Chess-pionage v2024.04.29

Team 7: PretendGINEERS

Mharlo Borromeo

Tuaha Khan

Jack Lu

Calvin Nguyen

Mervin Nguyen

Peter Nguyen

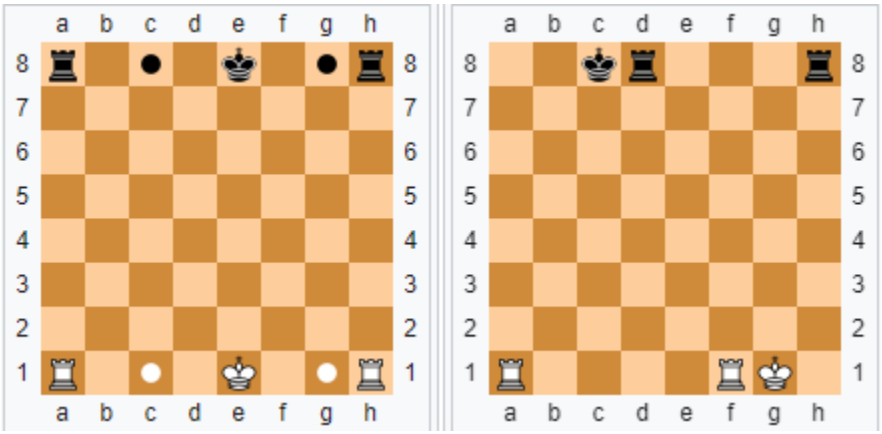
University of California, Irvine : EECS 22L S24

Table of Contents

Glossary

Castling - The castling maneuver can only be executed by a king and a rook if the king and the corresponding rook have not been moved from their starting positions before executing the move. The king will move two spaces towards the rook, either on the kingside or queenside of the chessboard, while the corresponding rook will move besides the king towards the center of the board ("hop over" the king). In order to be eligible to perform this maneuver, there must be no other pieces between the two pieces. Additionally, the king cannot castle out of check, into check, or through check.

En Passant - The en passant maneuver can only be performed by a pawn if the opposing pawn has moved two squares on the previous turn to be adjacent to the left or right of an opposing pawn, where the pawn has the right to perform "en passant" only in the immediate next turn, traveling diagonally to the space behind the opposing pawn and capturing it. If the move is not executed immediately, en passant is not eligible to be performed anymore.

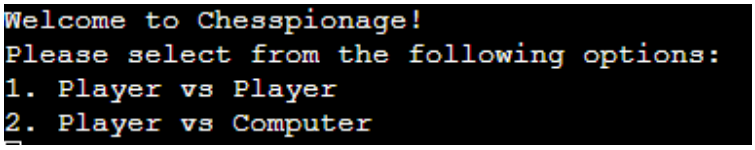


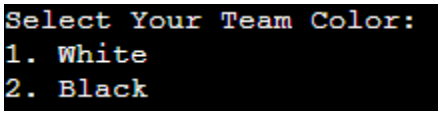
Pieces in Chess

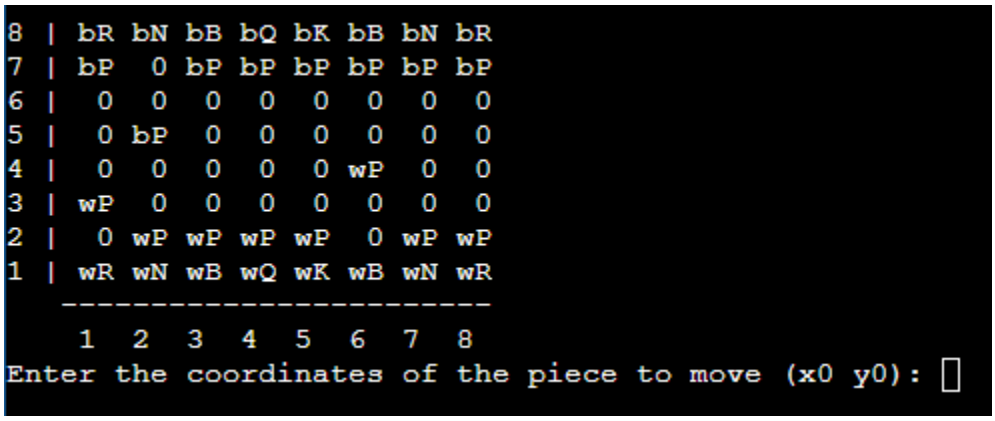
- Pawn (8) chess piece that moves one or two tiles that can capture a piece diagonally one tile left or right side.
- Rook (2) chess pieces that move forward or backward both horizontally and vertically.
- Bipshop (2) chess pieces that can move diagonally.
- Knight (2) chess pieces that can move in a L shaped pattern (3 tiles in horizontal or vertical direction, and 1 tile left or right of the main direction) either forward, backward, side to side and left or right .
- Queen (1) chess pieces that move horizontally, vertically and diagonally, like how the rook and bishop can.
- King (1) chess pieces that can move 1 tile in any direction, if no available space to move, result in a checkmate and the player loses.

