COMP3211: Software Engineering
Software Requirements Specification
Jungle Game

Student Name	Student ID	Contribution
Sun Hansong	23097775d	33.3%
Fu Guanhe	23097455d	33.3%
Wang Yuqi	23110134d	33.3%

1. Preface

Readership This document constitutes the *Software Requirements Specification (SRS)* for *Jungle Game*, a command-line turn-based board game. Intended audience for this document are:

- End users wo wish to understand game's capabilities
- Developers who will implement the system
- Quality assurance (QA) personnel wanting to validate requirement satisfaction

Version History: ver1.0 (Nov 19, 2025). Initial release. This version defines complete functional and non-functional requirements. Future version might want to extend capabilities by adding AI bot mode and online PvP modes.

2. Introduction

The *Jungle Game* implements a digital version of the traditional Chinese board game known as *Dou Shou Qi*, translating literally as "Game of Fighting Animals." The game involves two players commanding eight animal pieces as their armies and battling against each other in a turn-based manner. Victory is determiend either by capturing all opponent pieces or by occupying the opponent's den.

2.1. Need for the System:

Traditional physical board game needs face-to-face interaction and physical pieces. For a niche board game like the *Jungle Game*, these might not be readily available when on the go. A digitial implementation address this by allowing *Jungle Game* lovers to enjoy the game without needing a physical board. The command-line interface (CLI), while rudimentary, ensures accessibility across diverse platforms. Future development can also easily extend upon this and implement graphical user interfaces (GUI).

2.2. System Function

The *Jungle Game* systgem provides a complete gameplay experience through a CLI. The system shall maintain the game board state, and enforce all traditional *Jungle Game* rules. It should also provide visual represnetation of the current board state through ASCII art rendering, game persistence through save and load operations, and replay mechanisms.

As a foundational platform, the system shall not include advanced features such as a graphical user interface, network multiplayer capabilities, and artificial intelligence (AI) opponents, or game rule variants. The system should stick to the basics and allow extensibility for future users.

2.3. Business & Strategic Context

This system serves as a free educational software for a group project at the *Hong Kong Polytechnic University* COMP3211 Software Engineering course. It is intended to demonstrate studnet's understanding in requirements analysis, code architecture design, implementation, and testing abilities. The project emphasizes software quality in terms of maintainability, modularity, and reliability. In essence, the implementation provides a foundation for understanding software engineering development processes applicable to larger commercial systems students might encounter in the future; it is free, open-sourced, educational, and thus non-commercial. It does not fit into any business strategies.

3. Glossary

3.1. Game-specific Glossary

1. **Capture** The act of moving a piece onto a square occupied by an opponent's piece, removing the opponent's piece from the game.
2. **Den** Special square at the center of each player's back row, representing the player's home base. The game ends when a piece enters their opponent's den.
3. **Animal Piece** A game piece one of eight animal types (Elephant, Lion, Tiger, Leopard, Wolf, Dog, Cat, Rat), each with a distinct rank that determines the capture logic.
4. **Piece Rank** A numerical value (1-8) associated with each piece type. Higher ranked pieces can capture equal or lower ranked pieces, except for mice (lowest rank piece) which can capture elephants (highest rank piece).
5. **River/Water Square** One of twelve special squares (arranged in two 2×3 rectangular areas) that most pieces cannot enter, except for rats which can traverse and lions/tigers which can leap across.
6. **Square** An individual cell of the board, identified by column (A-G) and row (1-9) coordinates.
7. **Trap** A special square adjacent to each den that weakens pieces standing on it. Any piece that enters the opponent's trap will have their rank downgraded to the lowest (temporarily), allowing any opposing piece to capture them.
8. **Turn** A single move by one player, after which control passes to the opponent.
9. **Withdraw** The act of undoing a player's previous move. Each player has a limited quota of withdraw operations.
10. **Board** The 7×9 grid of squares on which the game is played. It contains land squares, water squares, trap squares, and den squares.

3.2. Implementation-specific Glossary

1. **Command** Text string entered by a user to instruct the system to perform an action. E.g., moving a piece, withdrawing a move, or saving the game state.
2. **Lower Player** The player whose pieces begin on rows 1-3 of the board (lower portion).
3. **Upper Player** The player whose pieces begin on rows 7-9 of the board (upper portion)
4. **Model Package** The collection of software modules implementing the main game states and game rule enforcement logics. It is isolated from user interface and input/output.
5. **Replay** A mode where the system replays a previously completed game by reading from a stored game file then executing recorded moves sequentially.
6. **Session** A continuous period of user interaction with the system, from launching the program to terminating it, which may or may not encompass multiple games.

