerfahrung und verhindert versehentliche Aktionen nach dem Spielende.

Ein Bild, das Screenshot, Quadrat, Text enthält.

Automatisch generierte Beschreibung

# Technische umsetzung der Themen

## Enum

Wir haben zwei Enum-Klassen in unserem Projekt implementiert:

**EColorPiece**

Diese Enum-Klasse enthält lediglich zwei Farben, Schwarz und Weiß. Durch die Verwendung dieser Enum-Klasse können wir im Code eine klare Unterscheidung zwischen den beiden Farben treffen und sie für die Zuordnung von Figuren nutzen.

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

Automatisch generierte Beschreibung

**Coordinates**

Diese Enum-Klasse enthält alle Koordinaten eines Schachbretts sowie Methoden, um von den Koordinaten auf dem GridPane zu den Schachkoordinaten zu gelangen. Dies ermöglicht beispielsweise eine einfache Ausgabe per Logger, um anzuzeigen, wo sich welche Figur sich hin bewegt.

Die Verwendung von Enum-Klassen bietet eine übersichtliche und strukturierte Möglichkeit, verschiedene Konzepte wie Farben und Koordinaten im Code zu repräsentieren und zu nutzen.  
Ein Bild, das Screenshot, Text, Multimedia-Software, Software enthält.

Automatisch generierte Beschreibung

## Abstrakte Klasse

Wir haben eine abstrakte Klasse namens "Piece" implementiert, um die gemeinsamen Funktionen aller sechs verschiedenen Figuren im Schach zu vereinen. Die "Piece"-Klasse erbt von der "ImageView"-Klasse, was es uns ermöglicht, das Bild der Figur direkt auf ein Piece zu setzen. Außerdem implementiert sie das Serializable-Interface, um das Speichern oder Laden von Spielen zu ermöglichen.

Durch die Verwendung dieser abstrakten Klasse können wir Code-Duplizierung vermeiden und eine gemeinsame Basis für alle Schachfiguren schaffen. Jede spezifische Schachfigur kann dann von der "Piece"-Klasse erben und individuelle Eigenschaften und Verhaltensweisen implementieren, während sie gleichzeitig auf die gemeinsamen Funktionen und Eigenschaften der "Piece"-Klasse zugreifen kann.

Ein Bild, das Text, Screenshot, Software, Display enthält.

Automatisch generierte Beschreibung

## Vererbung

In unserem Schachspielprojekt nutzen wir eine Vererbungshierarchie, um die Struktur und Funktionalität des Spiels zu organisieren:

* Die abstrakte Klasse "Piece" dient als Hauptvererbungsklasse, von der alle spezifischen Schachfiguren erben. Sie erbt ihrerseits von der "ImageView"-Klasse, um das Bild der Figur zu repräsentieren, und implementiert das "Serializable"-Interface für das Speichern und Laden.
* Die "Square"-Klasse erbt von "StackPane", um die einzelnen Felder des Schachbretts zu repräsentieren. Diese Klasse wird verwendet, um das Layout der Schachbrett-Felder zu definieren.
* Die "Board"-Klasse erbt von "GridPane" und wird verwendet, um das Schachbrett als Ganzes darzustellen. Sie enthält Logik zur Platzierung und Verwaltung der Schachfiguren auf dem Brett.
* Die "App"-Klasse erbt von "Application" und stellt den Einstiegspunkt für die JavaFX-Anwendung dar. Sie enthält die Hauptmethode "start", die die Benutzeroberfläche initialisiert und startet.

## Collections

In unserem Schachspielprojekt nutzen wir Collections an zwei Hauptstellen:

**Board-Klasse**

Hier werden alle Felder (Squares) des Schachbretts in eine ArrayList geladen. Diese ArrayList wird in der Methode `initializeBoard(GridPane board)` befüllt. Für jedes der 64 Felder wird ein Square angelegt und entsprechend die Farbe gesetzt. Diese ArrayList ermöglicht eine einfache Verwaltung und Manipulation der Felder des Schachbretts während des Spiels.

**Figuren:**

Insbesondere bei Figuren wie dem Turm, Läufer und der Dame, die keine begrenzte Reichweite haben, nutzen wir eine ArrayList, um alle spielstandabhängigen möglichen Züge zu ermitteln. Da die Länge des Arrays je nach Position der Figur und den Hindernissen auf dem Brett variieren kann, ist die ArrayList hier optimal geeignet. Sie ermöglicht es, die möglichen Züge dynamisch zu verwalten und zu aktualisieren, was für die Implementierung der Zuglogik entscheidend ist.

Ein Bild, das Text, Screenshot, Software enthält.

Automatisch generierte Beschreibung

## Serialisierung

Neuer Abschnitt:

Die „SaveGame“-Klasse in unserem Schachspielprojekt dient zur Speicherung aller notwendigen Informationen eines Spielstands. Durch die Implementierung des „Serializable“-Interfaces gestaltet sich die Speicherung sehr einfach. Diese Klasse verwendet ein 2D-Array, das alle Positionen der Figuren enthält, sowie die Namen der Spieler. Zusätzlich verfügt die Klasse über eine Methode, um bei einem neuen Spiel die Standardaufstellung zu erstellen.

