CCP6224 OBJECT-ORIENTED ANALYSIS AND DESIGN

Project

Trimester 2410

By <<Group E>>

Prepared by:

Student ID	Name	Phone Number	Email
1231303504	Haizatul Nazirah Nizam Binti Hairunizam	0134639373	1231303504@student.mmu.edu.my
1231302985	Nur Alia Shazwani Binti Mohd Nazri	0193282645	1231302985@ student.mmu.edu.my
1231303620	Nur Iman binti Mohamad Idros	0132091120	1231303620@student.mmu.edu.my
1211108662	Vinoshnee A/P Alagiri	01126825740	1211108662@student.mmu.edu.my

1 Table of Contents

1 Compile and Run Instructions

System requirements:

1. Java Runtime Environment (JRE) installed
2. Supported Operating Systems: Windows, macOS

Compilation and Execution Steps:

1. **Preparation** - Ensure Java Development Kit (JDK) is installed on your system. Open a terminal or command prompt and navigate to the project's root directory.
2. **Compilation** - Compile the Java source files using the following command:
   ```
   javac -d bin *.java
   ```
   This command compiles all Java files in the src/boardgame directory and places the compiled .class files in the Bin/boardgame directory.
3. **Running the Application**- Execute the game using the following command:
   ```
               java -cp bin KwazamChess
   ```

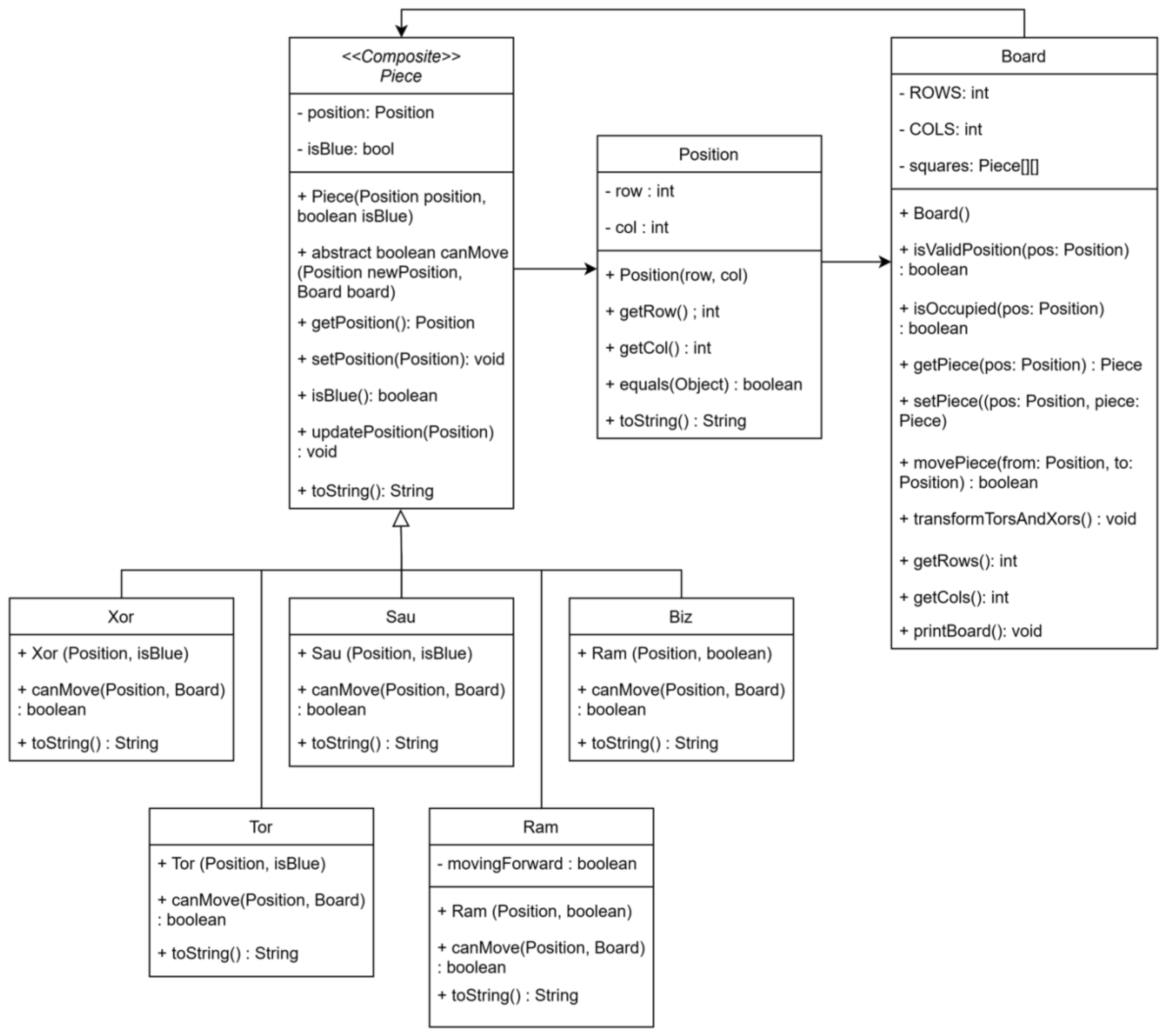
Troubleshooting

- Verify Java installation by running java -version
- Confirm all source files are present in the src/boardgame directory
- Ensure you are executing the command from the project's root directory

