Change Log

**Standards:**

* Use numpy arrays instead of lists for the board object ALWAYS
* Pass piece references instead of positions as this is the all-in-one method of passing data
* Positions and indexes are ALWAYS to be represented as tuples. Tuples behave like lists cannot be dynamically resized; hence, groups of tuples will be stored within lists.
* The board is always oriented from white’s perspective. This gives some symmetry in calculation and makes conceptualisation consistent and easier.
* Objects DO NOT mutually contain references to each other. ONE object may contain references to many others, but those objects MUST NOT contain references to the first object
* Colours must ALWAYS be passed as the characters ‘w’ and ‘b’. A separate mathematical value can be calculated if necessary to represent the colours (-1 for black and 1 for white, for instance)
* One-line comments are denoted by hash and a space ‘# ’. Commented lines of code are denoted by a hash with no space ‘#’
* Band-aid solutions are to be implemented in the form: title, fix, drawback, end. The title is the type of fix, a band-aid, the fix is what has been implemented to fix what problem, the drawback is the downside to implementing this fix – the reason it is not a permanent fix, the end is a comment at the end of the band-aid code block clearly signifying its end. Band-aid comments should be enclosed with ‘###’ braces.

**To-do’s:**

* Install new en pasan code for pawn moves. This will finish the game. Making it fully playable
* Player move is unable to obtain all relevant move data except when the band-aid solution (denoted in function) is applied. This band-aid results in computational inefficiencies. Restructure player\_move class to ensure this runs more effectively.
* Update castling as it does not move the rook
* It is assumed game over checks have been done before reaching the AI class, hence, design and implement these checks to ensure the class does not run out of useable moves
* Redo AI function so that it ONLY returns the move it has chosen. Allow the main script to edit and update game variables instead. Also update the class such that it runs on the new board/piece dataset
* Update all expressions of positions, they MUST be tuples as this is easier to index positions with. Tuples can be fed directly into a numpy array, but feeding it a list containing the same data will produce an unexpected list comprehension error. Tuples can be split and assembled in a similar way to lists so this should not be a major issue.
* Ensure the standard is pass pieces instead of positions as this is an all-in-one method of passing data – and since objects are passed as pointers it is not more computationally expensive. Also ensure that all move generation functions are identical in regards to inputs. Make it the standard to work with numpy arrays in place of lists unless absolutely necessary
* Redo piece and board datatypes. Pieces shall consist of small objects containing piece data: the type, colour, code, position, has\_moved variable, and value. The board will become a numpy array containing only Piece objects – empties will be denoted as a Piece object containing ‘-‘ characters as each of the class attributes.
* Redo piece functions to use tuples more concisely
* Multithread the script such that the window can be exited at any time, even mid-AI-move

**Log:**

**22/3**

Added additional comments to some functions

Player\_move function has been band-aided to ensure special moves are passed to subsequent function calls *in full*.

**6/3**

Discovered an issue in get\_moves where empty lists are resulting in list index issues when attempting to ‘de-list’ a piece-specific move set. Apply a better formatting algorithm to ensure this error is not thrown, as it is uncatchable by nature of the single line move generation piece of code.

Issue resolved by including cumbersome line-by-line for loops instead of a list comprehension

Added threatened\_only bool to get\_moves function

Ensure check function is acting properly for AI pieces. AI doesn’t seem to respond properly to check

Fixed this by adjusting an error which meant the king moves weren’t being loaded properly in the get\_moves function. This error simply ignored the king index when the PIECE\_CODES dictionary was loaded

Streamlined print\_board into a single line

Introduced a staggering number of list comprehensions for iterating move lists, validating moves in bulk, and iterating board squares. These run faster than nested for loops, look far nicer, are more easily read, and save on many lines of code.

Finish debugging is\_in\_check function to ensure optimal performance. Object references need to be removed from the piece\_dict in order to prevent them from being called ever again. Should these objects be emptied or sterilised (their attributes set to NULL and their move function returns an empty list)? Or can they be destroyed and removed from the list easily?

A sterilise function was added to Piece objects such that they can reset themselves on the fly. This reduces passed references to varying piece dictionaries which need to be updated as the pieces are removed, and instead, these pieces simply sterilise themselves such that their attributes become unusable (and in such a way that they are ignored instead of becoming error-prone)

**5/3**

Test if numpy array can hold objects in a similar way to a list.

They can if dtype is set to the object type

Test sizes of Piece object including/excluding relevant functions

Class objects can contain as many methods as desired and this **WILL NOT** increase the byte size of the object created, as is logical

Test if Piece object can contain pointers to functions passed from the piece declaration

They can, but this will require some heavy tweaking and may not be practical as functions MUST be written as if they were present within the Piece object from declaration, meaning syntax will undoubtedly be a problem. Piece functions could be unique and contain their own code – as this has been proven to not increase the size of the object at runtime – but this is cumbersome and less reusable as all move calculation code should be universally accessible.

Removed Empty() class as an empty object must be denoted by a Piece() object containing None variables. Lib.EMPTY now contains a blank Piece object

King value updated from 0 to 200

Check if lib.piece\_dict is updating when piece objects are updated elsewhere

Validated

Temporarily made the pseudo\_move\_piece function redundant as it seems to have no use

Added a ‘threatened’ Boolean to piece move functions. This separates threatened squares from available moves. If threatened is true, the function returns only threatened squares

Redo pawn move generation such that it is EASIER TO READ

Streamlined it by preloading indices similar to knight function

Reorganised get\_moves and is\_in\_check functions to use the new numpy board and piece\_dict sata structure

Updated Piece object to contain a sterilise function which essentially destroys the object, rendering it unusable

Redo is\_in\_check function. Function works but isn’t very efficient. Add piece tracking dictionaries for easy king finding etc. Dictionaries should include the piece code and its location. This will hardly matter for other pieces other than use for looping over to quickly collect available moves

Dictionaries added and are extremely effective and helpful. Used only for piece tracking and quick, easy move collection

**4/3**

Debug move collection functions. Bishop function seems to only allow for one move in the top left or bottom right corner for white and black respectively

Bishop function was return valid\_moves within the for loop meaning only one move could be collected before returning

Test dictionaries as function-holders and whether they can be key-matched to pieces.

Validated

If the above is successful, test dictionaries as object-holders and store pieces within them

Both work