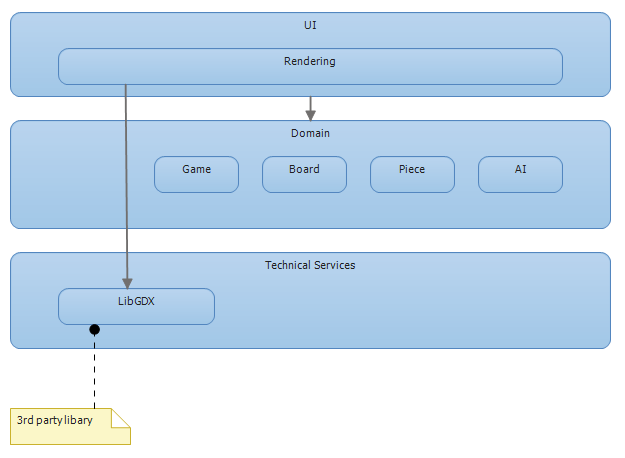
Design

## Projektmanagement

## Architektur

Wir haben uns entschieden necaREx mit Java zu entwickeln. Dies hat den Vorteil, dass wir mit sehr wenig Aufwand das Programm Plattformunabhängig entwickeln und anbieten können. Zusätzlich werden wir für das Frontend die Library „libgdx“ verwenden, diese bietet einfach zu verwendende Schnittstellen um 2d Anwendungen auf der Grafikkarte auszuführen. Ausserdem wird sie als zusätzlichen Abstraktionslayer im UI verwendet, anhand welchem wir, in einem nächsten Release, die Anwendung ebenfalls als iOS und Android App anbieten können.

Um eine möglichst flexible und erweiterbare Lösung aufzubauen haben wir uns für folgendes Layering entschieden:

Erläuterungen zu den von uns entwickelten Packages:

* Rendering

Hier wird das 2d Userinterface gerendert. Mit Hilfe der LibGDX Funktionalitäten werden hier die nötigen Informationen aus den Models gelesen und dargestellt.

* Game

Das Game verwaltet die Spieler, Züge und getöteten Figuren.

* Board

Hier geschieht die Verwaltung des Schachbretts. Es wird Logik zur Verfügung gestellt, um Figuren zu bewegen, Figuren zu töten und alle Figuren auf die Initialposition zu setzen.

* Piece

Die einzelnen Figuren stellen Logik zur Verfügung um ihre möglichen Bewegungen zu berechnen.

* AI

Logik für die künstliche Intelligenz.

## Design-Klassendiagramme

## Klassenverantwortlichkeiten

Im folgenden Abschnitt werden die Verantwortlichkeiten und Aufgaben der einzelnen Klassen kurz beschrieben.

### Package client.UI

|  |  |
| --- | --- |
| Klassenname | Verantwortlichkeiten / Aufgaben |
| NecarexDesktop | Einstiegsklasse für Desktop Anwendungen. |
| NecarexGame | LibGDX Main Klasse, übernimmt das Draw/Update. |

### Package client.UI.drawing

|  |  |
| --- | --- |
| Klassenname | Verantwortlichkeiten / Aufgaben |
| BoardDrawer | Der BoardDrawer übernimmt alle Zeichnungsaufgaben bezogen auf das Schachbrett. |
| PieceDrawer | Der PieceDrawer übernimmt alle Zeichnungsaufgaben bezogen auf die Schachfiguren. |

### Package client.viewmodel

|  |  |
| --- | --- |
| Klassenname | Verantwortlichkeiten / Aufgaben |
| ChessBoardViewModel | Model mit View-spezifischen Daten für das Schachbrett wie zum Beispiel das zuletzt vom Benutzer angeklickte Feld. |

### Package control

|  |  |
| --- | --- |
| Klassenname | Verantwortlichkeiten / Aufgaben |
| GameController | Kontrolliert den Ablauf des Schachspiels |

### Package model

|  |  |
| --- | --- |
| Klassenname | Verantwortlichkeiten / Aufgaben |
| ChessBoard | Das ChessBoard kontrolliert die Schachfelder und macht die Anfangsaufstellung zu Beginn des Spiels. |
| ChessField | Auf einem Schachfeld (ChessField), das auf einer Spalte und einer Zeile steht, kann eine Figur stehen. |
| ComputerPlayer | Stellt einen Computerspieler dar. Benutzt für die Zugberechnung das Package model.ai |
| Game | Ein Spiel (Game) besteht aus weissem und schwarzem Spieler, sowie einer Referenz des Schachbretts. Die Liste der vergangenen Züge und die bisher geschlagenen Figuren werden hier geführt. |
| Player | Ein Schachspieler (Player) besitzt eine Farbe. |
| PlayerColor | Enumeration für die beiden Figurenfarben im Schach: Weiss und Schwarz |
| Turn | Ein Zug (Turn) im Schach besteht aus einer Figur, die von einem Feld zu einem anderen zieht und dort allenfalls eine andere Figur schlägt. Die Attribute werden über den Konstruktor gesetzt, danach sind keine Änderungen mehr möglich, da ein gemachter Spielzug nicht geändert werden kann. |
|  |  |

### Package model.ai

|  |  |
| --- | --- |
| Klassenname | Verantwortlichkeiten / Aufgaben |
| PossibleSituation | Stellt eine mögliche Schachbrettstellung dar. Diese Klasse nimmt die Bewertung der Schachsituation vor. |
| PossibleSituationTree | Stellt eine Baumstruktur für die möglichen Schachbrettstellungen (PossibleSituation) in der Zukunft dar und führt die Rekursion durch. |

### Package model.pieces

|  |  |
| --- | --- |
| Klassenname | Verantwortlichkeiten / Aufgaben |
| Piece | Die Klasse Piece (abstrakt) beschreibt alles, was alle Schachfiguren gemeinsam haben. Alle Schachfiguren gehören einem Owner, stehen auf einem Schachfeld.Jede Schachfigur muss eine Methode implementieren, die die möglichen Felder zurück gibt. |
| Pawn | Spielfigur Bauer, basiert auf Piece. Kennt die Gangart des und kann dessen mögliche Felder ausrechnen. |
| Knight | Spielfigur Springer, basiert auf Piece. Kennt die Gangart des und kann dessen mögliche Felder ausrechnen. |
| Bishop | Spielfigur Läufer, basiert auf Piece. Kennt die Gangart des und kann dessen mögliche Felder ausrechnen. |
| Rook | Spielfigur Turm, basiert auf Piece. Kennt die Gangart des und kann dessen mögliche Felder ausrechnen. |
| Queen | Spielfigur Dame, basiert auf Piece. Kennt die Gangart des und kann dessen mögliche Felder ausrechnen. |
| King | Spielfigur König, basiert auf Piece. Kennt die Gangart des und kann dessen mögliche Felder ausrechnen. |

### Test

Die für den Test verwendeten Klassen werden hier nicht genauer spezifiziert.

## Zusammenarbeitsdiagramme

## Glossar

## GUI-Design