# Objectives

## MVP – Basic Terminal Game – No piece rules or collisions

1. Store the location of all chess pieces on the board and be able to reset the board to the correct positions for each piece
2. Be able to print the board to the terminal with the correct row and column headings
3. Have the game alternate between players who can input 2 board positions in the terminal in order to move the piece in position 1 to position 2
4. Check that the position chosen by a player has a piece of their colour
5. Check that the destination position either has an opponent's piece or is empty
6. Move piece and switch turns

## Stage 2 – Actual Rules

1. Check that the position to move to is valid for that piece and starting position according to chess rules, e.g. a rook can only move vertically OR horizontally
2. Check that the position means that the piece isn't going through any other pieces, unless it is a knight
3. Be able to detect check by determining if the king can be taken next move
4. Be able to detect checkmate by determining that they are no possible moves to get out of check, and end the game and say who won
5. Check a move doesn't put their king in check, and gets them out of check if the king is currently in check
6. If all the checks are passed, move the piece and switch turns, and take an opponent's piece if there is one there
7. Determine if there is a stalemate using basic checks, e.g. if there are only 2 kings left
8. Be able to either play 2 players, or 1 player against an "AI" that just plays random, legal moves

## Stage 3 – More Rules and GUI

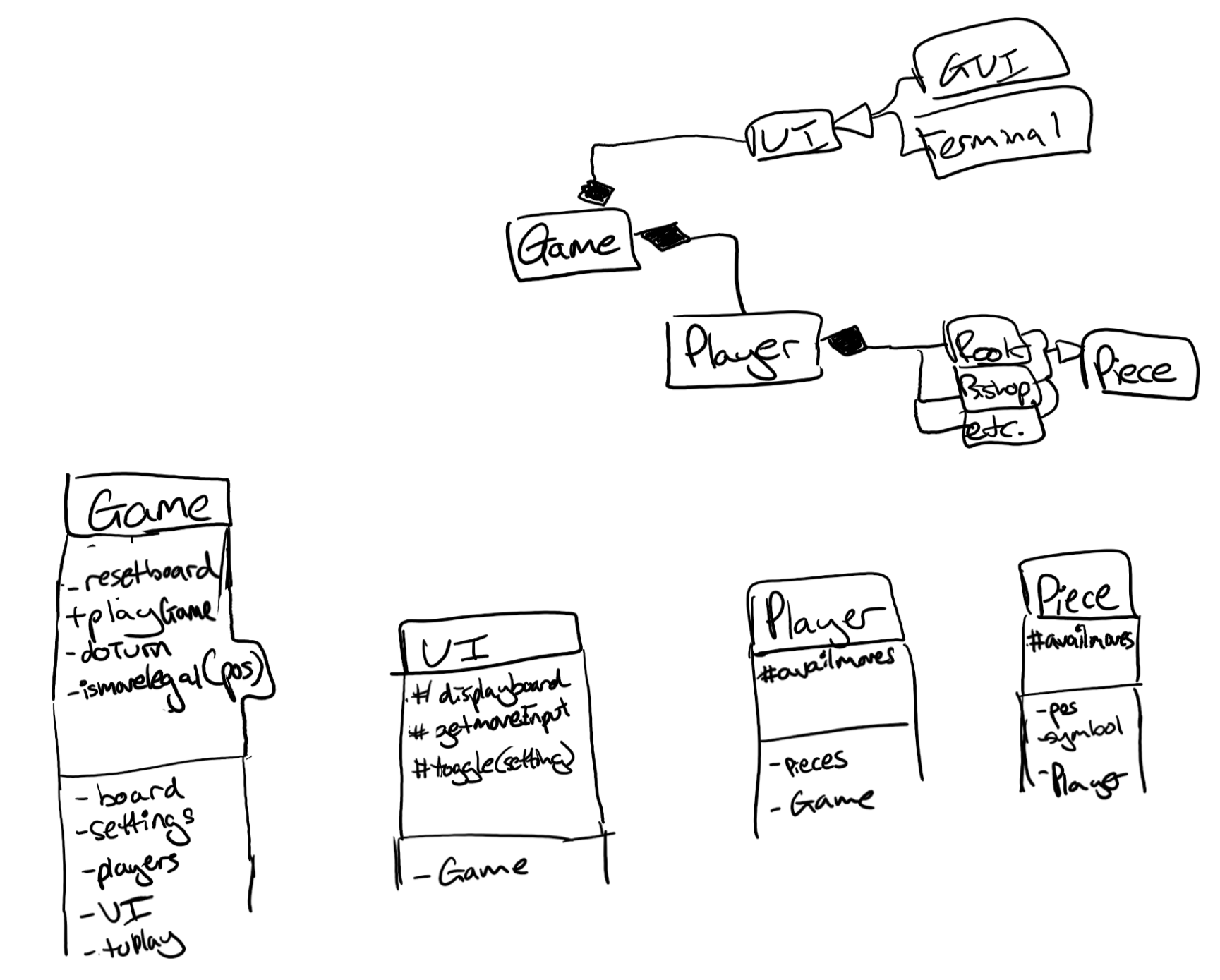
1. Implement the rest of the rules, including castling, en passent, the rest of the stalemate clauses, pawn promotions etc.
2. Provide a GUI to display the board state
3. Develop the GUI so that the player can drag and drop pieces to play a move, instead of entering them in a terminal
4. Add game options to the GUI, like AI difficulty (for when it's implemented), restart game, who goes first

## Stage 4 – AI and extra features

1. Implement an actual AI with varying levels of difficulty than can be selected
2. Be able to load and save games using the standard .pgn format
3. When loading a game, be able to step through the saved moves (replaying the game)
4. Be able to show the player a suggested move if they ask
5. Have a toggle to show the valid moves for a piece whenever it is selected (in the process of being dragged and dropped)
6. Include different rulesets, such as Fischer random chess
7. Implement optional accounts to save stats, played / in progress games
8. Be able to play across LAN or the internet using the accounts to uniquely identify every player

# Documented Design

## OO Model



## File Model

Pieces.py – contains the base piece class and all the descendants

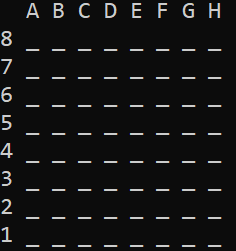
Game.py – contains the game and player classes

UI.py – contains the UI abstract base class and its 2 descendants

Chess.py – the program the user runs to play a game

## UI Design

### Terminal



Underscores represent empty cells

Pieces are represented using the [Unicode chess piece characters](https://en.wikipedia.org/wiki/Chess_symbols_in_Unicode), e.g. ♔, though reversed colours due to terminal being white text on black background.

### GUI

A

## Data Structures

Game.\_\_board = [[Game.EMPTY for \_ in range(Game.DIM)] for \_ in range(Game.DIM)] # A 2-d, square (nxn) array, initialised with empty squares

As said above, it will be OO based, with a main game class, which contains 2 player instances and a UI, which themselves contain attributes and objects detailed above.

A move is a tuple or list of 2 strings, the old position then the new position, letter then number, e.g. [“A3”, “B5”]

The available moves, for the AI and checking if a move is legal, is a list of moves, i.e. a list of 2-tuples / 2-d list of (2xn)

Symbols dictionary to map codes to Unicode piece symbols, e.g. Kw represents the white player’s king, and can be used in code, instead of the Unicode symbol

## Algorithms

A