Chess Engine Design Doc

Desired Features

Engine Startup

Users will be able to start the board as any colour in any position based on FEN notation. There will also be functionality to import games to step through and analyse with the engine's predictions.

Ease of Use Functionality

Users will be able to generate all valid moves in their position to help them play the game.

Engine AI

The engine will be able to perform static evaluation on any position of the board, and conduct a search to a certain depth, yielding the best move. This will contain the main bulk of the program and be improved upon using various techniques from the wiki.
It will also store databases containing opening books and endgame tablebases to help with early game and end game.
I will implement Iterative Deepening so that it can play in a timed setting.

Engine/Performance Testing

The engine will be able to play hundreds of games simultaneously in order to determine the elo rating of its current version. This will be helpful when testing different search & evaluation techniques.
There will be an implementation of a machine learning algorithm to determine the optimal parameters for values in the static evaluation and searching algorithm.

Lichess Compatibility

It will be able to input and output moves based on the notation of lichess.org, and so will be uploaded there to be tested against real people.

Software Requirements

M1 - Milestone 1	M2 - Milestone 2	M3 - Milestone 3	M4 - Milestone 4	FR - Functional Requirement	NFR - Non Functional Requirement

In the acceptance tests and project plan, refer to the M*.* rather than FR*.* and NFR*.*

Milestone 1 - Board Representation & Move generation

M1.1: Board Representation

- FR1: The engine shall have an internal representation of the board using a bitboard.
- FR2: Each piece (rook vs bishop, black vs white) shall have a separate ASCII value to differentiate them.

M1.2: Move Generation

- FR1: The engine shall be able to generate all possible moves for a given position.

M1.3: Gameplay

- FR1: The user shall be able to start the game as any colour in any starting position.
- FR2: The user shall be able to play any valid move at a given time.
- FR3: The interface shall display the current position of the board after every move.
- FR4: The engine shall determine if the game has ended via 50 move repetition, stalemate, or checkmate.

Milestone 2 - Basic AI & Performance Testing

M2.1: Searching

- FR1: The engine shall be able to make and unmake any valid move.
- FR2: Negamax shall be implemented.
- FR3: Alpha-beta pruning shall be implemented
- FR4: Iterative deepening shall be implemented.

- NFR5: The engine should be able to reach a depth of 10 within 5 seconds.

M2.2: Evaluation

- FR1: There shall be a handcrafted static evaluation system for the early-game, mid-game, and end-game.

M2.3: Performance Testing

- FR1: The engine shall be able to understand FEN notation.
- FR2: The engine shall be able to conduct a sequential probability ratio test to determine its elo

- NFR3: The engine should have an elo above 1500.

Milestone 3 - Advanced AI & Lichess Compatibility

M3.1: Advanced AI

- FR1: Concurrency shall be utilised for the searching algorithm.
- FR2: The engine shall store a database of opening books for the early-game.
- FR3: The engine shall store a database of endgame tablebases for the end-game.
- FR4: Basic move ordering shall be implemented.
- FR5: The results of prior searches shall be stored within a transposition table.
- FR6: Quiescent search shall be implemented.

- NFR7: The engine should have a x5 speedup using Gustafon-Barsis's Law.

- NFR8: The engine should have an elo above 2200

M3.2: Machine learning

- FR1: Hyperparameter tuning will be utilised for the evaluation.

M3.3: Lichess compatibility

- FR1: The engine shall be made compatible with and uploaded as an AI bot to lichess.com

Milestone 4 - Evaluation Improvements

- To be determined.
- Likely based on: https://www.chessprogramming.org/Search_Progression

Acceptance Tests

Includes all requirements.