2 UML Class Diagram

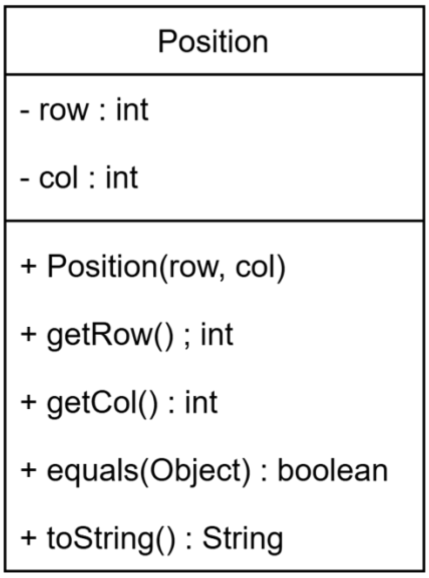
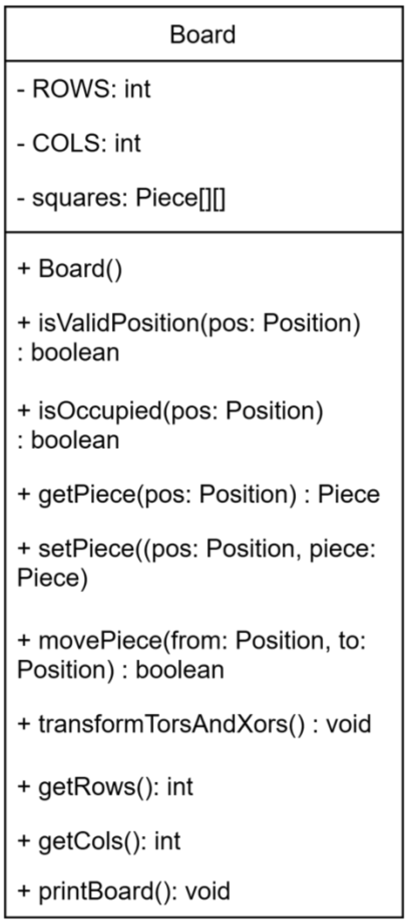
2.1 Class Diagram

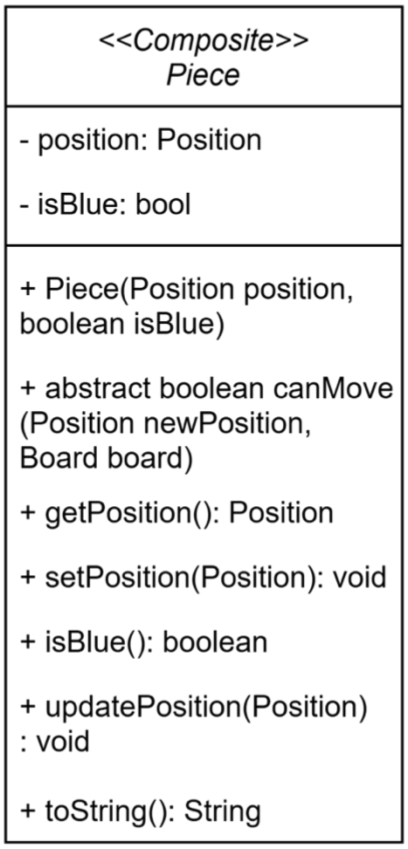
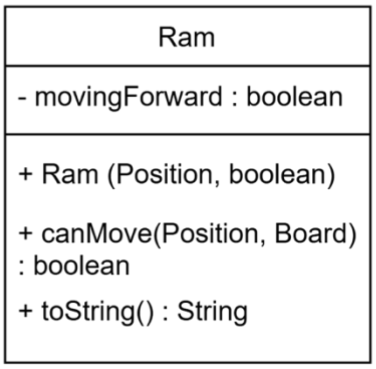
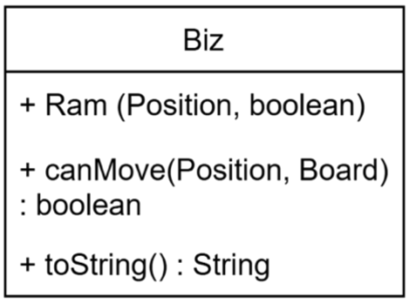
The diagram below is the class diagram for this project.

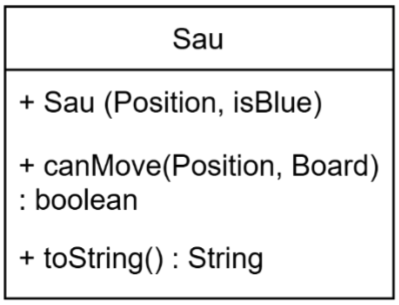
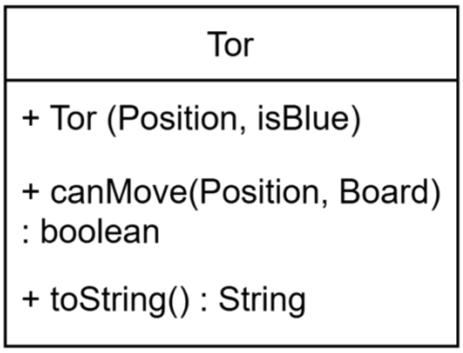
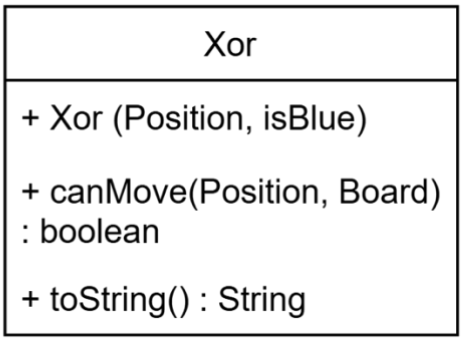


<<Composite>>
Piece
- position: Position
- isBlue: bool
+ Piece(Position position, boolean isBlue)
+ abstract boolean canMove (Position newPosition, Board board)
+ getPosition(): Position
+ setPosition(Position): void
+ isBlue(): boolean
+ updatePosition(Position) : void
+ toString(): String
Position
- row : int
- col : int
+ Position(row, col)
+ getRow() ; int
+ getCol() : int
+ equals(Object) : boolean
+ toString() : String
Board
- ROWS: int
- COLS: int
- squares: Piece[][]
+ Board()
+ isValidPosition(pos: Position) : boolean
+ isOccupied(pos: Position) : boolean
+ getPiece(pos: Position) : Piece
+ setPiece((pos: Position, piece: Piece)
+ movePiece(from: Position, to: Position) : boolean
+ transformTorsAndXors() : void
+ getRows(): int
+ getCols(): int
+ printBoard(): void
Xor
+ Xor (Position, isBlue)
+ canMove(Position, Board) : boolean
+ toString() : String
Sau
+ Sau (Position, isBlue)
+ canMove(Position, Board) : boolean
+ toString() : String
Biz
+ Ram (Position, boolean)
+ canMove(Position, Board) : boolean
+ toString() : String
Tor
+ Tor (Position, isBlue)
+ canMove(Position, Board) : boolean
+ toString() : String
Ram
- movingForward : boolean
+ Ram (Position, boolean)
+ canMove(Position, Board) : boolean
+ toString() : String

