# Validation

It is important to be able to determine if a move is legal or not. For this purpose, a validation class was created. The validation class has several validation functions that take in a position and a move and check to see if it is valid. It was separated from the move class to create a proper separation of concerns where the move can check if it is valid, but the logic is abstracted to a separate class.

NOTE: It probably makes sense to add the position and move as class member variables and create an instance of the Validation class. For now, the static functions are fine to have.

## Functions

[public boolean isDiagonal (Position position, Move move)](#_isDiagonal)

[public boolean isStraight (Position position, Move move)](#_isStraight)

[public boolean isKnight (Position position, Move move)](#_isKnight)

[public boolean isKing (Position position, Move move)](#_isKing)

[public boolean isPawn (Position position, Move move)](#_isPawn)

[private boolean isValidDestination (Position position, Move move)](#_isValidDestination)

### isDiagonal

NOTE: This function assumes that the function was called in good faith, that is to say that only valid pieces (bishops and queens) are called for the isDiagonal function.

NOTE: This function only checks if a piece can move from square A to square B for a given board position. It assumes the legality of the given position and does not deal with anything to do with checks (if for example moving a piece put the king in check). It also assumes that all moves are within the realm of possibility (that is that they do not exceed the bounds of the board, etc.).

This function checks to see if a move is legal diagonally. There are a few key points to that being the case.

1. The path of travel must have a slope of 1
2. The path of travel (with the exception of the destination square) must be empty
3. The destination square must be valid. That is, it cannot contain a piece of the same colour as the one was moved

If those three conditions are satisfied, then the move is diagonally valid, and the function returns true. If any of those conditions are false, the move is not diagonally valid, and the function returns false.

### isStraight

NOTE: This function assumes that the function was called in good faith, that is to say that only valid pieces (rooks and queens) are called for the isStraight function.

NOTE: This function only checks if a piece can move from square A to square B for a given board position. It assumes the legality of the given position and does not deal with anything to do with checks (if for example moving a piece put the king in check). It also assumes that all moves are within the realm of possibility (that is that they do not exceed the bounds of the board, etc.).

This function check to see if a move is a valid straight move.

1. One of the x or y coordinates of the initial and destination squares must be equal.
2. The path between the two squares must be empty
3. The destination square must be valid. That is, it cannot contain a piece of the same colour as the one that was moved

If all three conditions are true, the function returns true.

### isKnight

### isKing

The king can move one square in any given direction. It is trivial to check every square that the king can move to (8 possible squares) to see if it is valid (see the isValidDestination function). Things become more complex when castling is factors in, however the algorithm runs as follows.

If the absolute value of Δx is equal to two, castling was attempted. Castling needs to be split up into queenside and kingside cases as they are slightly different.

Generic Requirements

1. Neither the rook or king can have moved

Kingside

1. The two squares in the positive x direction must be empty
2. Three squares, beginning with the king and moving to the right cannot be controlled by the opponent’s pieces

Queenside

1. All squares between the king (decreasing x values from the king) and rook must be empty
2. Three squares, beginning with the king and moving to the right (decreasing x values from the king, is actually left for black) cannot be controlled by the opponent’s pieces

NOTE: The destination square can actually be omitted as it will be analyzed in the isCheck function but doesn’t hurt to include. The isCheck function assumes that it only needs to check the destination square of the king, and nothing else.

### isPawn

There are four different types of moves that can be legally played.

1. A pawn moving one square forward
   1. The destination square must be empty
2. A pawn moving two squares forward
   1. Both squares in front must be empty
   2. The pawn cannot have moved before
3. The pawn captures
   1. The pawn moves one square forward, and one left or right
   2. The destination square must contain one of the opponent’s pieces
4. Enpassant
   1. The pawn must have moved 3 squares forward
   2. The pawn must move one square forward, and one left or right
   3. The destination must be empty
   4. There must be a pawn of the opponent’s colour one square above the destination square
   5. The pawn must have just move there in the last turn by going two squares forward

### isValidDestination