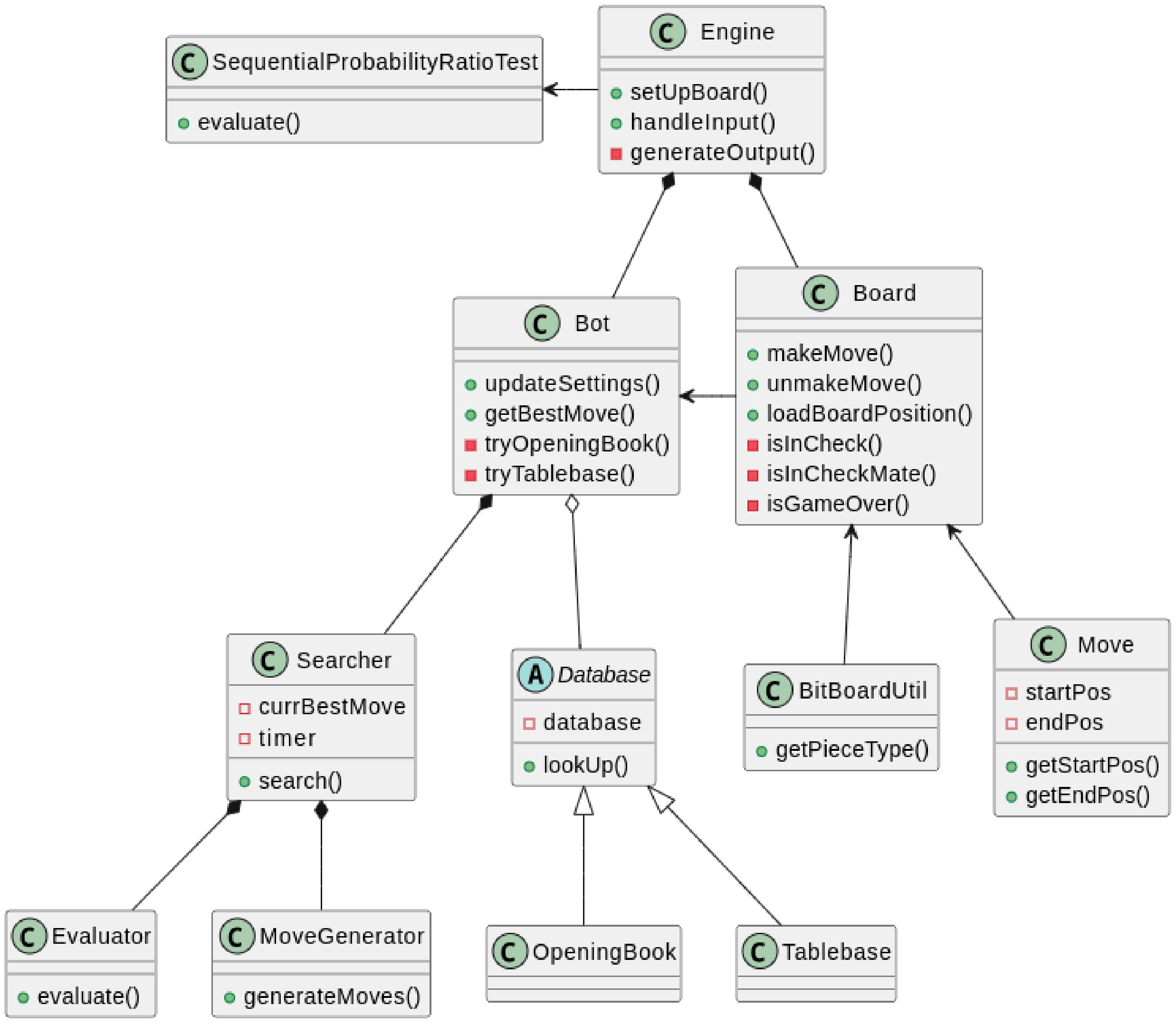
ID	Description	Input	Output
Milestone 1 - Board Representation & Move Generation			
M1.1.1	Board representation	Open and start the game.	The board should be outputted with the standard starting position.
M1.1.2	Piece ASCII values	Open the game with any starting position.	The outputted board should have different symbols for each piece type.
M1.2.1	Generate possible moves	The user will type 'display moves' or similar in various given positions.	The output should concur with Perft.
M1.3.1	Starting positions	The user will enter a starting position using FEN notation.	The board should be outputted with said starting position.
M1.3.2	Validate moves	The user will play an incorrect move then a correct move.	The incorrect move should not occur, the correct move should.
M1.3.3	Board display	Play multiple moves.	After every move, the new board position should be outputted to the console.
M1.3.4	Game endings	Play a game to completion by 50 move repetition, stalemate, and checkmate.	The engine should stop the game each time outputting the outcome for each game.
Milestone 2 - Basic AI & Performance Testing			
M2.1.1	Make/unmake moves	Make and unmake moves via the CLI.	Said moves should be made and unmade on the outputted board.