Figure 2.1 Class Diagram

Class	Description	Attributes/Operations
Position	Represent the chess piece position in the board	 Position - row : int - col : int + Position(row, col) + getRow() ; int + getCol() : int + equals(Object) : boolean + toString() : String
Board	Represent the board for the game itself	 Board - ROWS: int - COLS: int - squares: Piece[][] + Board() + isValidPosition(pos: Position) : boolean + isOccupied(pos: Position) : boolean + getPiece(pos: Position) : Piece + setPiece((pos: Position, piece: Piece) + movePiece(from: Position, to: Position) : boolean + transformTorsAndXors() : void + getRows(): int + getCols(): int + printBoard(): void

Pieces	Getters and setters for the chess pieces	 <<Composite>> Piece - position: Position - isBlue: bool + Piece(Position position, boolean isBlue) + abstract boolean canMove (Position newPosition, Board board) + getPosition(): Position + setPosition(Position): void + isBlue(): boolean + updatePosition(Position) : void + toString(): String
Ram	Represent a chess piece that can only move forward one step and cannot skip over other piece but is able to turn back and move again after reaching the opposite end of the board	 Ram - movingForward : boolean + Ram (Position, boolean) + canMove(Position, Board) : boolean + toString() : String
Biz	Represent the chess piece that move 3x2 L shape in any orientation and the only one chess piece that can skip over other pieces	 Biz + Ram (Position, boolean) + canMove(Position, Board) : boolean + toString() : String

Sau	Represent the chess piece that can only move one step in any direction and will automatically end the game if it was captured by the opposing side	 Sau + Sau (Position, isBlue) + canMove(Position, Board) : boolean + toString() : String
Tor	Represent the chess piece that can only move orthogonally but in any distance and will transforms to Xor chess piece after two turns.	 Tor + Tor (Position, isBlue) + canMove(Position, Board) : boolean + toString() : String
Xor	Represent the chess piece that can only move diagonally but in any distance and will transforms to Tor chess piece after two turns.	 Xor + Xor (Position, isBlue) + canMove(Position, Board) : boolean + toString() : String

2.2 Classes Utilising Design Pattern

Design Patterns	**Class**	**Description /Reason**
Creational		
Singleton Pattern	Board	As it ensures there is only one instance during the game
Factory Method Pattern	Piece	Piece is an abstract and acts as blueprints for creating specific piece types using factory method for its subclasses (Ram, Biz, Sau, Tor and Xor)
Structural		
Composite Pattern	Piece	Have a parent – child relationship where all subclasses inherit the common behaviour and attributes from the Piece class for its subclasses (Ram, Biz, Sau, Tor and Xor)
Behavioural		
Strategy Pattern	Piece	Piece class and its canMove method which is overridden in its subclasses (Ram, Biz, Sau, Tor and Xor) so it provides a way to define movement logic for each piece independently

3 Use Case

3.1 Use Case

This use case table covers the key functionalities of the Kwazam Chess game that includes the description of each use case and the actors involved.

Use Case	**Description**
Move Piece	Player selects a piece and moves it to a valid destination square
Capture Piece	Player moves their piece to an opponent's piece, capturing it
Select Piece	Player chooses a piece they control to move
Validate Move	System checks if the player's selected move is valid
Piece Transformation	System automatically transforms Tor pieces to Xor, and vice versa, every 2 turns
Start New Game	Player initiates a new game, setting up the initial board state
Save Game	Player saves the current game state to a file
Load Game	Player loads a previously saved game state from a file
Reset Game	Player resets the game, clearing the board and starting a new game
Determine Winner	System determines the winner of the game based on capture of opponent's Sau piece
End Game	Game ends when one player's Sau piece is captured

