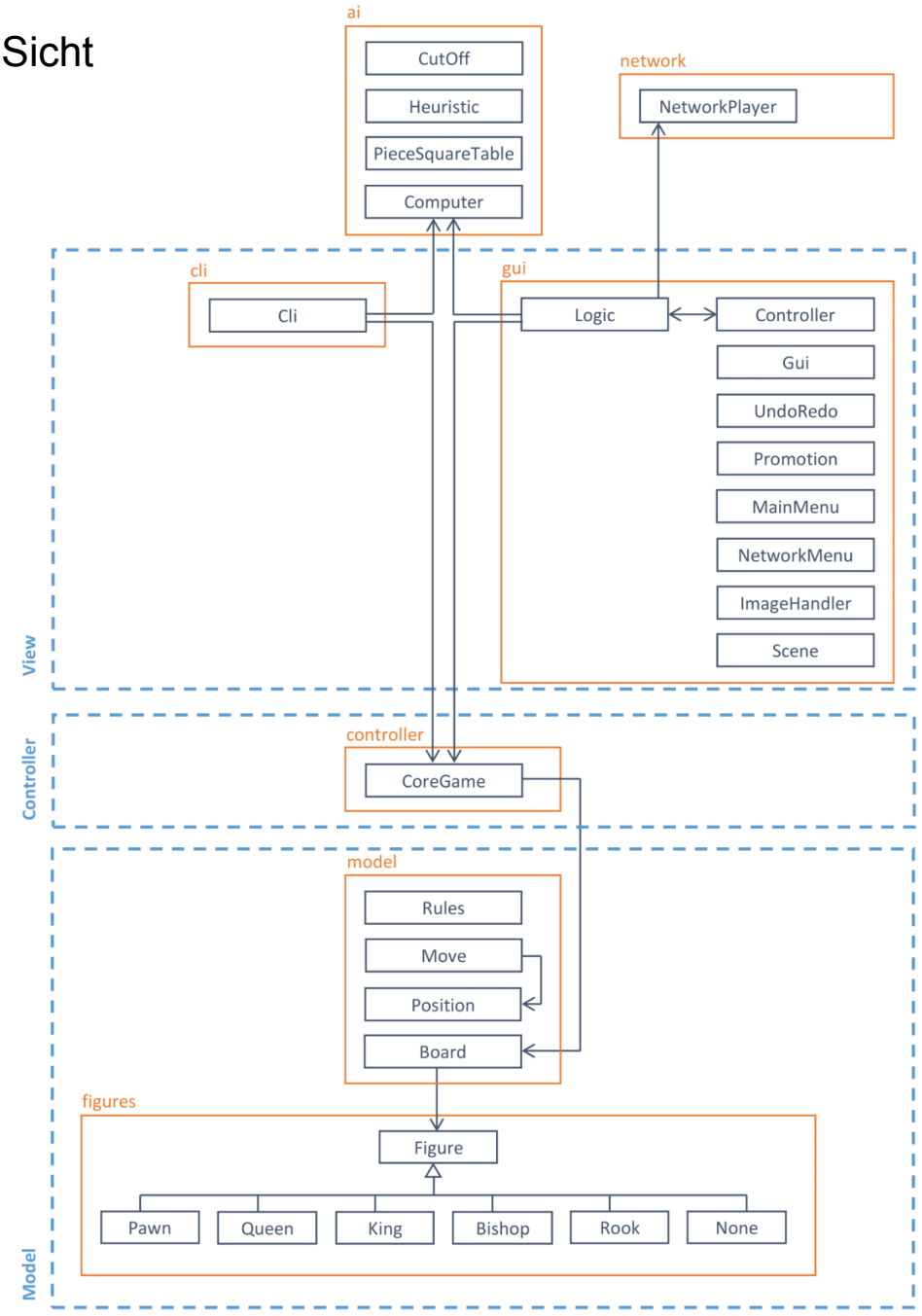
Architektur Dokumentation

Nutzersicht

siehe Anforderungsdokumentation[1] (Anwendungsfalldiagramm und Story Cards)

Statische Sicht



UML-Klassendiagramm[2]

