APSC 143: Introduction to Computer Programming for Engineers

Chess Engine Project Proposal

Jeevan S, Alexander S, Edward S

November 7, 2025

1. Introduction

This proposal outlines how the chess board and moves will be represented in code, using data structures that support board state tracking, move encoding, and special rules such as castling. The scope of this project is to define these fields to support board initialization and classification, move parsing, completion, and recommendation, while applying our knowledge of C.

2. Board State Representation

The state of the board is represented and stored in three fields: two arrays of 64 Booleans, with each Boolean representing one square on the board, and one array of 8-bit integers storing the type of piece on each square. The first array of Booleans stores the state of the square's occupancy.

The rationale behind the use of three fields for the board state representation is simplicity. Although the array of 8-bit integers is sufficient to denote the board state, it will be much easier to use the three arrays since it will more clearly represent the board. It will also allow functions which parse moves to quickly check square states and piece properties before doing any operations on the main array.

The first array of Booleans, **piece_state**, stores the state of the square's occupancy. If the given square is occupied, it will be followed by a 'true' (1) statement. It is then indexed by the first element representing square A1.

The second array of Booleans, **piece_side**, is used to differentiate the different colors of the piece on the board. This integer will be used to determine if a square is occupied by a white or a black piece, with 1 representing a white piece and a 0 representing a black piece. Refer to Figure 1for the use of **piece_side.**

Additionally, there will be two Boolean fields, **can_castle,** for each color containing information on castling rights for each side. This is necessary as the board state is independent of the player's castling rights, meaning that two identical board positions can have different **can_castle** values.

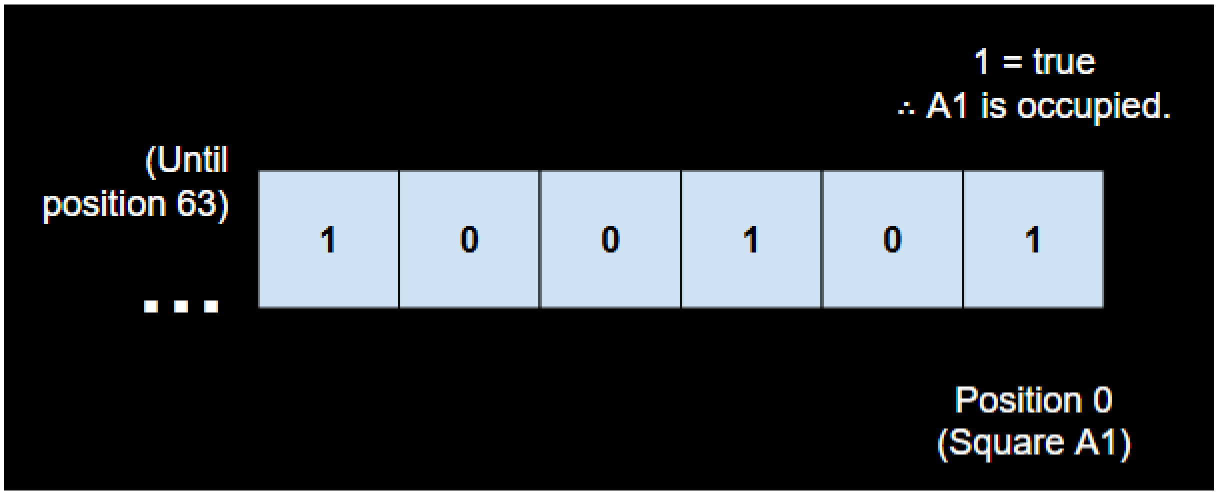

Figure 1: Binary representation of chessboard square colour. '1' indicates a white square and '0' indicates a black square.

3. Move Representation

A normal chess move can be categorized fairly easily with a field of where the piece is coming from, **from_square**, and where it is headed, **to_square**. Extra fields can be added to denote which piece is being moved, **piece_id,** and if it is capturing a piece, **is_capture**. However, there are multiple special fields that need to be added for special moves. **promotes_to_id** is a field that stores an id of a piece that a pawn will promote to. Finally, there are two more fields, **is_castle** and **is_long_castle**, these fields are both Boolean specific for the two castle scenarios, a long castle or a short castle. The full chess_move struct can be seen in Figure 2, this would serve as a great way to not only store moves but parse moves from text to our program later.

```
struct chess_move
{
    enum chess_piece piece_id; // The moving piece id

    int from_square;     // From which square id is the piece moving from
    int to_square;       // To   which square id is the piece moving to

    bool is_capture;     // True if the moving piece is capturing a piece

    int promotes_to_id;  // If a pawn is promoted, the new piece id; -1 otherwise

    bool is_castle;      // True if the move is a castle
    bool is_long_castle; // True if the move is a long castle
};
```

Figure 2: Code snippet of chess_move data structure with comment