3.2 Use Case Diagram

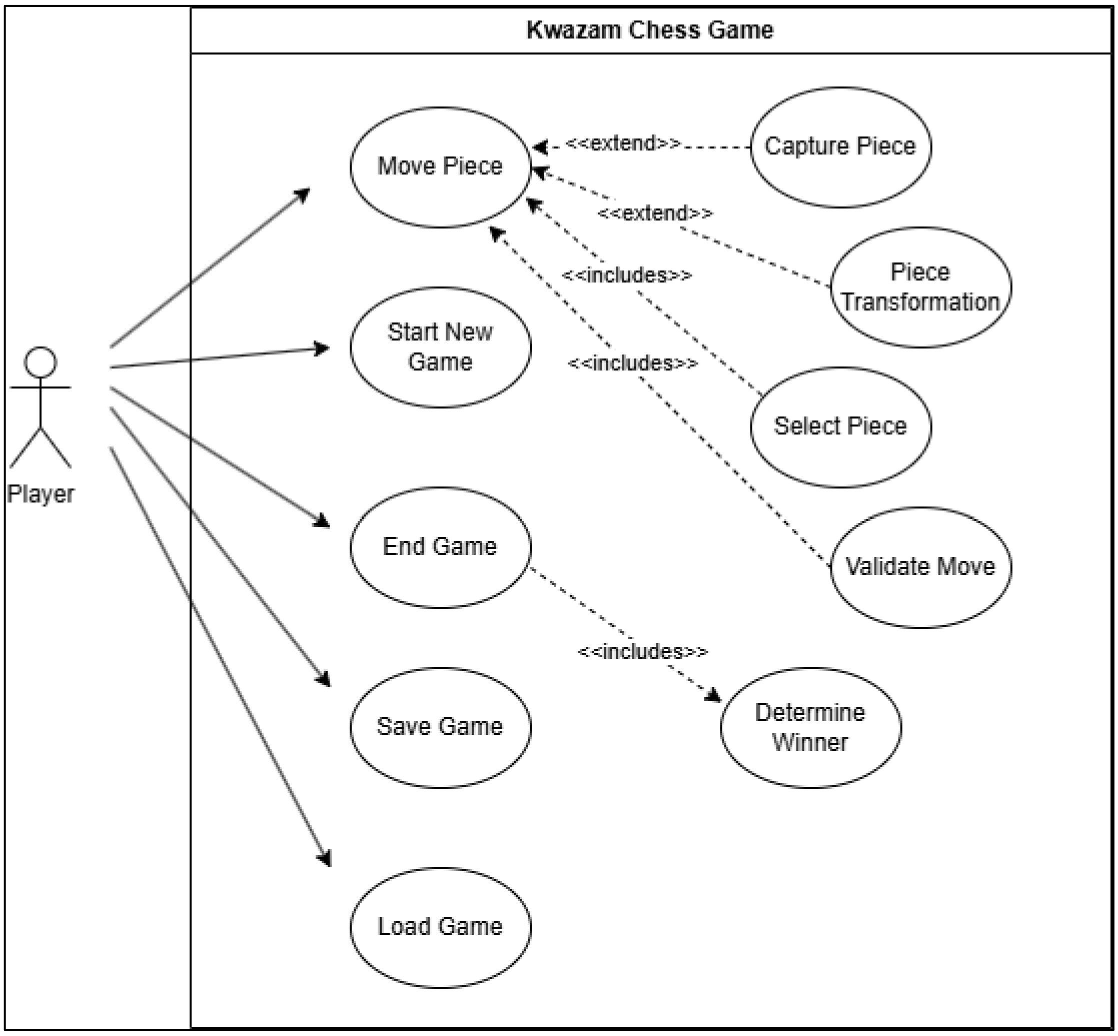


Figure 3.2 Use Case Diagram

4 Sequence Diagram

4.1 Move Piece

This sequence diagram shows the process of moving a piece when a player makes a move in the game.

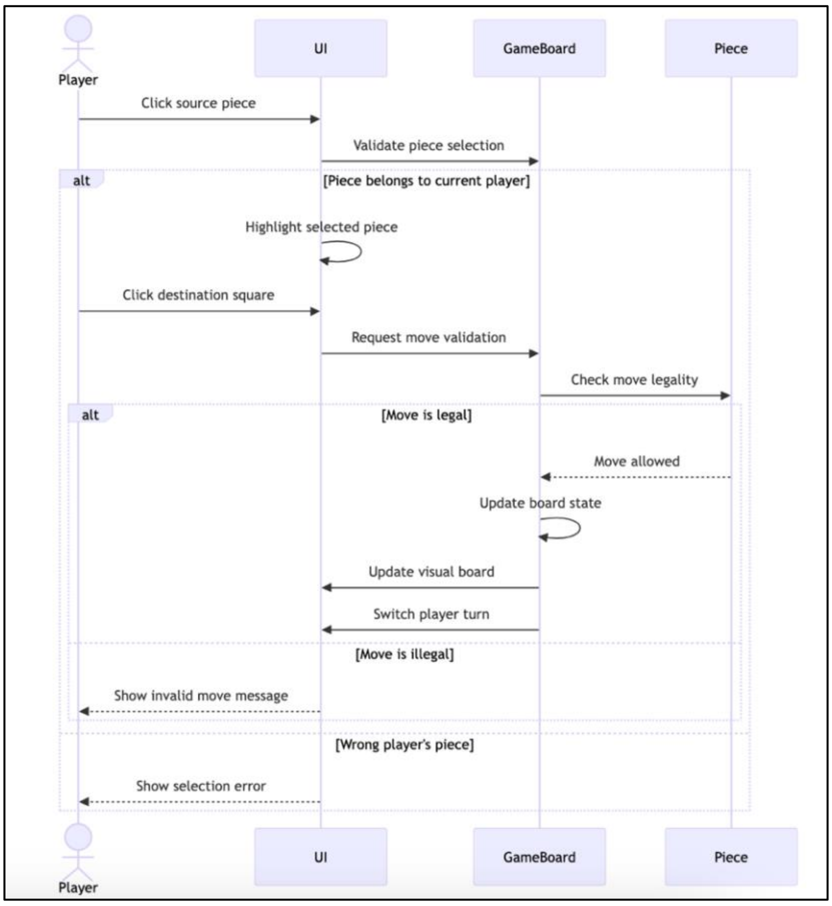


Figure 4.1 Move piece sequence diagram

4.2 Capture Piece

This sequence diagram shows the process of capturing a piece when a piece is move to capture the enemy's piece.

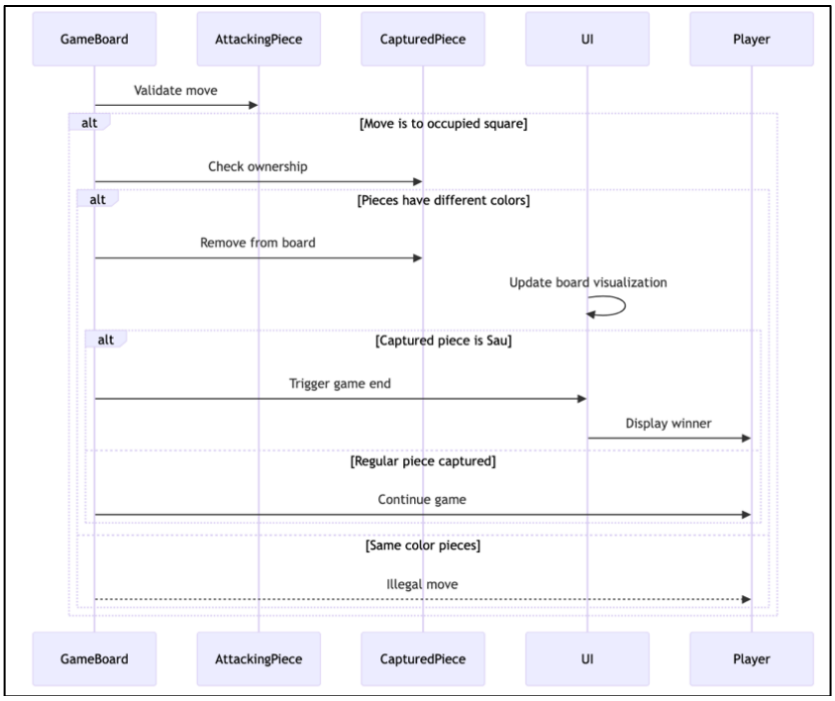


Figure 4.2 Capture piece sequence diagram

4.3 Piece Transformation

This sequence diagram shows the process when Xor ↔ Tor piece transformed after two turns.