1. Computer Chess

1.1 Usage Scenario







1.2 Goals

To achieve victory, the player must strategically capture the opponent's King piece by executing a checkmate maneuver. This is performed whenever a player's king chess piece is unable to perform any move without being captured.

1.3 Features

Display:

- Text user interface
- Choice of black or white chess pieces
- Chessboard and all pieces displayed
- Text prompt to ask user to input start and end coordinates of desired piece to move

Gameplay:

- Ability to choose color for player versus computer gamemode
- Ability to choose player vs player, or player versus computer
- Repeating menu to prompt user for coordinates of piece to move
- Coordinate system for moving piece

2. Installation

2.1 System Requirements

Hardware	**Minimum Specification**
CPU	Any x86_64 processor with clock speed of at least 1GHz
RAM	At least 1 GB
Disk Space	At least 500 MB
Operating System	Linux (RHEL7/RHEL8)

2.2 Setup and Configuration

- Installation (basic user, no building or compiling necessary)

- Download Chess.tar.gz compressed folder.
- Navigate to the directory where the Chess.tar.gz folder is located after downloading(for example, 'cd Downloads'), and then run command (in terminal) ‘gtar xvzf Chess.tar.gz’ in order to unpack/unzip the file.
- Navigate within the newly unzipped folder in the terminal ('cd chess'), and then navigate to the ‘bin’ folder in order to run the program executable (‘./chesspionage’).

2.3 Uninstalling

- Uninstallation (for both basic and advanced users)

- Navigate to the folder where Chess.tar.gz was uncompressed within terminal ('cd Downloads', then 'cd chess'), taking extra care that there are no other contents in the folder besides the files that were unzipped from the Chess.tar.gz file, then use 'rm -f *' to remove all files within the downloaded and unzipped folder without needing to authorize the deletion of each file at a time.

3. Chess Program Functions and Features

3.1. Detailed description of printBoard

1. Purpose:

The primary purpose of the printBoard function is to present the current configuration of the chessboard in a human-readable format to the user.

2. Function Signature:

- printBoard(struct Board *board)

3. Parameters:

- *board: This parameter represents the current state of the chessboard, typically stored as a 2-dimensional array or similar data structure. The function utilizes a pointer to this input to generate the visual representation of the board.

4. User Input:

- The printBoard function does not directly solicit input from the user. Instead, it relies on the board parameter provided as input to render the chessboard.

5. Program Output:

- Upon invocation, the printBoard function generates output in the form of a textual representation of the chessboard. This output is typically displayed in the console or terminal window.
- The output consists of a grid representing the chessboard with rows and columns labeled for easy reference. Each square of the grid corresponds to a specific position on the chessboard.
- Pieces occupying each square are displayed using standard algebraic notation, with distinct symbols or characters representing different types of chess pieces (e.g., "K" for king, "Q" for queen, "R" for rook, etc.).
- The visual representation includes markings to differentiate between squares of different colors (usually light and dark) to mimic the appearance of a real chessboard.