[1] https://projects.isp.uni-luebeck.de/gruppe-10/schach/-/blob/main/documentation/1_Anforderungsdokumentation.pdf
[2] https://projects.isp.uni-luebeck.de/gruppe-10/schach/-/blob/main/images/Klassendiagramm.png
Dieses Diagramm entspricht nicht den Regeln des UML-Klassendiagramms, sondern wurde so vereinfacht, dass die Klassen und ihre Beziehungen untereinander gut ersichtlich sind.
Für ein vollständiges UML-Klassendiagramm siehe
https://projects.isp.uni-luebeck.de/gruppe-10/schach/-/blob/main/images/KlassendiagrammVollstaendig.png

Wie im Klassendiagramm sichtbar, orientieren wir uns an dem Architektur-Pattern der Schichtenarchitektur und dem Design-Pattern MVC (Model-View-Controller).

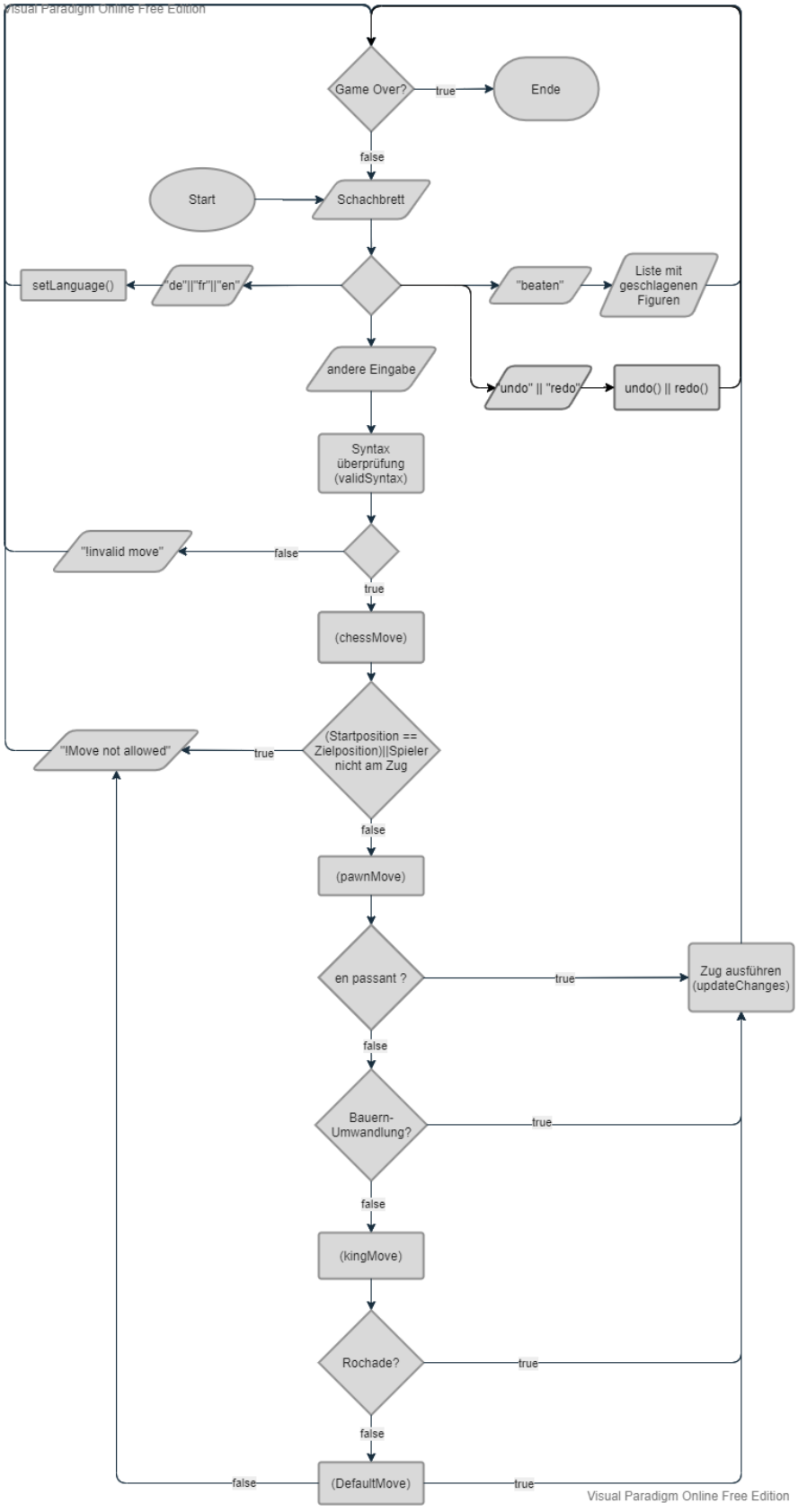
- View: Die Klassen bzw. Packages CLI und GUI übernehmen dabei die Aufgabe der Darstellung nach außen auf der Konsole bzw. als grafische Schnittstelle.
- Model: Die Klassen Position, Move, Board, Parser und Rules bilden die Spiellogik. Die Klasse Board hat zudem Zugriff auf die einzelnen Figuren.
- Controller: Die Klasse CoreGame übernimmt die Kontrolle und definiert das Verhalten des Spiels. Zudem vermittelt diese Klasse zwischen View und Model.