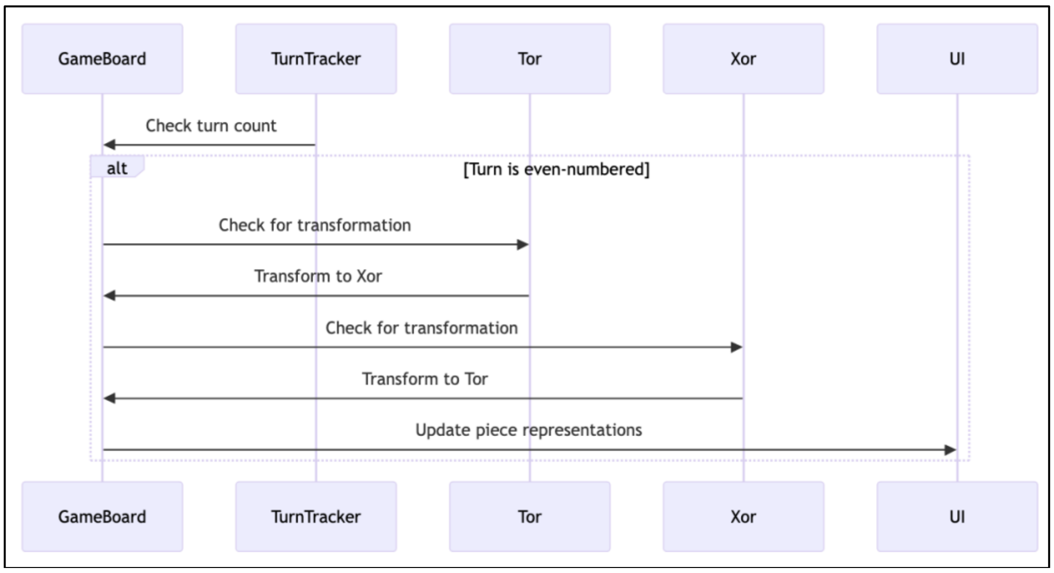


Figure 4.3 Piece transformation selection diagram

4.4 Piece Selection

This sequence diagram shows the process of when player selecting the piece to move before moving the piece.

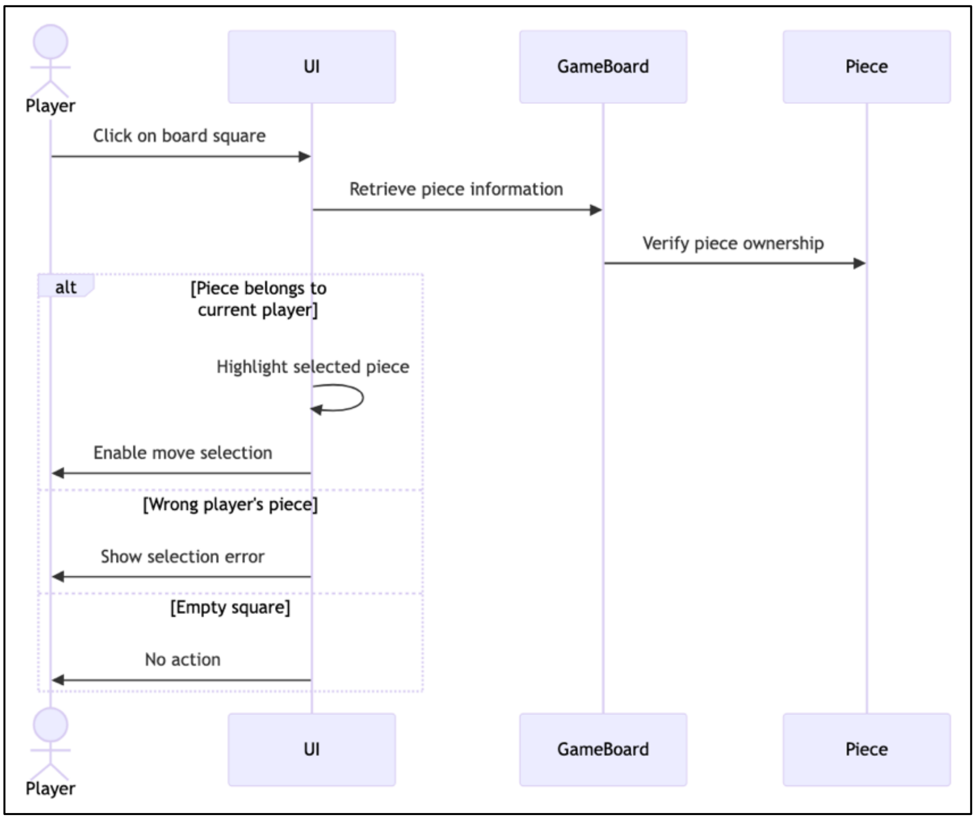


Figure 4.4 Piece selection sequence diagram

4.5 New Game

This sequence diagram shows the process when player choose to start a new game.

