SOFTWARE ARCHITECTURE DESCRIPTION

for

MELT Chess

Version 0.1

17. Juni 2021

Inhaltsverzeichnis

1 Einführung

Die Applikation MELT-Chess unterteilt sich in die Module

- `model`: für die Implementierung der Datenstrukturen und Schachregeln
- `cli`: für die Implementierung der rudimentären Konsolenschnittstelle mit eingeschränktem Umfang der Features[1].
- `gui`: für die Implementierung der grafischen 2d-Schnittstelle.
- `engine`: für die Implementierung der simplen Schach Engine

Im Folgenden wird nun der Aufbau dieser vier Module erläutert.

[1] Siehe Anforderungen.pdf Dokument

2 model

Das `model` Paket teilt sich im wesentlichen in zwei Arten von Klassen: Zum einen die Datenstrukturen `Piece, Move, Board` und die Implementierung der Schach-Regeln[1] in den Klassen `MoveGenerator und MoveValidator`. Die Klasse `Game` verpackt obige Klassen noch einmal für den einfachen Zugriff durch die Clients.

2.1 Piece

Eine Figur wird vollständig durch einen Integer Wert beschrieben, der sich aus $Typ + Farbe$ zusammensetzt. Dazu sind in der Klasse `Piece` folgende konstante Felder definiert:

Typ	Wert
None	0
King	1
Pawn	2
Knight	3
Bishop	4
Rook	5
Queen	6
White	8
Black	16

Weiter bietet die Klasse ein paar statische Helfermethoden, eine komplette Auflistung befindet sich in der Java Documentation.

Das Erweitern um mögliche neue Spielfiguren ist "etwas umständlich", da die Flags mit Hinblick auf eine Implementierung durch `byte` anstelle von `int` geplant wurden. Soll mehr als eine neue Figur eingeführt werden, müssen daher die Flags für die Farben sowie die entsprechenden Bitmasken angepasst werden.

2.2 Move

Die Klasse `Move` ist eine simple Kapselung für die drei Felder:
`Move(int startSquare, int targetSquare, int flag)`,
wobei der `flag`-Parameter optional ist. Für einen Zug sind folgende mögliche Flags definiert:

[1] gemäß den FIDE-Regeln von 2018

Typ	Wert
None	0
EnPassantCapture	1
Castling	2
PromoteToQueen	3
PromoteToKnight	4
PromoteToRook	5
PromoteToBishop	6
PawnTwoForward	7

2.3 `Board`

Die Klasse `Board` liefert eine vollständige Beschreibung einer Position auf dem Schachbrett. Dies ist notwendig damit Instanzen von `Board` dem Spielverlauf hinzugefügt, und somit die Funktionalität "Rückgängig machen" implementiert werden kann.

`Board` merkt sich die Spielfiguren in einem `int[64]` Array, sowie in einer `List<Integer>` für die geschlagenen Figuren.

Eine wichtige Methode ist `Board.makeMove(Move move)`, welche eine neue Instanz von `Board` zurückgibt in der `move` ausgeführt wurde. Die Methode beachtet dabei keine der Schachregeln, setzt allerdings beim Bewegen von Figuren die an der Rochade-Regel teilnehmen (also die vier Türme sowie beide Könige) ein entsprechendes Flag das die Rochade verhindert[2].

2.4 `Coordinate`

Die statische Klasse `Coordinate` liefert einfache Wege um die verschiedenen Repräsentationen von Brettpositionen (zB "b7" $\leftrightarrow$ 9 $\leftrightarrow$ $\{1, 1\}$) zu konvertieren.

2.5 `MoveGenerator`

Die `MoveGenerator` Klasse mit den statischen Auslagerungen implementieren die Regeln zum Bewegen *und nur zum Bewegen* einer Figur[3].

Von außen betrachtet sind zwei Methoden interessant:

`List<Move> generateMoves()`
gibt alle möglichen Züge zu gegebener Spielposition zurück.

`List<Move> generateMovesStartingAt(int position)`
gibt alle möglichen Züge zu gegebener Spielposition zurück, die vom Feld `position` ausgehen.

[2] Damit wird jedoch keine Regel umgesetzt, die Flags dienen lediglich zum frühzeitigen Abbrechen bei der Generierung von Zügen

[3] Das Bedeutet der `MoveGenerator` liefert auch regelwidrige Züge!

Für den Fall dass sich die Anforderungen an die Regeln zum Bewegen von Schachfiguren ändern, muss ledliglich von einer dieser Klasse geerbt und gewünschte Regeln überschrieben werden.