Im „GameController“ gibt es zwei Methoden, „loadGame()“ und „saveGame()“, die für das Laden und Speichern des Spielstands verantwortlich sind. Diese Methoden verwenden die Serialisierung bzw. Deserialisierung, um den Spielstand zu sichern oder wiederherzustellen. Dabei wird über einen „FileChooser“ entweder der Speicherort und Dateiname ausgewählt oder die zu ladende Datei. Dadurch wird das Spielgeschehen auf einfache und benutzerfreundliche Weise geladen oder gespeichert.

Hier ist ein Beispiel für die „loadGame()“-Methode im „GameController“:

Ein Bild, das Text, Screenshot, Software enthält.

Automatisch generierte Beschreibung

## Logger

Im Rahmen unseres Schachspielprojekts haben wir den Logger intensiv genutzt, um eine Spielhistorie zu erstellen. Dabei verfolgten wir die Idee, dass die Spielhistorie ähnlich wie im klassischen Schach protokolliert wird. Das bedeutet, dass jeder Zug aufgezeichnet wird, zum Beispiel "Der Bauer von C1 wurde auf C2 gesetzt". Zusätzlich werden Informationen wie die Farbe der Figuren oder geschlagene Figuren protokolliert.

Des Weiteren wird der Logger verwendet, um Aktionen wie das Starten eines neuen Spiels, das Speichern oder Laden eines Spiels zu dokumentieren. So behält der Spieler stets den Überblick über die Spielaktionen und den Spielverlauf.

Ein besonderer Einsatz des Loggers erfolgt bei einem Schach-Matt. In diesem Fall wird eine Warnung ausgegeben, um den Spieler darauf aufmerksam zu machen, dass das Spiel zu Ende ist. Durch die Verwendung des Loggers können Spieler leichter den Spielverlauf nachverfolgen und das Spielgeschehen besser verstehen.  
Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Neuer Abschnitt:

Um die Spielhistorie ansprechend darzustellen, haben wir die "logging.properties "-Datei verwendet und dem "SimpleFormatter " ein spezifisches Format zugewiesen. Das Format lautet:

java.util.logging.SimpleFormatter.format=[%1$tF %1$tT] [%4$-7s] %5$s %n

Dieses Format legt fest, wie die Protokollmeldungen formatiert werden sollen. Es enthält ein Zeitstempel, den Log-Level, den Log-Namen und die eigentliche Nachricht.

Der Logger wird nur in der `Game`-Klasse und dem "GameController " verwendet. In der "Game"-Klasse wird der Logger in der Methode "movePiece()" verwendet, während im "GameController" die Logger in den Methoden "createBoard()", "saveGame()" und "loadGame()" zum Einsatz kommen.

Durch die Verwendung des Loggers können wir den Spielverlauf aufzeichnen und den Spielern ermöglichen, die Aktionen und Ereignisse im Spiel nachzuvollziehen. Das definierte Format hilft dabei, die Protokollmeldungen übersichtlich und leicht verständlich zu gestalten.

## Unit Test

Wir haben für die wichtigsten Funktionen einzelne Tests verfasst, um sicherzustellen, dass unsere Software korrekt funktioniert. Die Klasse "Game" wird auf ihre Einrichtung überprüft sowie auf die möglichen Bewegungen und die korrekte Ausführung des Schlagens von Figuren. Das Enum "Coordinates" wird daraufhin geprüft, ob die Koordinaten korrekt hin und her übersetzt werden können. In der Klasse "Square" wird überprüft, ob ein Quadrat von einer Figur besetzt wird und ob das Setzen des Namens funktioniert. Bei der "Save Game"-Funktion wird die vollständige Startaufstellung überprüft, um sicherzustellen, dass diese immer korrekt ist. Es darf nicht vorkommen, dass eine Figur plötzlich an der falschen Stelle platziert wird.

## Exceptions

Es scheint, dass nur zwei Klassen tatsächlich verwendet werden:

* Coordinates

In der Klasse Coordinates wird eine Ausnahme (Exception) ausgelöst, wenn das Argument beim Übersetzen der Koordinaten nicht existiert. Dies stellt sicher, dass ungültige Eingaben abgefangen werden.

* GameController

In der Methode saveGame() des GameController werden Ausnahmen durch einen Try-Catch-Block abgefangen, falls currentGame = null ist oder während des Speicherns oder Ladens des Spielstandes ein Fehler auftritt. Ebenso wird eine Ausnahme abgefangen, wenn beim Speichern der Namen currentGame = null ist. Die ObjectOutputStream wird im Finally-Block geschlossen, um sicherzustellen, dass Ressourcen ordnungsgemäß freigegeben werden. Die loadGame()-Methode verwendet die AutoClose-Funktion, um sicherzustellen, dass Ressourcen automatisch freigegeben werden.

# Fazit

Insgesamt war es eine äußerst interessante Aufgabe. Wir haben festgestellt, dass die Entwicklung der Spiellogik eine anspruchsvolle und zeitintensive Aufgabe darstellt, was uns dazu veranlasst hat, uns hauptsächlich darauf zu konzentrieren. Dadurch blieb weniger Zeit für die Implementierung umfangreicher Unit-Tests und die Bearbeitung anderer Aufgabenbereiche. Dennoch war es spannend, die Spiellogik zu entwickeln und erste Erfolge zu erzielen.

Für eine frühe Entwicklerversion bin ich mit dem aktuellen Stand zufrieden. Es gibt jedoch noch einige Möglichkeiten zur Verbesserung. Dazu gehört unter anderem die Implementierung weiterer Sonderzüge sowie die Verbesserung des Designs, um das Spielerlebnis zu optimieren und das Spiel ästhetisch ansprechender zu gestalten.