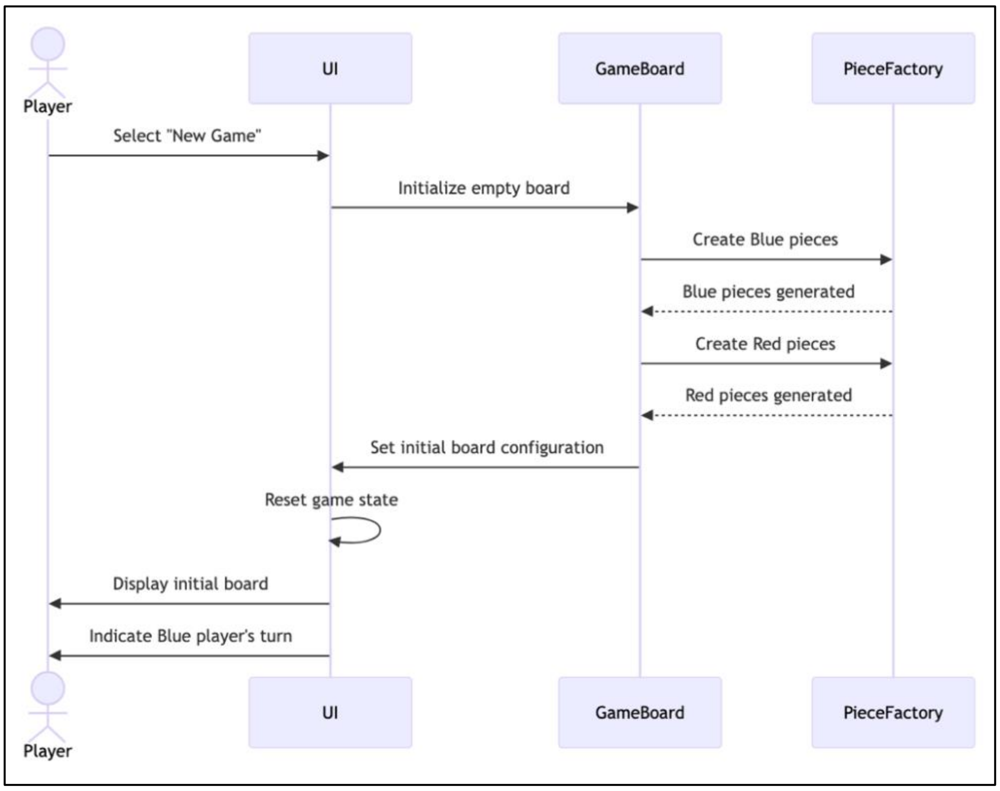


Figure 4.5 New game sequence diagram

4.6 End Game

This sequence diagram shows the process where a condition are met and the game is finished.

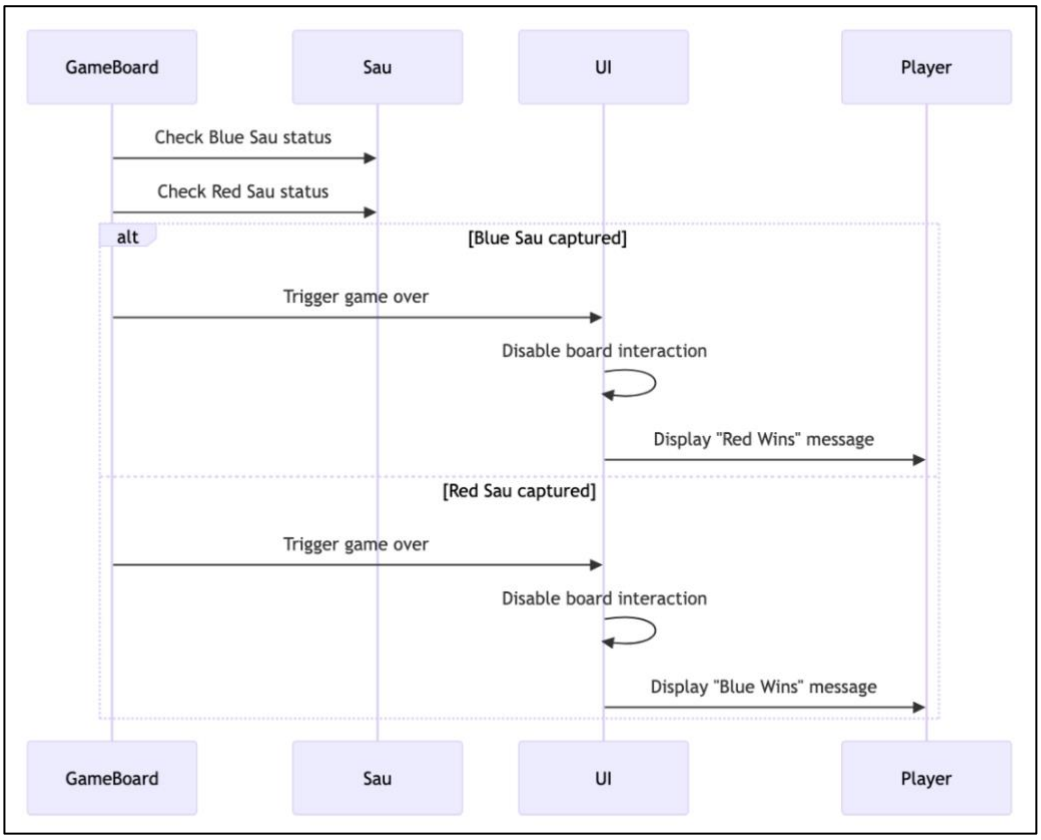


Figure 4.6 End game sequence diagram

4.7 Save Game

This sequence diagram shows the process when player choose to save the current game to load it to resume play for another time.