M2.1.2	Negamax	Add print statements at various points within the algorithm.	The output should concur with known correct values.
M2.1.3	Alpha-beta pruning	Add print statements at various points within the algorithm.	The output should concur with known correct values.
M2.1.4	Iterative deepening	Add print statements after each iteration.	The output should concur with known correct values.
M2.1.5	Search depth	Add a print statement at the end of each search displaying the depth reached.	This should be >= 10.
M2.2.1	Static evaluation	Add print statements at various points within the algorithm.	The output should concur with known correct values.
M2.3.1	FEN notation	Input a starting position using FEN notation.	The board should be outputted with said starting position.
M2.3.2	Sequential probability ratio test	Add print statements at various points within the algorithm.	The output should concur with known correct values.
M2.3.3	Elo rating	Calculate the elo using SPRT.	This should be >= 1500.
Milestone 3 - Advanced AI & Lichess Compatibility			
M3.1.1	Concurrency	Add print statements at various points within the algorithm.	These values should be non-deterministic and concurrent.
M3.1.2	Opening books	Play the opening of a game with and without the opening book.	The opening moves should now include moves for the opening book.
M3.1.3	Tablebases	Play the ending of a game with and without the tablebases.	The end-game moves should now include moves from the tablebases.
M3.1.4	Basic move ordering	Add print statements at various points within the algorithm. Play the game with and without the Transposition table.	The output should concur with known correct values. SPRT should show an increase in elo.
M3.1.5	Transposition table	Add print statements at various points within the algorithm. Play the game with and without the Transposition table.	The output should concur with known correct values. SPRT should show an increase in elo.
M3.1.6	Quiescent search	Add print statements at various points within the algorithm. Play the game with and without the Transposition table.	The output should concur with known correct values. SPRT should show an increase in elo.
M3.1.7	Concurrency speedup	Calculate the speedup.	This should be >= 5.

M3.1.8	Elo rating	Calculate the elo using SPRT.	This should be >= 2200
M3.2.1	Hyperparameter tuning	Play the game with and without hyperparameter tuning.	SPRT should show an increase in elo.
M3.3.1	Lichess compatibility	Go to the bot on lichess and play a match.	This should work properly.
Milestone 4 - Evaluation Improvements			
TBD			

Code Design

Class diagram



Design principles

[S] - Single Responsibility Principle:

- Each class is only responsible to one actor and serves one function:
    - Bot module - Finds the best move to perform in a given position; responsible to the engine as a whole.
    - Board module - Represents a board state; responsible to the engine as a whole & the Bot.
    - Searcher module - Performs a search through possible board positions to attempt to find the best move possible; responsible to the Bot class.
    - Evaluator module - Performs static evaluation of a given position; responsible to the searcher.
    - Database module - Abstract class representing a move database such as an opening book or tablebase.
    - Move module - Stores data related to a single move.

[O] - Open-Closed Principle:

- You can improve the evaluation stage and search state and database stage of the search independently.
- You can add extra databases without affecting other databases.

[L] - Liskov-substitution Principle:

- OpeningBook and Tablebase inherit from the abstract class Database, and have the same interface to the rest of the project.

[I] - Interface Segregation Principle:

- There is no class which depends on a method that it doesn't use.

[D] Dependency Inversion Principle:

- Needs to be efficient so this is ignored for performance critical relations such as between the searcher and evaluator, but is incorporated in other non performance critical areas:
    - The Bot class depends on the abstract class Database instead of OpeningBook & Tablebase.

Module structure

The project is split up into 4 main modules: performance testing that uses heuristics to determine the elo rating of the engine; the board that represents the board and performs operations on it; The bot that attempts the find the best move possible in any position; and the main engine that handles I/O and communication between the Bot and the Board.

