Final Chess Specification

CST 338 - Aaron Gordon

Overview

This is a Java Swing implementation of chess for two human players (i.e. no AI will be implemented).

The application will be created using an MVC pattern, where a GameController class contains references to both a GameModel class and a GameView interface.

The GameModel class stores a data representation of the current state of a chess game. Additional classes for representing the board and pieces will be required.

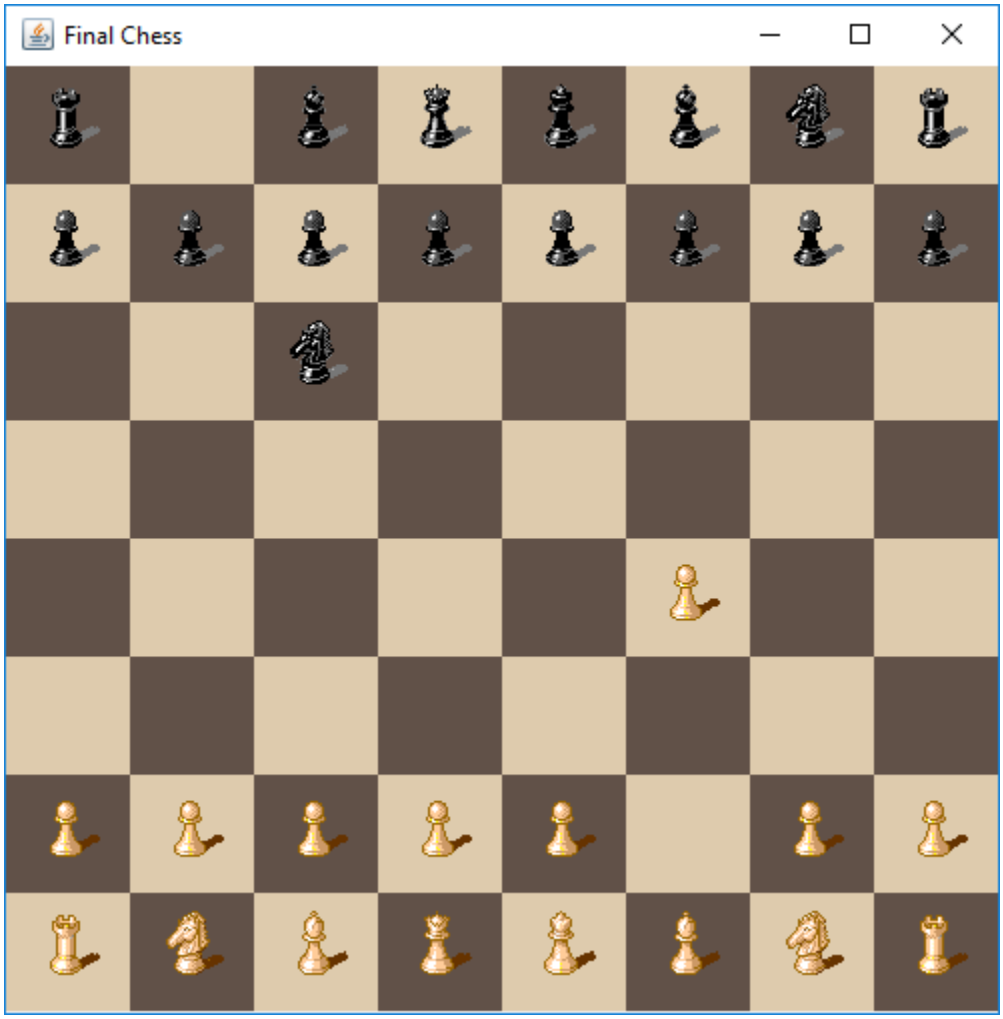
The GameView interface exposes an update method that accepts a GameModel as a single parameter, and which can be implemented to output the GameModel. This project will include only a single implementation of GameView (using Swing to render a UI) but using an interface leaves room for future, alternative views such as a console display or even an outbound network connection.

The GameController class acts as the "glue" between the GameModel and a GameView implementation. The GameController is responsible for reacting to user input and enforcing the rules of Chess.

User input is handled via an implementation of the GameViewListener which exposes various callbacks for implementation and is attached to a GameView implementation within the GameController.

Rule enforcement is handled via subclasses of the abstract PieceBehavior class. PieceBehavior exposes a public method for identifying if a given move is legal, which in turn relies upon a protected, abstract method for calculating a movement path. Individual subclasses of PieceBehavior implement this calculatePath method with the appropriate logic for each type of piece.

User Interface



Class List

BishopBehavior

Subclasses PieceBehavior to implement movement rules for the bishop. The bishop may move any distance in a diagonal line, but may not pass through other pieces.

Board

Represents a chess board of arbitrary dimensions. Pieces may be added to any open position on the board and removed from the board. Pieces already on the board may be added to any other open position on the board. Note that the board does not enforce any movement logic beyond forbidding the stacking of pieces. Pieces on the board may be looked up via board positions.

ChessVector

Stores two integer components (x, y) and exposes helper methods for working with them. Intended for use in enumerating and peforming calculations with board positions.

DebugLog

Helper class for recording debug information. Currently only writes to standard output.

GameController

Implements high-level game rules via a GameViewListener implementation. This class functions as the "glue" between GameView and GameModel. Note that only high-level rules are implemented in this class, such as turn order and win conditions. Rules relating to individual pieces are handled by their respective PieceBehavior subclasses.

GameModel

Stores data related to a game of chess, including the board and a list of captured pieces used for calculating scores in some chess variants. Exposes methods for accessing and mutating said data.

GameView

Interface to be implemented by classes intended for use as a GameView with the GameController class. This interface allows the same controller and model to be used with any view, such as a Swing UI view, a console display or even an outbound network connection.

GameViewListener

Interface to be implemented by classes that need to respond to a GameView.

KingBehavior

Subclasses PieceBehavior to implement movement rules for the king. The king may move to any adjacent square.

KnightBehavior

Subclasses PieceBehavior to implement movement rules for the knight. The knight moves in an "L" shape, which may also be thought of as moving any three squares away provided the resulting motion is not a straight line.

Main

The main entry point for the program. Initialized a GameModel, GameView and GameController.

PawnBehavior

Subclasses PieceBehavior to implement movement rules for the pawn. The pawn may only move forward one square, but may move forward two squares on its first movement. The pawn is only able to capture pieces in an adjacent, diagonal square, referred to as En Passant.

Piece

Represents a chess piece, including both its type (pawn, rook, etc.) and team (white or black).

PieceBehavior

Abstract class that determines if a given movement is valid for a given board. Classes handling movement rules for individual piece types should subclass this class and implements its calculatePath method.

PieceBehaviorFactory

Exposes static helper methods for identifying which subclass of PieceBehavior to use for a given Piece. Note that PieceBehaviors are lazy-loaded and cached, meaning no manual initialization or caching is necessary.

PieceTeam

Enumerates the teams a piece may belong to.

PieceType

Enumerates the types of pieces in a game.

QueenBehavior

Subclasses PieceBehavior to implement movement rules for the queen. The queen has the combined movement capabilities of both the bishop and the rook.

RookBehavior

Subclasses PieceBehavior to implement movement rules for the rook. The rook may move any distance in a straight line, but may not pass through other pieces.

SwingGameView

Implementation of the GameView interface that uses Java swing to render the board and pieces to the screen.

SwingPieceIconFactory

Exposes static helper methods for loading Swing icons from disk. Note that icons are lazy-loaded and cached, meaning no manual initialization or caching is necessary.