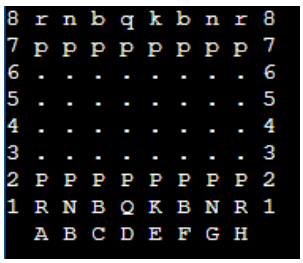


Figure 1: Output of printBoard(*board).

6. Additional Features:

The printBoard function may include additional features to enhance the user experience, such as highlighting the squares corresponding to valid moves for a selected piece.

- It may also incorporate color-coding or Unicode symbols to enhance the visual appeal of the chessboard representation.
- Some implementations may provide options for customizing the appearance of the board, such as choosing between different themes or board sizes.

7. Error Handling:

- The printBoard function typically does not involve error handling related to user input, as it receives the board configuration as a parameter. However, it may include error handling mechanisms to address issues such as invalid board configurations or unexpected data types.

3.2: Detailed description of isCheckMate

1. Purpose:

The primary purpose of the check4checkmate function is to confirm that the current board contains a checkmate of either white or black, therefore exiting the main loop, ending the game, and declaring a winner.

2. Function Signature:

- isCheckMate(struct Board *board,)

3. Parameters:

- board: This parameter represents the current state of the chessboard, typically stored as a 2-dimensional array or similar data structure. The function utilizes this input to generate the visual representation of the board.
-

4. User Input:

- The check4checkmate function does not directly solicit input from the user. Instead, it relies on the board parameter provided as input to analyze and detect for checkmate.

5. Program Output:

- When this function is called, it returns a value of 1 or 0. 1 represents a checkmate occurring, and therefore the game stops and the winner is declared. 0 represents no current checkmate, and the game continues.
- This function is called after every iteration of the loop, and if a value of 1 is returned, the main loop breaks, and the game ends.

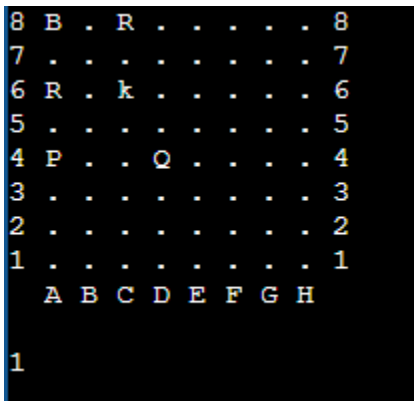


Figure 2: Program output after calling check4checkmate(*board).

6. Additional Features:

- A simple search algorithm to find the king on the board.
- Once the king is found, another loop will iterate through the opponents pieces, calling isValidMove to see if any pieces are able to capture the king. If they are, the king is in check.
- Finally, the function will use isValidMove on the king to verify if there are any more moves that the player can use. If not, a checkmate has occurred.

7. Error Handling:

- The check4checkmate function, when calling isValidMove for each enemy piece, will also check if the king is able to capture that given piece at the current turn. If so, the king is not in danger, and no check has occurred.

3.3: Detailed description of isValidMove

1. Purpose:

The primary purpose of the isValidMove function is to confirm that the piece that is being moved in the current turn is within the bounds of the specific pieces moveset according to the chess rule book.

2. Function Signature:

- isValidMove(struct Board *board, Move *move, pieceColor color);

3. Parameters:

- board: This parameter represents the current state of the chessboard, typically stored as a 2-dimensional array or similar data structure. The function utilizes this input to generate the visual representation of the board.
- move: This parameter represents the starting coordinates and ending coordinates of the piece after the user has entered in their desired move that they want to make.
- pieceColor: This parameter represents the color of the piece being moved around.

4. User Input:

- The isValidMove contains the values input from the user from the movePiece function, which are the x and y coordinates of the piece they want to move, as well as the coordinates of the square on the board in which they would like to move it (destination coordinates).
- The function also receives the value Piece, which contains the type and color of piece that is being moved

5. Program Output:

- When this function is called, it returns a value of 1 or 0. 1 represents a valid move, and the movePiece function will be called. 0 represents an invalid move, where an error message will be printed in the output console and the user will be prompted to move a different piece.
- This function is called inside of the movePiece function to validate the move. The function will be contained in a loop that will prompt the user with error messages until a valid move is input.