- Engine: The main module of the project, coordinates communication & I/O, and handles the game logic. This will be the module that communicates with the lichess interface, and is made up of the Bot and Board.
- Performance testing: Made up of the SequentialProbabilityRatioTest class that performs an evaluation of the elo rating of the engine. Will use the Engine class in order to do this.
- Board: Represents the Board as a bitboard and will provide logic for making and unmaking moves. Will need to make use of the Move class to store moves and BitBoardUtil class for managing the bitboard.

- Bot: Contains the logic used to determine the best move in any position, using multiple move databases and the Searcher class. Stores a reference to the board.
    - Database: Abstract class that acts as the interface for the OpeningBook & Tablebase.
    - Searcher: Performs a searching algorithm(s) in order to find the best move in any position. Uses the MoveGenerator to generate all moves for a given position and the Evaluator to perform a static valuation of a given position.

Structure of data

Bitboards - The board will be stored in memory as a bitboard for efficiency reasons.
Polyglot format - The opening book will be stored in memory in Polyglot format (just a .bin file).
Syzygy tablebase - The tablebase will be stored on disk as a syzygy tablebase.

Ownership

Includes all functional requirements.

Bot::Searcher	Engine	Bot::Searcher::Evaluator	Bot::Database	Board	Performance testing
M1.2.1 Generate possible moves	M1.1.2 Piece ASCII values	M2.2.1 Static evaluation	M3.1.2 Opening books	M1.1.1 Board representation	M2.3.2 Sequential probability ratio test
M2.1.2 Negamax	M1.3.1 Starting positions	M3.2.1 Hyperparameter tuning	M3.1.3 Tablebases	M2.1.1 Make/unmake moves	
M2.1.3 Alpha-beta pruning	M1.3.2 Validate moves				
M2.1.4 Iterative deepening	M1.3.3 Board display				
M3.1.1 Concurrency	M1.3.4 Game endings				
M3.1.4 Basic move ordering	M2.3.1 FEN notation				
M3.1.5 Transposition table	M3.3.1 Lichess compatibility				
M3.1.6 Quiescent search					

Risk analysis of the tasks in project plan

Includes all requirement headings

	Severity					
Likelihood		None	Low	Medium	High	Catastrophic
	None	🟩	🟩	🟩	🟨 M1.3 Gameplay	🟨
	Low	🟩	🟩 M3.3 Lichess compatibility	🟨	🟨	🟨 M1.1 Board representation
	Medium	🟩	🟨 M2.3 Performance testing	🟨	🟨 M1.2 Move generation	🟥 M2.1 Searching M2.2 Evaluation
	High	🟨	🟨	🟨	🟥 M3.1 Advanced AI M3.2 Machine learning	🟥
	Catastrophic	🟨	🟨	🟥	🟥	🟥

