Stage #2

Group #57 Project Requirements Modeling: UCI Chess Engine

Alexander King Perocho
Vivian Guo
Martin Hung Nguyen
Ethan Duy Thanh Trinh
Patrick Cauilan Connors

COMPSCI 3307 - Section 001
Mr. Marios-Stavros Grigoriou
28 Feb 2025

Contents

1 Introduction **2**
1.1 Assignment Purpose & Scope 2
1.2 Overall Goal 3

2 User Stories **4**
2.1 1.1.1 FEN Parsing 4
2.2 1.1.2 Board Representation 4
2.3 1.1.3 Move Generation 5
2.4 1.1.4 Search Algorithm (Alpha-Beta Pruning) 5
2.5 1.1.5 Position Evaluation 6
2.6 1.1.6 UCI Protocol Support 6
2.7 1.2.1 Adjustable Skill Levels 7
2.8 1.2.2 Opening Book 7
2.9 1.2.3 Endgame Tables 8
2.10 1.2.4 Multithreading 8
2.11 1.3.1 Machine Learning 9
2.12 1.3.2 GUI Develop 9
2.13 1.3.3 Online Play 10

3 UML Class Diagram **10**

4 Jira Task Management **10**

1 Introduction

Dear TA,

Below is an overview of the scope for our CS3307 group project assignment, designed to lay the groundwork for our agile development process.

1.1 Assignment Purpose & Scope

Requirement Formalization:
Our team has developed user stories that capture the essential, optional, and wish-listed features for our project. Each story follows a concise format—detailing the user role, desired functionality, and the resulting benefit. Additionally, every user story includes clear acceptance criteria and an estimated story point value to gauge complexity and effort.

Initial Design:
We have created a detailed UML class diagram outlining the key classes, their attributes (with types and visibility), methods (with parameters and return types), and relationships (associations, hierarchies, etc.). This diagram reflects our initial design thinking and helps ensure that our implementation will cover all major requirements.

Project Tracking Setup:
To effectively manage and track our progress, we have set up a Jira project with a Kanban board. Each user story is entered into Jira with its corresponding story points and is assigned to a team member—ensuring accountability and a clear development workflow. We have also integrated Bitbucket for change tracking.

Key Deadlines & Requirements

Due Date:
The assignment is due on Friday, February 28, 2025, by 11:55 pm via OWL, with a late submission policy in place (20% mark reduction per day up to two days).

Group Collaboration:
This project stage is a collaborative effort, with each team member contributing equally. While we encourage discussion with other groups for idea exchange, the final deliverable is solely our group's work.

Deliverables:

- **User Stories Document (PDF):** Contains all user stories with acceptance criteria and story point estimates.
- **UML Class Diagram (PDF):** Illustrates our proposed class design, including attributes, methods, and relationships.
- **Jira Board Setup:** Our Kanban board is fully populated with our user stories, with proper assignments and status updates.
- **Bitbucket Repository:** Used for version control and change tracking throughout the project.

1.2 Overall Goal

This stage aims to ensure that we have a clear, well-documented understanding of the project requirements and initial design, setting the stage for effective development and agile progress tracking. We look forward to demonstrating how our planned approach will guide our project towards successful implementation.

Thank you for your time and consideration. We are open to any feedback or questions you might have regarding our approach.

Sincerely,

Team 57

2 User Stories

Below are the user stories detailing each feature's title, priority, user story, acceptance criteria, and estimated story points.

2.1 1.1.1 FEN Parsing

Title: FEN Parsing
Priority: High
User Story:
As a chess GUI user, I want to correctly parse FEN strings so that I can load and analyze my chess positions accurately.
Acceptance Criteria:

- The engine correctly interprets FEN strings and sets up the board state accordingly.
- Valid FEN positions (e.g., castling rights, en passant targets, move counts) are handled without error.
- An error message is returned for invalid FEN strings.
- The engine parses FEN strings into bitboards correctly.
- The engine can generate a correct FEN string from any given board position.

Story Points: 3

2.2 1.1.2 Board Representation

Title: Board Representation
Priority: High
User Story:
As a developer, I want to represent the chessboard using bitboards so that move generation and evaluation are efficient.
Acceptance Criteria:

- The chessboard and associated attack/occupancy masks are stored using 64-bit bitboard representation.
- The board supports a standard 8x8 chess layout.
- Separate bitboards are implemented for each piece type and color.
- The board representation allows for efficient legal move calculations.
- Special moves (castling, en passant, promotions) are supported.
- The board state updates correctly after moves.

Story Points: 5

2.3 1.1.3 Move Generation

Title: Move Generation
Priority: High
User Story:
As a chess engine, I want to generate all legal moves from a given position so that I can analyze and play valid chess moves.
Acceptance Criteria:

- All legal moves, including special moves (castling, en passant, promotions), are correctly generated.
- Illegal moves (such as moving into check) are filtered out.
- Precomputed attack masks are cached and utilized during move generation.
- Magic bitboards are implemented for sliding pieces (rooks, bishops, and queens).
- Moves are encoded and decoded using a 20-bit scheme.

