Checks and Pinned Pieces (Bitboards) (/Checks+and+Pinned+Pieces+%28Bitboards%29)

Table of Contents

This is about whether the king is in check. If so, one likely uses a specialized check evasion move generator. One may also trigger search extensions - based on the king is in check - or based on the check evasion move generator only reports one valid move. Related to determining discovered check is to look for absolute pins.

Checks

Whether the king is in check may be determined on the fly by looking up attacks to the king square - or based on the last move made by the other, probably checking side. Another option is to determine check giving moves already at generation time, and to flag moves accordantly.

On the Fly

Whether a king is in check can be determined by the attacked-routine mentioned in Square Attacked By. We pass the square of the king and opposite color for the potential attackers.

If one needs the set of attackers, either empty, single or double populated - one likely better relies on a specialized attacksTo-routine for each side, e.g. as member of the standard bitboard board-definition:

```
U64 CBoard::attacksToKing(enumSquare squareOfKing, enumColor colorOfKing) {
   U64 opPawns, opKnights, opRQ, opBQ;
   opPawns     = pieceBB[nBlackPawn   - colorOfKing];
   opKnights   = pieceBB[nBlackKnight - colorOfKing];
   opRQ = opBQ = pieceBB[nBlackQueen  - colorOfKing];
   opRQ       |= pieceBB[nBlackRook   - colorOfKing];
   opBQ       |= pieceBB[nBlackBishop - colorOfKing];
   return (arrPawnAttacks[colorOfKing][squareOfKing] & opPawns)
        | (arrKnightAttacks[squareOfKing]            & opKnights)
        | (bishopAttacks (occupiedBB, squareOfKing)  & opBQ)
        | (rookAttacks   (occupiedBB, squareOfKing)  & opRQ)
        ;
}
```

By Move

Another idea is to determine the check inside the search-routine by the last move done by the opponent. It might be a direct or discovered check - or in case of double check, both. With bitboards the possible savings to determine checks by last move seems negligible - and one probably better relies on the branch-less on the fly solution, since we may reuse the rook-wise and bishop-wise attacks for other purposes - like determining absolute pinned pieces, kingsafety-evaluation, and/or whether the opponent side threatens checks or discovered checks.

Direct Check

For the direct check we may use the routine mentioned in Square Attacked By:

```
if ( isAttacked(squareOfKing, move.to, move.piece(), occupiedBB) ) -> direct check
```

Discovered Check

One solution is to determine the direction between the from-square and the king square (e.g. by 0x88 difference). If both squares share a common line, one calls an appropriate sliding ray- or line-attack getter with king square and occupancy, to look whether this set intersects the possible opponing sliders of that ray. One also has to prove, whether a possible true result was not caused by a direct check of a sliding capture.

Absolute Pins

To detect absolute pins is necessary for legal move generation and may be considered in evaluation. While there are different approaches, the most common for programs based on single sliding piece attacks rather than direction wise set-wise attack getter, relies on the xrayRookAttacks or xrayBishopAttacks routines - called with square of own king and own pieces as blockers. An in-between lookup determines the set of pinned pieces while traversing the pinning pieces.

```
pinned = 0;
pinner = xrayRookAttacks(occupiedBB, ownPieces, squareOfKing) & opRQ;
while ( pinner ) {
   int sq  = bitScanForward(pinner);
   pinned |= obstructed(sq, squareOfKing) & ownPieces;
   pinner &= pinner - 1;
}
pinner = xrayBishopAttacks(occupiedBB, ownPieces, squareOfKing) & opBQ;
while ( pinner ) {
   int sq  = bitScanForward(pinner);
   pinned |= obstructed(sq, squareOfKing) & ownPieces;
   pinner &= pinner - 1;
}
```

Opposite Ray-Directions

Another idea to determine absolute pins as well as distant check block-targets, or possible discovered check origins, is to apply disjoint direction-wise attacks, as demonstrated in the DirGolem proposal, where this technique is used for all eight ray-dirctions. The intersection of direction attacks of potential pinning sliding pieces with the opposite ray-direction attacks of the king treated as sliding super piece, gains enough information if further intersected by empty squares, own, or opponent pieces. For instance as illustrated with a black rook and white king on the same rank, with no, one and two pieces inbetween:

No obstruction, king in check. In-between intersection consists of empty squares as target set to block the distant check:

```
     . r . . . . K .
r->  . . 1 1 1 1 1 .
<-K  . 1 1 1 1 1 . .
&    . . 1 1 1 1 . .
```

One piece in-between. Intersection leaves a pinned piece if of king color, otherwise a possible discovered checker:

```
     . r . . x . K .
r->  . . 1 1 1 . . .
<-K  . . . . 1 1 . .
&    . . . . 1 . . .
```

Two (or more) pieces in-between. Intersection is null:

```
      . r . x x . K .
r->   . . 1 1 . . . .
<-K   . . . . 1 1 . .
&     . . . . . . . .
```

See also

- Check
- DirGolem
- Discovered Check
- Double Check
- Pin

Forum Posts

- Fast check detection in bitboard engine by Jean-Francois Romang, CCC, December 10, 2003
- Bitboards and inCheck by Andreas Guettinger, Winboard Forum, January 03, 2007
- Bitboard of Pinned Pieces by Grant Osborne, CCC, July 24, 2008
- Check detection and move generation using bitboards by Lasse Hansen, CCC, December 06, 2011

What links here?

Page	Date Edited
Andreas Guettinger	Aug 14, 2013
Bitboards	Nov 14, 2017
Check	Feb 1, 2018
Checks and Pinned Pieces (Bitboards)	Aug 14, 2013
DirGolem	Jun 5, 2016
Discovered Check	Feb 1, 2018
Double Check	Apr 4, 2013
Intersection Squares	Oct 4, 2014
Mate at a Glance	Sep 24, 2014
Pin	Mar 7, 2017
Square Attacked By	Jan 20, 2018
Tactics	Jan 12, 2018

Up one Level