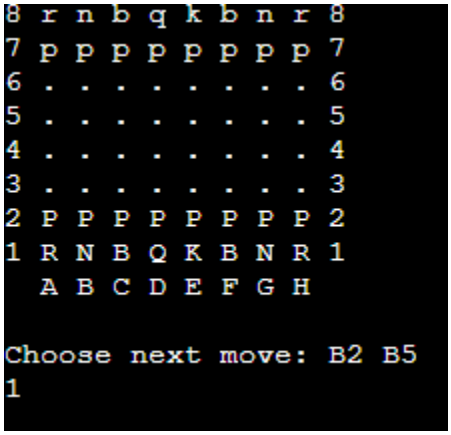


Figure 3: Output after calling isValidMove(Piece, *board).

6. Additional Features:

- The function will contain if statements for each enumerated type to check if the move input by the user is within the piece's moveset.
- For each type of piece, the destination coordinates will be checked to see if the piece can legally move in that fashion according to the rules of chess.
- The destination coordinates will also be checked to see if there is a piece with the current player's color currently present at that square. If so, the move is invalid.

7. Error Handling:

- The function will also check the destination coordinates are within the bounds of the chessboard for the row and column.

3.4: Detailed description of initializeBoard

1. Purpose:

The primary purpose of the initializeBoard function is to create a 2D array of ranks (rows) and files (columns) to represent an 8x8 chess board.

2. Function Signature:

- initializeBoard(struct Board *board)

3. Parameters:

- The function uses a pointer to an instance of a Board struct, so that it knows to create a board and fill it up with pieces.

4. User Input:

- For the initializeBoard function, the user input isn't considered. This is because the function is used within the main function to essentially create the board, which needs to be used in any game of chess in order for the pieces to work.

5. Program Output:

- The program will not output any board through this function, however, it will display a message that confirms whether or not the function call was successful.

```
char chessboard[8][8] = {
    {'r', 'n', 'b', 'q', 'k', 'b', 'n', 'r'},
    {'p', 'p', 'p', 'p', 'p', 'p', 'p', 'p'},
    {'-', '-', '-', '-', '-', '-', '-', '-'},
    {'-', '-', '-', '-', '-', '-', '-', '-'},
    {'-', '-', '-', '-', '-', '-', '-', '-'},
    {'-', '-', '-', '-', '-', '-', '-', '-'},
    {'P', 'P', 'P', 'P', 'P', 'P', 'P', 'P'},
    {'R', 'N', 'B', 'Q', 'K', 'B', 'N', 'R'}
};
```

Figure 4: What initializeBoard will create.

6. Additional Features:

- N/A

7. Error Handling:

- The function will check if enough memory was allocated for the creation of the chessboard.

3.5: Detailed description of pieceMove

1. Purpose:
    - update the game state by when a piece is moving to a new position on board.
2. Function Signature:
    - pieceMove(struct Board *board, Move *move, pieceColor color);
3. Parameters:
    - *board refers to the board that the chess game is currently being played on, referenced through a pointer.
    - *move refers to the move that the player has just made through inputting start and end coordinates of the respective piece.
    - color refers to the color of the chess piece that is being moved.
4. User Input:
    - N/A
5. Program Output:

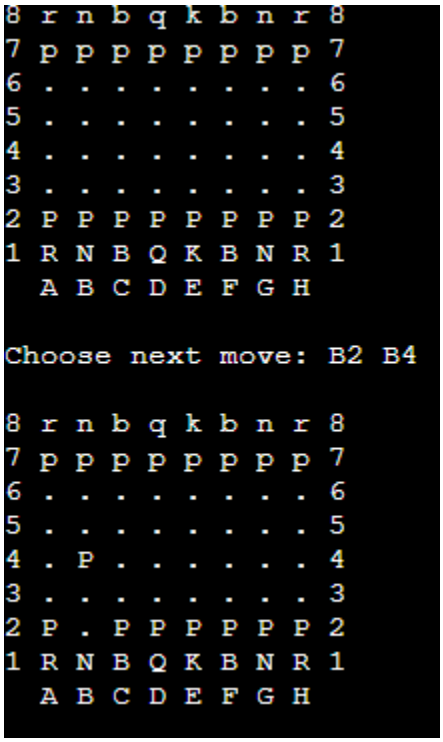


Figure 5: Output of pieceMove demonstrated by output of displayBoard.

6. Additional Features:
    - Although there is only one function described here, each piece has its own pieceMove function (eg. pawnMove, bishopMove, queenMove, etc.).
7. Error Handling:
    - Check if the new position where the user wants to move is a valid move.
    - If move i s invalid, function should return an error and notify the player the move is not valid.

3.6: Detailed description of isCheck

1. Purpose:
    - Enable checking of the board to analyze whether a enemy piece of the opposing color team is able to attack the king, thus signaling a potential end to the game.
2. Function Signature:
    - isCheck (struct Board *board, pieceColor color);
3. Parameters:
    - *board is a pointer to the board that is currently in play.
    - color refers to the color of the king that needs to be analyzed whether it is check or not
4. User Input:
    - N/A
5. Program Output:
    - Return 1 if the king being analyzed is currently in check due to an opponent's piece, or return 0 if the king being analyzed is not currently under any threat.
6. Additional Features:
    - Related to isCheckMate function.
7. Error Handling:
    - N/A

3.7: Detailed description of isCastle

1. Purpose:
    - Allowing the user to move two pawns at once involves moving the king toward a rook two squares and the rook moves a square over which the king is crossed.
2. Function Signature:
    - isCastle(Move *move, struct Board *board, pieceColor color);
3. Parameters:
    - *move is a pointer to the start and end coordinates of the move being executed.
    - *board is a pointer to the 8x8 array that is used to represent the chessboard and pieces.
    - color refers to the color of the side that is currently executing the castling maneuver.
4. User Input:
    - N/A
5. Program Output:
    - Return 1 if the castling maneuver is successful and has met all preliminary conditions. Returns 0 if castling maneuver cannot be executed due to preliminary conditions not being met.
6. Additional Features:
    - Check if conditions for castling are met, no pawn is between king and rook, neither pawn has moved, no squares between the king and rook is under attack.
7. Error Handling:
    - Handle the situation when the user tries to perform a castling while the king is in check or other illegal moves.
    - Notify the user if a castling condition is not fulfilled and they can’t perform a castling.

3.8: Detailed description of setPiece

1. Purpose:
    - allow each piece to be initialized and set on the board struct.
2. Function Signature:
    - setPiece(struct Board *board, int file, int rank, pieceType piece, pieceColor color);
3. Parameters:
    - *board is a pointer to the 8x8 board that represents the chessboard and pieces.
    - file and rank represent the position of a piece on the chessboard.
    - pieceType refers to what type of piece a chess piece is, such as a rook or a pawn.
    - color refers to the color of the piece being created, either black, white, or none.
4. User Input:
    - N/A
5. Program Output:
    - N/A
6. Additional Features:
    - utilized in initializeBoard.
7. Error Handling:
    - N/A.

3.9: Detailed description of capture

1. Purpose:
    - capture a piece within the chess game.
2. Function Signature:
    - capture(struct Board *board, struct Move *move);
3. Parameters:
    - *board is a pointer that refers to the 8x8 array that represents the chessboard within our game.
    - *move is a pointer that refers to the representation of a move being made in our game, with start and end coordinates.
4. User Input:
    - N/A
5. Program Output:
    - Board will be updated according to the move that captures an enemy piece.
6. Additional Features:
    - N/A
7. Error Handling:
    - N/A, checked in other functions, such as isValidMove.

3.10: Detailed description of isPromotion

1. Purpose:
    - The purpose of this function is to check whether the prerequisites for a promotion has been met, when a pawn has reached the opposite side of the board where it started from.

2. Function Signature:

    - isPromotion (Move *move, struct Board *board);

3. Parameters:

    - *board is a pointer to the 8x8 representation of the chessboard being used within the game.
    - *move is a pointer to the representation of a move being made in the game, with start and end coordinates.

4. User Input:

    - N/A

5. Program Output:

    - returns 1 if pawn passes all prerequisites for executing a promotion.

6. Additional Features:

    - Allows for the pawn to be changed into a queen, bishop, rook, or knight, as well as supporting no promotion being selected.

7. Error Handling:

    - N/A

3.11: Detailed description of promotePawn

1. Purpose:
    - promote a pawn to a user designated piece type after reaching the opponent's side of the board
2. Function Signature:
    - promotePawn (struct Board *board, int promoteChoice);
3. Parameters:
    - *board is the pointer to the board where all the chess pieces are located, and where the promotion of the pawn will take place
    - promoteChoice represents the user input in what piece type they wish to promote the pawn to (eg 1 for queen, 2 for rook, etc).
4. User Input:
    - N/A
5. Program Output:
    - the pawn is removed from the game, the square is set to a NULL/EMPTY piece, and then the desired piece is then created at the same square of the pawn after the move
6. Additional Features: N/A
7. Error Handling:
    - promoteChoice is guaranteed to be within a certain range of numbers, will error otherwise

3.12: Detailed description of movePiece

1. Purpose:
    - move a piece on the board.
2. Function Signature:
    - movePiece (struct Move *move, struct Board *board);
3. Parameters:
    - *move is a pointer to the move that is currently being executed, containing the start and end x and y coordinates.
4. User Input:
    - N/A
5. Program Output:
    - the function moves the piece to the new position on the board and then places a NULL piece at the newly vacated position on the board
6. Additional Features:
    - N/A
7. Error Handling:
    - error handling is performed by isValidMove, movePiece does not have any inherent error handling

3.13: Detailed description of generateLegalMoves

1. Purpose:
    - generate a 1-dimensional array of all possible legal moves for a certain color
2. Function Signature:
    - generateLegalMoves(struct Board *board, pieceColor color, Move moves[], int *moveCount);
3. Parameters:
    - *board is a pointer to the board that currently contains all the existing pieces
    - color is the color of the side that the list of legal moves will be generated for
    - moves[] is the 1 dimensional array that the legal moves will be added to
    - *moveCount is a pointer to the number of moves that the array contains
4. User Input:
    - N/A
5. Program Output:
    - function is a void function, but instead updates the array with the list of all possible moves for a certain color
6. Additional Features:
    - N/A
7. Error Handling:
    - N/A

3.14: Detailed description of aiTurn

1. Purpose:
    - functions as the AI/CPU for the player versus computer gamemode
2. Function Signature:
    - aiTurn (struct Board *board, pieceColor color);
3. Parameters:
    - *board is a pointer to the board that the current chess game being played on, also containing all chess pieces
    - color is the color that the AI is playing as
4. User Input:
    - N/A
5. Program Output:
    - the function is a void function, but will make a move
6. Additional Features: N/A
7. Error Handling:
    - N/A

Back Matter

Copyright Information

Index

Error Messages

1: “King not found on board, error” refers to when the king is somehow missing from the board, either through capture of the king piece or through an inaccurate board setup, and should be printed out after a move is made by either a black or white player.
2: “Computer cannot move” refers to when the computer is unable to make a move due to it generating a list of possible moves that is of size 0, meaning that it cannot make a move and it has somehow not activated the checkmate condition.
3: “Invalid coordinates” refers to when a user has typed in coordinates that are completely outside of the chessboard, such as (-1, -1), and is prompted to enter valid coordinates.
4: “No piece at that square” refers to when a user has typed in coordinates that select an empty square as the starting point for their move, which will lead to the user being prompted to enter in another set of coordinates to select their own colored pieces.
5: “Not your piece” refers to when a user has typed in coordinates that select a piece of the opposite color, and the user will be prompted by the program to put in a new set of coordinates to select a piece of the player’s chosen color.

6: “Can’t put a piece there” refers to selecting an end/destination coordinate for a piece that they have chosen to move that will end up outside of the chessboard, such as (9, 9); the user will then be prompted to start their turn again to make a valid move.
7: “Invalid move” will be output into the terminal when the move being made by the user has passed all other error checking, but has not passed the isValid function, indicating that the move is somehow not valid and prompting the user to start their turn again by entering in coordinates for the piece they want to move and where they want to move it to.