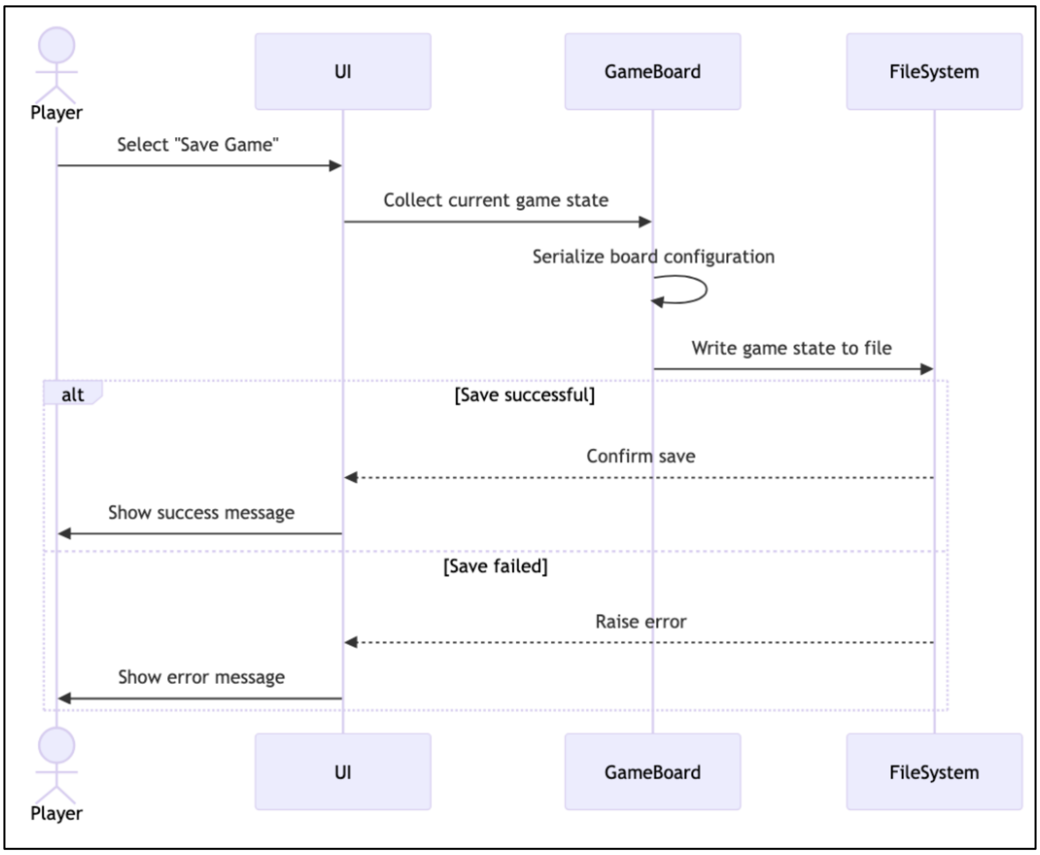


Figure 4.7 Save game sequence diagram

4.8 Load Game

This sequence diagram shows the process when player load a saved game to resume the game.

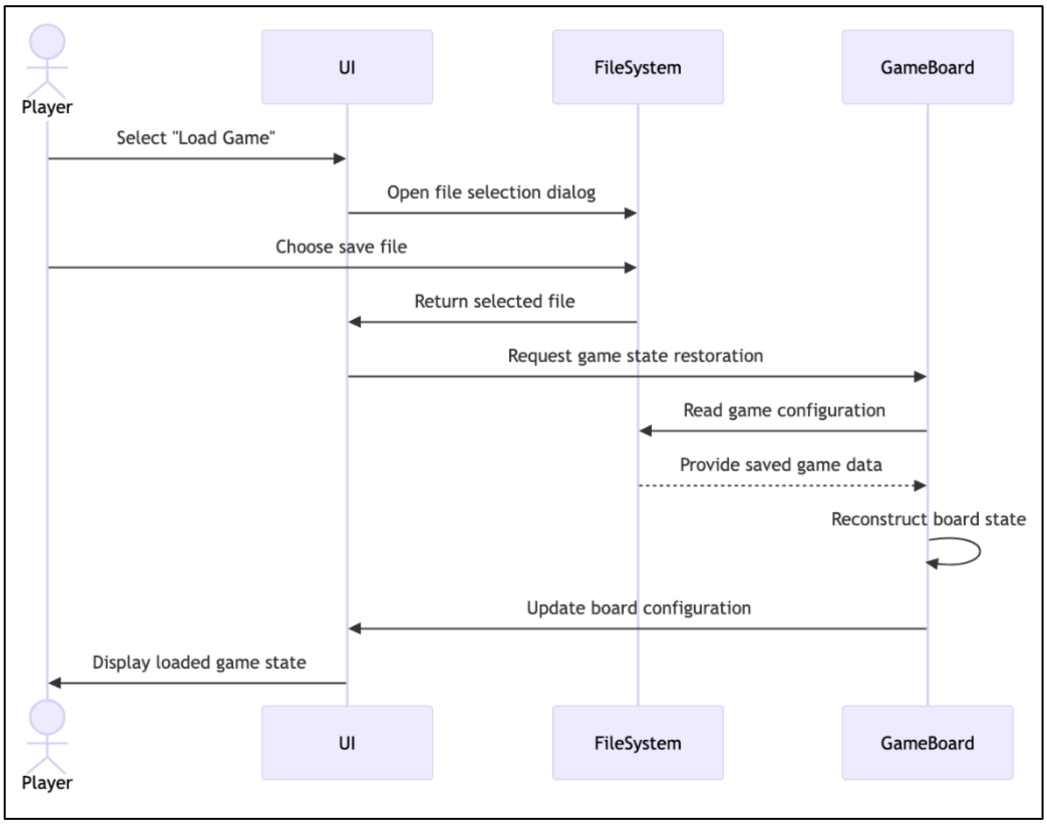


Figure 4.8 Load game sequence diagram

4.9 Validate Move

This sequence diagram shows the process when the gameboard request for validation before moving a piece to another parameter.

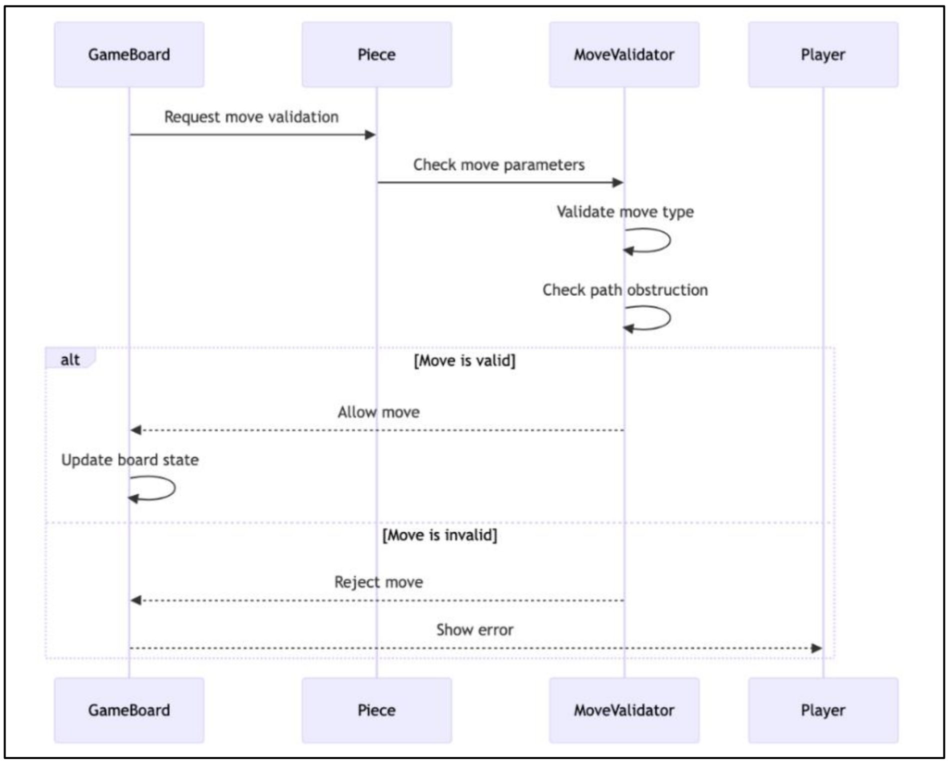


Figure 4.9 Validate move sequence diagram

4.10 Determine Winner

This sequence diagram shows the process when determining which player are the winner by checking each player's Sau piece.

