Chess Table-base Programming Project

# Still to do

* For chess algorithms:
  + Generate moves, pawn move, pinned checks (using checkRays)
  + Generate moves, already in check, everything apart from king moves
  + Undo moves, king moves *properly*
  + Undo moves, already in check (all of it)
* For Evaluation algorithm (increasingly longer term):
  + Use tableIndex numbers rather than boards and FEN’s for everything (including text file storage and in ‘Tablebase’ array of positions. This will require changing the position object, creating a tableIndexToBoard function, among other things
  + Change the iterative part of the algorithm so that it does not store a list of positions (or indexes) but updates all indexes immediately. This approach can be adapted for ‘potential mate in x’ (e.g check all escape options immediately after ‘undo moves’ for every position)
  + Update tableIndex to work for a variable number of pieces (atm it is fixed at 3)
  + Update tableBaseIndex to work with more pieces
  + Update GenerateAllPositions for same
  + Update the part of the algorithm that finds all dependencies so that it actually works for all cases.
  + Treat rotationally similar and symmetrical positions as the same positions, using a ‘genEquivBoards’ function that takes the board / index, flips and rotaes it in all unique ways, and returns an array/ list of all indexes. This reduces the storage requirements roughly 8 times. This change means that, when a new positions has an evaluation of DTM 5, all 8 similar positions must be checked, if 1 has an existing evaluation use that for all 8, if not assign the evaluation to the first in the list and move on. Since so many positions are ignored with this method, a null flag will be used (as an extra boolean attribute of the position object) so that empty positions can be passed over quickly. This will likely be the final required change before attempting to create a 4 piece tablebase.