Story Points: 8

2.4 1.1.4 Search Algorithm (Alpha-Beta Pruning)

Title: Search Algorithm (Alpha-Beta Pruning)
Priority: High
User Story:
As a chess engine, I want to search the move tree efficiently so that I can select the best possible move.
Acceptance Criteria:

- A Minimax search algorithm with Alpha-Beta pruning is implemented to reduce the number of evaluated positions.
- The search depth is configurable by the user.
- The engine selects the best move based on evaluation scores.
- Move ordering techniques (e.g., prioritizing captures and checks) are applied to search the most promising moves first.
- Heuristic scores are used to propagate and determine the optimal move.

Story Points: 8

2.5 1.1.5 Position Evaluation

Title: Position Evaluation
Priority: High
User Story:
As a chess engine, I want to evaluate a position using a heuristic function so that I can determine the relative strength of a move.
Acceptance Criteria:

- The evaluation function assigns values based on material balance, piece activity, king safety, pawn structure, and center control.
- Evaluation outputs are consistent with established chess heuristics.
- The evaluation function is designed for easy tuning and optimization.
- It provides scored moves to the search algorithm for move propagation.

Story Points: 5

2.6 1.1.6 UCI Protocol Support

Title: UCI Protocol Support
Priority: High
User Story:
As a user, I want the engine to support the UCI protocol so that I can use it with GUI applications like Arena or ChessBase.
Acceptance Criteria:

- The engine processes standard UCI commands (uci, isready, position, go, stop, quit) correctly.
- Appropriate UCI responses are sent for each command.
- The engine successfully communicates with a UCI-compatible GUI.

Story Points: 3

2.7 1.2.1 Adjustable Skill Levels

Title: Adjustable Skill Levels
Priority: Low
User Story:
As a user, I want to adjust the engine's playing strength based on an ELO rating so that I can play against opponents of different skill levels.
Acceptance Criteria:

- The engine allows skill adjustment via UCI parameters (e.g., `UCI_LimitStrength` and `UCI_Elo`).
- The skill level influences move selection by reducing search depth for lower levels, introducing controlled randomization, and adjusting evaluation precision.
- The engine mimics various ELO ratings successfully when tested against other engines.

Story Points: 2

2.8 1.2.2 Opening Book

Title: Opening Book
Priority: Low
User Story:
As a chess engine, I want to use an opening book so that I can play established openings efficiently.
Acceptance Criteria:

- The engine can load an opening book file (formats such as .bin, .pgn, .polyglot).
- Moves from the opening book are prioritized during the opening phase.
- The engine can switch between using the opening book and calculated search based on position depth.
- The opening book can be disabled via a UCI option.

Story Points: 2

2.9 1.2.3 Endgame Tables

Title: Endgame Tables
Priority: Low
User Story:
As a chess engine, I want to use precomputed endgame tablebases so that I can play perfectly in simplified positions.
Acceptance Criteria:

- The engine supports Syzygy tablebases for endgame play.
- It correctly queries tablebases to determine win/draw/loss outcomes.
- When applicable, tablebase moves are prioritized over search.
- Tablebase functionality can be enabled or disabled via a UCI option.

Story Points: 2

2.10 1.2.4 Multithreading

Title: Multithreading
Priority: Medium
User Story:
As a developer, I want the engine to utilize multiple CPU cores so that move search is faster.
Acceptance Criteria:

- The engine supports multithreaded search by distributing work across CPU cores.
- The number of threads is configurable via a UCI option (e.g., `Threads`).
- Performance benchmarks indicate a significant speedup compared to single-threaded execution.
- The implementation avoids race conditions and maintains thread safety.

Story Points: 3

2.11 1.3.1 Machine Learning

Title: Machine Learning
Priority: Low
User Story:
As a developer, I want to enhance the evaluation function using a machine learning model trained on grandmaster games so that the engine plays more human-like and accurately assesses positions.
Acceptance Criteria:

- A neural network-based evaluation function is integrated into the engine.
- The model is trained on high-quality chess games (e.g., grandmaster games, high-ELO engine games).
- The engine can switch between handcrafted evaluation and ML-based evaluation via a UCI option.
- Performance tests confirm that ML-based evaluation improves the engine's playing strength.

Story Points: 5

2.12 1.3.2 GUI Develop

Title: GUI Develop
Priority: Low (Because UCI support is already provided)
User Story:
As a user, I want a custom graphical user interface so that I can interact with the engine without relying on third-party software.
Acceptance Criteria:

- A standalone GUI is created for playing games against the engine.
- The GUI displays the chessboard, move list, evaluation bar, and time controls.
- It can send and receive moves from the engine using UCI commands.
- The interface is user-friendly and supports common chess features (e.g., takeback, move highlighting).

Story Points: 3

2.13 1.3.3 Online Play

Title: Online Play
Priority: Low
User Story:
As a developer, I want the engine to connect to online chess platforms so that it can play against human opponents in real time.
Acceptance Criteria:

- The engine can connect to online chess platforms (e.g., Lichess, Chess.com) using their respective APIs.
- It correctly handles game state updates and move submissions.
- The connection is stable and gracefully manages disconnections and reconnections.
- The implementation complies with API terms of service to ensure fair play.

Story Points: 5

3 UML Class Diagram

4 Jira Task Management