Project plan for implementation

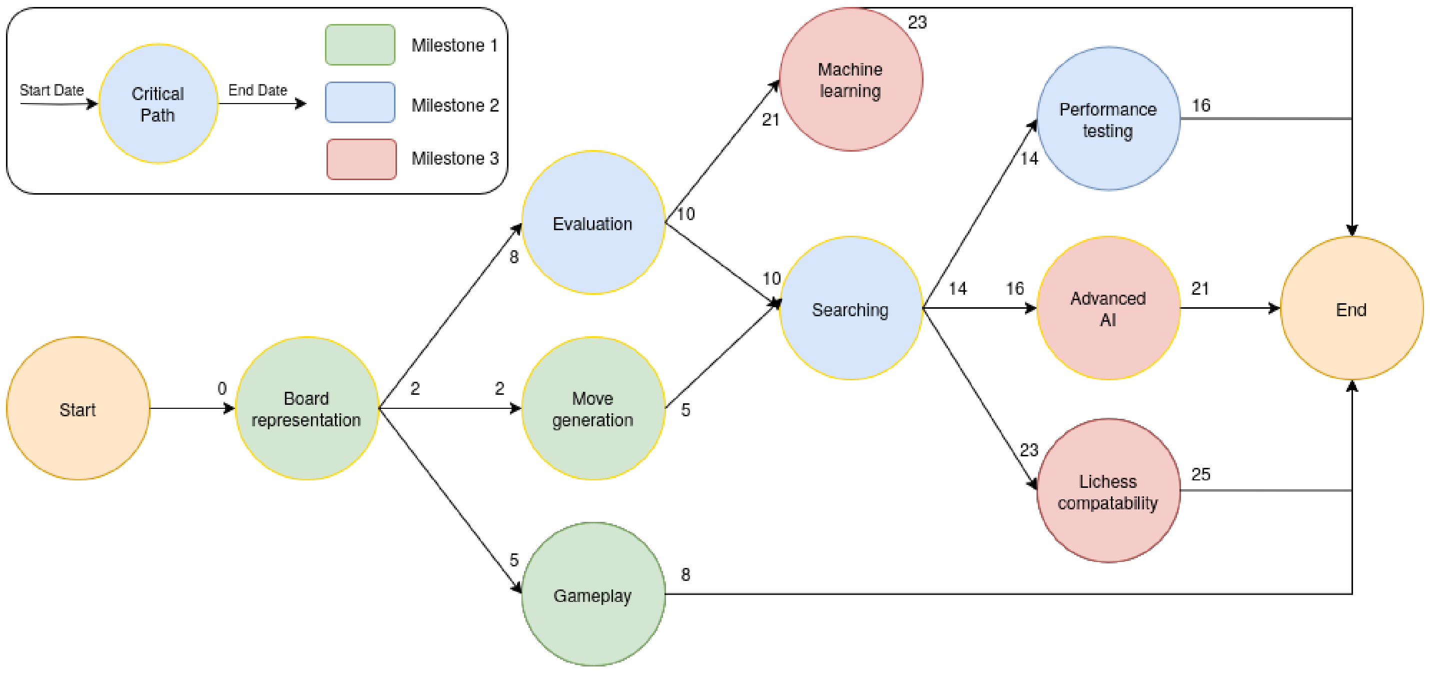
Includes all requirement headings; requirements are under 'Deliverables'.

ID	Description	Dependencies	Deliverables	Deadline	Met (□/×)
Milestone 1 - Board Representation & Move Generation					
M1.1	Board representation	None	M1.1.1 Bitboard M1.1.2 ASCII values	17/06/25 2 days	
M1.2	Move generation	M1.1	M1.2.1 Generate possible moves	20/06/25 3 days	
M1.3	Gameplay	M1.1	M1.3.1 Starting positions M1.3.2 Validate moves M1.3.3 Board display M1.3.4 Game endings	25/06/25 3 days	
Milestone 2 - Basic AI & Performance Testing					
M2.1	Searching	M1.2, M2.2	M2.1.1 Make/unmake moves M2.1.2 Negamax M2.1.3 Alpha-beta pruning M2.1.4 Iterative deepening M2.1.5 Search depth >= 10	03/07/25 4 days	

M2.2	Evaluation	M1.1	M2.2.1 Static evaluation	27/06/25 2 days	
M2.3	Performance testing	M2.1	M2.3.1 FEN notation M2.3.2 SPRT M2.3.3 Elo rating >= 1500	07/07/25 2 days	
Milestone 3 - Advanced AI & Lichess Compatibility					
M3.1	Advanced AI	M2.1	M3.1.1 Concurrency M3.1.2 Opening book M3.1.3 Tablebase M3.1.4 Basic move ordering M3.1.5 Transposition table M3.1.6 Quiescent search M3.1.7 Concurrency speedup >= 5 M3.1.8 Elo rating >= 2200	14/07/25 5 days	
M3.2	Machine learning	M2.2	M3.2.1 Hyperparameter tuning	16/07/25 2 days	
M3.3	Lichess compatibility	M2.1	M3.3.1 Lichess compatibility	18/07/25 2 days	
Milestone 4 - Evaluation Improvements					
TBD					

Activity Network

Includes all requirement headings. Days ignore weekends.



Start Date
Critical Path
End Date
Milestone 1
Milestone 2
Milestone 3
Start
0
Board representation
2
8
Evaluation
2
Move generation
5
Gameplay
10
21
Machine learning
23
10
5
Searching
14
14
Performance testing
16
16
Advanced AI
21
23
Lichess compatability
25
8
End