Durch diese Trennung der Komponenten ist es vergleichsweise einfach möglich, Erweiterungen zu realisieren, z.B. eine weitere View-Klasse hinzuzufügen, da die Spiellogik von der Ausgabe getrennt ist. Dadurch ist das Programm relativ unkompliziert erweiterbar und wartbar. Im Laufe des Projektes wurde die Wahl dieses Patterns auch sinnvoll, um Aufgabenbereiche aufzuteilen und so eine möglichst effiziente und agile Entwicklung zu ermöglichen. So haben sich zwei Mitglieder der Gruppe in der zweiten Iteration um die grafische Darstellung nach außen und zwei um die Realisierung eines Computer-Gegners kümmern können, ohne sich gegenseitig in der Entwicklung zu behindern.

Dynamische Sicht

CLI

Der Programmablauf ist im Folgenden für das Spiel auf der Konsole skizziert:



[Programmablaufplan CLI]

Grundlegender Ablauf in der Konsole:
Sobald das Spiel gestartet wird, wird in einer while-Schleife die Eingabe abgefragt und darauf reagiert. Bei einer Schachzug-Eingabe prüft das System, ob die Syntax korrekt war, dann ob der Zug semantisch gültig ist und führt anschließend den entsprechenden Zug aus. Das Spielfeld wird aktualisiert und das Spiel wartet auf eine erneute Eingabe des Spielers.
Bei einer anderen Eingabe wie beispielsweise “beaten” oder einer Spracheingabe wie “en”, “de” oder “fr” für ein Spiel auf Englisch, Deutsch oder Französisch, werden die geschlagenen Figuren, die in einer Array-Liste gespeichert werden, ausgegeben, oder mithilfe des Sprachmanagers die Sprache des Spiels geändert.
Zusätzlich sind als Zusatzfeature die Eingaben “undo” und “redo” gültig, die Züge rückgängig machen und wiederherstellen können. Hierzu werden die ausgeführten Züge in der Zug-Historie abgefragt und die entsprechenden Spielfelder wiederhergestellt.

GUI

In der GUI ist die Ein- und Ausgabe entsprechend angepasst, die interne Spiel-Logik bleibt aber dieselbe, da auch hier CoreGame gültige Eingaben prüft und Züge ausführt. Allein die Darstellung erfolgt nun über eine grafische Oberfläche.

Netzwerkspiel

Beim Netzwerkspiel haben wir uns für eine Client-Server-Architektur entschieden. Ein Spieler startet das Spiel und bildet somit den Server, ein zweiter Spieler tritt dem Netzwerkspiel als Client bei.
Beide kommunizieren über den vorher festgelegten Port 5555, wobei der Spieler, der den Server stellt, diesen Port freigeben muss, damit der Client-Spieler darauf zugreifen kann. Der Server-Spieler muss dem Client-Spieler entsprechend (extern) seine IP-Adresse mitteilen. Zudem legt der Server-Spieler zu Anfang fest, mit welcher Farbe gespielt werden soll. Die Farbe des Client-Spielers wird dem Client als “black” oder “white” übermittelt. Haben beide nun das Spiel eingerichtet beginnt der als weiß spielende mit dem ersten Zug.
Es wird davon ausgegangen, dass alle gesendeten Züge, sowohl vom Client als auch vom Server, sowohl syntaktisch als auch semantisch korrekt sind. Dies wird im jeweiligen Controller vor dem Verschicken der Züge überprüft.

Die Kommunikation läuft nun wie folgt ab: