# Use case diagram

A diagram of a computer program

AI-generated content may be incorrect.

# Class diagram

# Use case scenarios

## Scenario 1: Valid Initial Position

|  |  |
| --- | --- |
| **Actor Action** | **System Response** |
| 1. The user starts the program and selects a chess piece. |  |
|  | 2. System verifies the piece |
| 3. User selects the color of the piece |  |
|  | 4. System applies that color to the piece |
| 5. User selects the initial place of the piece |  |
|  | 6. System validates the placement of the piece |

### Sub-flow: System validates initial piece placement

|  |  |
| --- | --- |
| **Actor Action** | **System Response** |
| 6. System validates the placement of the piece |  |
|  | 6.1 If incorrect the system displays an error message and invalidates the user input |

## Scenario 2: Valid Move

|  |  |
| --- | --- |
| **Actor Action** | **System Response** |
| 1. The user selects piece to move |  |
|  | 2. System validates the placement of the piece |
| 3. User selects where to move the piece |  |
|  | 4. System validates the new position |
|  | 5. System moves the piece to the new location |

### Sub-flow: System validates moved piece

|  |  |
| --- | --- |
| **Actor Action** | **System Response** |
| 4. System validates the new position |  |
|  | 4.1 If incorrect the system displays an error message and invalidates the user input and don’t move the piece |

# Purpose of classes

**Main**

**Purpose:** It handles user input to set up chess pieces, validates their positions, and checks if the pieces can move to a specified target position.

**ChessBoard**

**Purpose:** Represents the chessboard and provides methods to verify if a given coordinate is valid (e.g., within the 8x8 grid).

**Figure**

**Purpose:** The base class for all chess pieces. It contains common attributes like color, position (column and row), and piece type, as well as methods like moveTo() to determine if a move is valid.

**PieceType**

**Purpose:** An enumeration that defines the types of chess pieces (e.g., Pawn, Rook, Knight, Queen, King).

**PieceColor**

**Purpose:** An enumeration that defines the two possible colors of chess pieces: WHITE and BLACK.

**LocationX**

**Purpose:** An enumeration that defines the valid columns on a chessboard (A to H).

**IntChessBoard**

**Purpose:** An interface that defines the contract for chessboard-related operations, such as verifying coordinates.

**Pawn, Rook, Knight, Queen, King**

**Purpose:** These are specific implementations of the Figure class for each chess piece. Each class overrides the moveTo() method to implement the movement rules for that piece.

**Bishop**

**Purpose:** A chess piece class (excluded in this program's setup). It inherits from Figure and implements diagonal movement rules.

# Pseudocode

1. Start the application and initialize the GUI.

2. Create the main frame for the chess game:

- Set up the layout and size.

- Add a chessboard panel with an 8x8 grid.

- Initialize the board with alternating colors (white and gray).

3. Create a control panel with:

- Dropdowns for selecting piece type, color, row, and column.

- Buttons for placing, moving, and clearing pieces.

4. Initialize backend logic:

- Create a 2D array to represent the game board.

- Maintain a list of placed pieces and a set of added piece types.

5. Add functionality for the "Place Piece" button:

- Get the selected piece type, color, and position.

- Validate the position and ensure the piece is not already placed.

- Create the piece object and add it to the board.

- Update the GUI and backend logic.

6. Add functionality for the "Move Piece" button:

- Get the current and target positions.

- Validate that a piece exists at the current position.

- Check if the move is valid for the selected piece.

- Update the GUI and backend logic to reflect the move.

7. Add functionality for the "Clear Board" button:

- Remove all pieces from the board.

- Clear the backend data structures.

- Update the GUI.

8. Display the GUI and wait for user interaction.

9. End.