2.6 `MoveValidator`

Der `MoveValidator` ist eine rein statische Klasse, die zu gegebener Spielposition und einem Zug überprüft ob der Zug Regelkonform ist. Da die Bewegungsregeln bereits in `MoveGenerator` implementiert sind, wird zunächst überprüft ob der Zug von ihm gefunden wurde. Danach werden noch die folgenden Regeln überprüft:

- Der König steht nach ausführen des Zugs nicht im Schach
- Falls der Zug eine Rochade ist, überprüfe das keines der betroffenen Felder angegriffen wird.
- Ist der König im Schach und es existiert kein valider Zug nachdem der König nicht mehr im Schach steht, ist das Spiel verloren: SCHACHMATT!
- Falls kein valider Zug möglich ist, gibt es ein remis: PATT![4]

2.7 `Game`

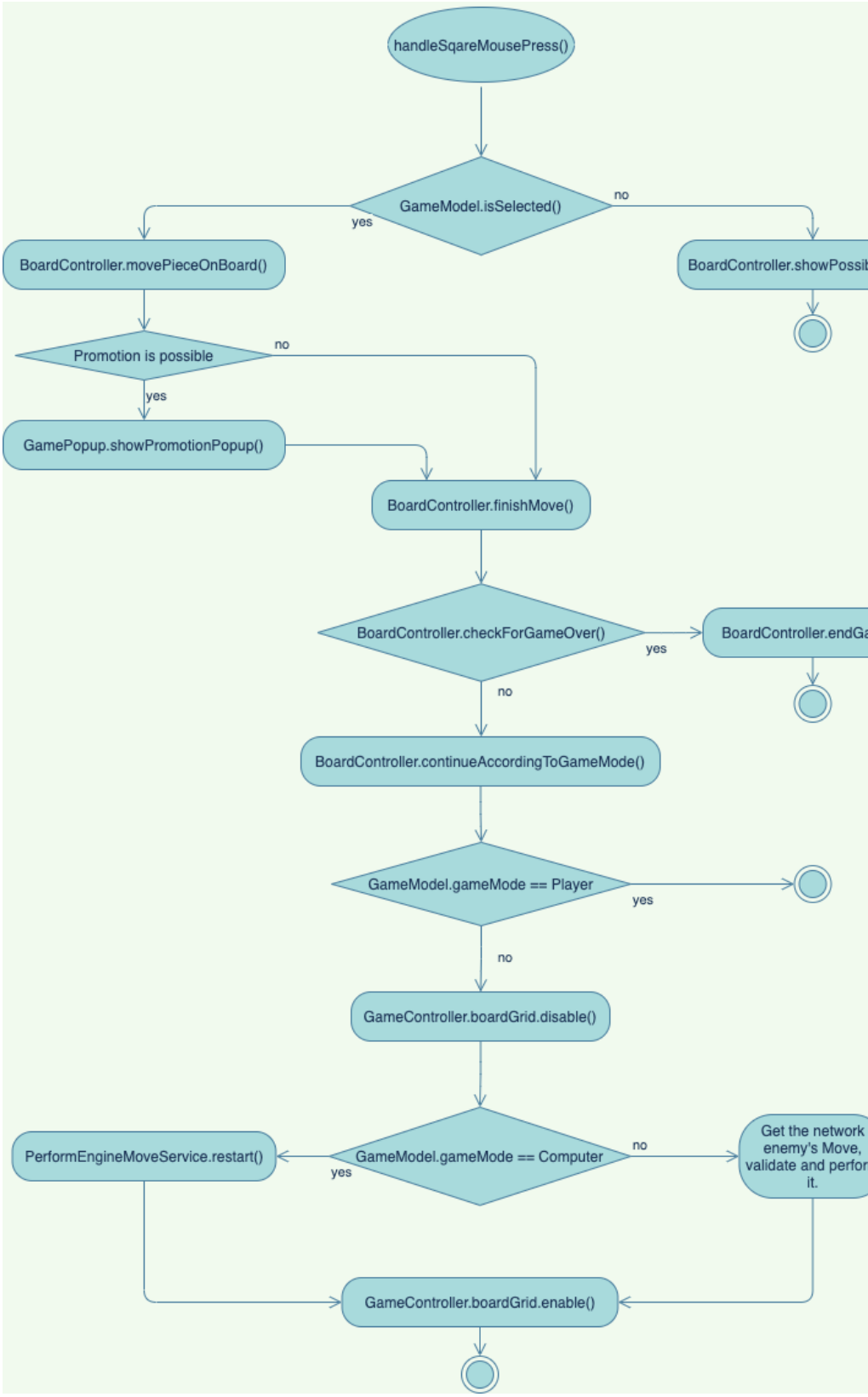
Die Klasse `Game` bietet ein einfaches, übersichtliches Interface zum ausführen eines Zugs und dem erhalten von Informationen der aktuellen Partie für die Clients.