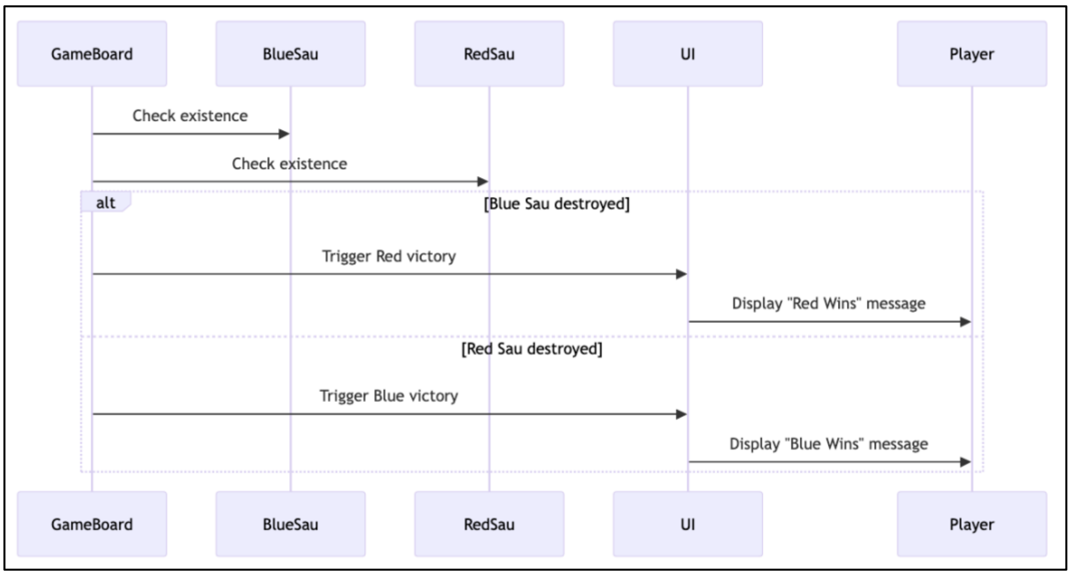


Figure 4.10 Determine winner sequence diagram

5 User Documentation

5.1 Introduction

The Kwazam Chess program is a digital recreation of the classic Kwazam board game, designed to deliver a modern, intuitive, and interactive gaming experience. It is a two-player game where players take turns strategically moving unique pieces on a grid-based board with the ultimate goal of capturing the opponent's Sau piece. The program enforces all game rules, manages piece interactions, and features a graphical user interface (GUI) for a seamless gameplay experience. This documentation provides an overview of the program's functionality and guidance for using its features effectively.

5.2 User Interface Overview

The program features two main interfaces:

1. Home Page
2. Game Page

5.2.1 Home Page

The Home Page allows users to begin their journey with Kwazam Chess. Users can select from several options:

- **Create New Game**: Starts a new match with all pieces in their initial positions.
- **Resume Game**: Allows players to continue a previously paused game by resuming from an autosave file.
- **Load Game**: Loads a saved game from a file chosen by the user.
- **How to Play** : Displays the rules of Kwazam Chess and a guide on using the application.
- **Exit:** Closes the application.

5.2.2 Game Page

The Game Page is the central interface for playing the game. It includes:

- A grid-based board displaying all active pieces.
- Dynamic turn management, where the board flips to the current player's perspective.
- Highlights for valid moves when a piece is selected.

The Menu in the Game Page provides additional options:

- Resign: Quit the game and declare the opponent the winner.
- Save: Save the current game progress to a file.
- Load: Load a previously saved game.
- Home: Return to the Home Page.

5.3 Piece Movement Rules

There are 5 unique pieces in this Kwazam Chess for each player. Total for all these pieces that are available on the board for both layers is 10 pieces.

Piece Name	Piece Picture	Piece’s Movement	Piece’s Unique Features
Ram		Moves one step forward only and cannot move backward initially.	• Reverses direction upon reaching the board's edge • Cannot jump over pieces; only moves to adjacent empty squares.
Biz		Moves in an L-shape (3x2 pattern) in 8 possible configurations, jumping over other pieces.	Precise movement (2 squares in one direction, 1 square perpendicular).
Tor		Moves any number of squares horizontally or vertically but cannot jump over pieces.	Changes into Xor after 2 complete turns (1 blue + 1 red)
Xor		Moves diagonally any number of squares, without jumping over pieces.	Reverts to Tor after 2 complete turns.
Sau		Moves one square in any direction (8 possibilities).	The game ends immediately if Sau is captured.

5.4 Game Instructions

1. Objective: Capture your opponent’s Sau piece or force them to resign.
2. Gameplay:
   - The Blue player begins the game.
   - Players alternate turns, moving one piece at a time.
   - Click a piece to see valid moves (highlighted tiles). Select a highlighted tile to complete the move.
3. Winning Conditions:
   - The game ends when a player captures the opponent's Sau piece.
   - A player can also win if the opponent resigns.
   - The game begins with the first move assigned to the blue player.

Each player can only interact with their respective pieces.

1. To move a piece:
   - Click on the piece you want to move.
   - Valid moves will be highlighted on the board.
2. Select a highlighted destination to complete the move.
   - The turn then switches to the opponent, following the same procedure.

The match ends when:

- A player captures the opponent's Sau piece.
- A player resigns, forfeiting the match.

5.5 File Management Information

1. Save Files:
   - Players can save their current game progress using the Save option. The saved file is stored in a human-readable format and contains the position of all pieces and the current game state.
   - Autosave files are created when the user leaves the game or returns to the Home Page.
2. Loading Files:
   - Saved games can be loaded using the Load Game option, either from the Home Page or the Game Page menu. Players can select files from the predefined save local.