[4] Laut FIDE-Regeln gibt es noch weitere Möglichkeiten ein remis zu generieren, diese sind aber nicht Teil der Anforderungen.

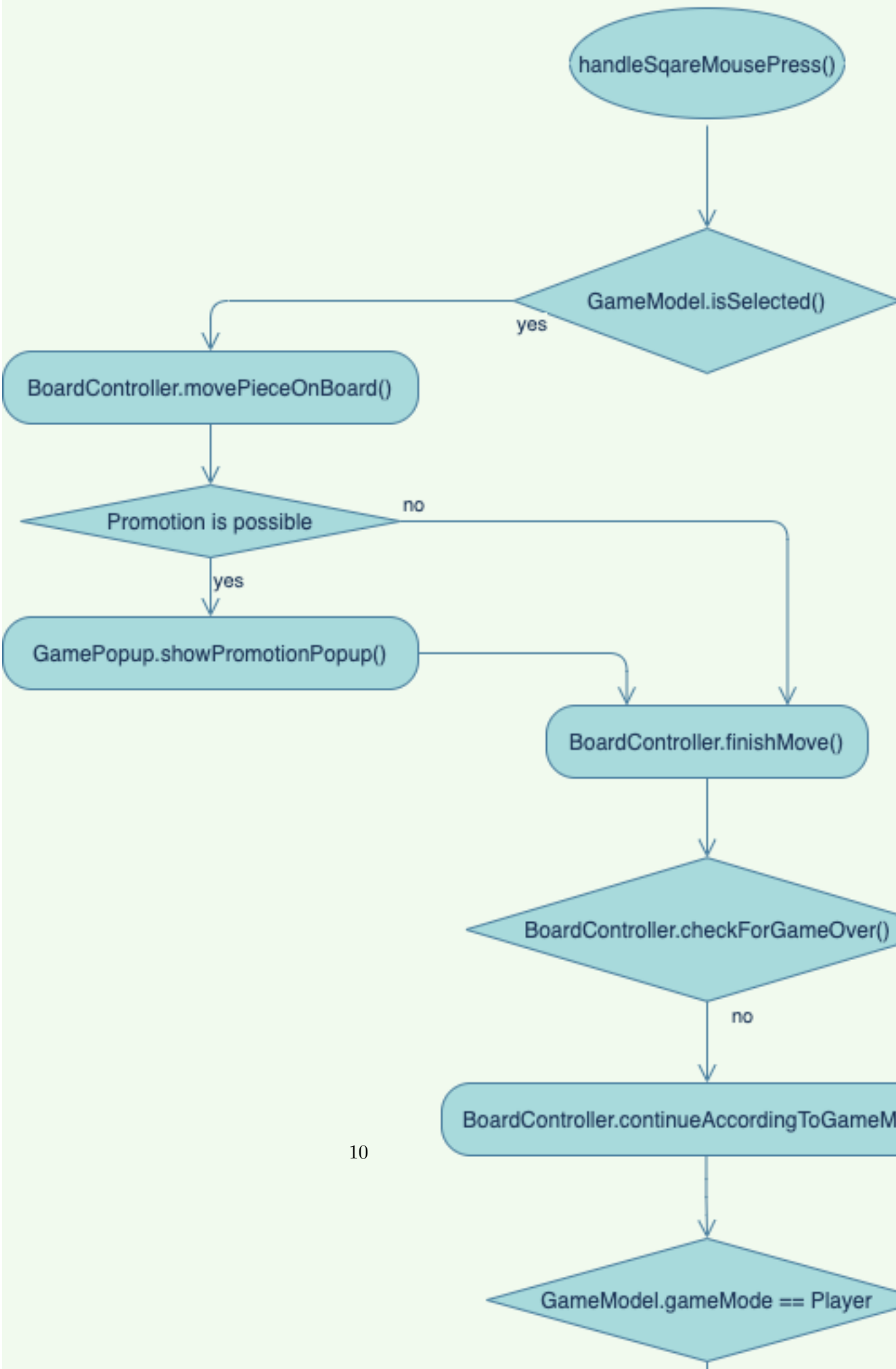
3 cli

4 gui

Dieses Aktivitäts Diagramm zeigt (in abstrahiert), wie ein Klick des Users auf dem Spielfeld verarbeitet wird. Dabei ist zu sehen, dass es nicht für jeden Spielmodus einen eigenen loop"gibt. Stattdessen teilen sich die Spielmodi weitgehend funktionen. Dadurch werden



Codeduplikate vermieden.


handleSqareMousePress()
GameModel.isSelected()
yes
BoardController.movePieceOnBoard()
Promotion is possible
no
yes
GamePopup.showPromotionPopup()
BoardController.finishMove()
BoardController.checkForGameOver()
no
BoardController.continueAccordingToGameM
GameModel.gameMode == Player

5 engine