3.3. Technical Glossary

1. **Command Line Console (CLI)**: text-based system for interacting with a computer by typing commands

4. User Stories

- **US1**: As a player, I want to start a new game.
- **US2**: As a player, I want to end an ongoing game.
- **US3**: As a player, I want to name the players with input or randomly generated strings to better differentiate the players.
- **US4**: As a player, I want to play the game via the command line console.
- **US5**: As a player, I want to see the status of the game, including all the remaining pieces and their positions on the game board, as well as the next player to make a move.
- **US6**: As a player, I want to take back at most 3 moves in each game.
- **US7**: As a player, I want to record all the information about a game, including the players and their moves, into a file with an extension name ".record".
- **US8**: As a player, I want to use the record from a file with an extension name ".record" to replay a game.
- **US9**: As a player, I want to save the current game to a file with an extension name ".jungle". US10. As a player, I want to load a game from a file with an extension name ".jungle" and continue playing the game.

5. User Requirements Definition

This section describes services that the system shall provide to its users, expressed in natural language, intentionally avoiding implementation detail to be understandable by non-technical stakeholders.

5.1. Functional User Requirements (UR-F)

- **UR-F1**: System ***shall*** accept "startNewGame" command from main menu. (US1)
- **UR-F2**: System ***shall*** create new game session on receiving "startNewGame". (US1)
- **UR-F3**: System ***shall*** accept "close" command during active gameplay. (US2)
- **UR-F4**: System ***shall*** return to main menu on "close" without forced saving. (US2)
- **UR-F5**: System ***shall*** prompt for upper player name at game start. (US3)
- **UR-F6**: System ***shall*** prompt for lower player name at game start. (US3)
- **UR-F7**: System ***shall*** generate random name if user provides empty input. (US3)
- **UR-F8**: System ***shall*** use command-line interface exclusively for all interactions. (US4)
- **UR-F9**: System ***shall*** display board after every game state change. (US5)
- **UR-F10**: System ***shall*** show all remaining pieces and positions on board. (US5)
- **UR-F11**: System ***shall*** indicate which player's turn using distinct prompt. (US5)
- **UR-F12**: System ***shall*** visually differentiate upper and lower player pieces. (US5)
- **UR-F13**: System ***should*** display coordinate labels on board for move identification.
- **UR-F14**: System ***should*** include legend explaining symbols on first board display.
- **UR-F15**: System ***shall*** enforce single-step orthogonal movement for all pieces.
- **UR-F16**: System ***shall*** prohibit pieces from moving into own den.
- **UR-F17**: System ***shall*** allow rats to enter water squares.
- **UR-F18**: System ***shall*** allow lions and tigers to leap across rivers.
- **UR-F19**: System ***shall*** enforce rank-based capture between pieces.
- **UR-F20**: System ***shall*** allow rat to capture elephant.
- **UR-F21**: System ***shall*** prohibit elephant from capturing rat.
- **UR-F22**: System ***shall*** allow any piece to capture trapped opponent piece.
- **UR-F23**: System ***shall*** prohibit capture between water and land pieces.
- **UR-F24**: System ***shall*** reject illegal moves without altering game state.

- **UR-F25**: System ***should*** display explanatory message for each illegal move.
- **UR-F26**: System ***shall*** detect when piece enters opponent's den.
- **UR-F27**: System ***shall*** declare winner when den is occupied.
- **UR-F28**: System ***shall*** detect when player has no legal moves remaining.
- **UR-F29**: System ***shall*** declare opponent winner when player cannot move.
- **UR-F30**: System ***should*** automatically save replay file upon game victory.
- **UR-F31**: System ***shall*** accept "withdraw" command from current player. (US6)
- **UR-F32**: System ***shall*** restore board to state before withdrawn move. (US6)
- **UR-F33**: System ***shall*** limit each player to three withdraw operations. (US6)
- **UR-F34**: System ***shall*** reject withdraw when quota is exhausted.
- **UR-F35**: System ***shall*** restore captured pieces during withdraw reversal.
- **UR-F36**: System ***shall*** maintain current player turn after successful withdraw.
- **UR-F37**: System ***should*** display remaining quota after successful withdraw.
- **UR-F38**: System ***shall*** accept "saveGame" command with optional filename. (US9)
- **UR-F39**: System ***shall*** save complete game state to ".jungle" file. (US9)
- **UR-F40**: System ***shall*** include board configuration in saved file. (US9)
- **UR-F41**: System ***shall*** include player names in saved file. (US9)
- **UR-F42**: System ***shall*** include withdraw quotas in saved file. (US9)
- **UR-F43**: System ***shall*** accept "loadGame" command from main menu. (US10)
- **UR-F44**: System ***shall*** restore complete game state from ".jungle" file. (US10)
- **UR-F45**: System ***shall*** resume gameplay from loaded position. (US10)
- **UR-F46**: System ***should*** generate timestamp-based filename if none provided.
- **UR-F47**: System ***should*** display confirmation message after successful save.
- **UR-F48**: System ***shall*** accept "saveReplay" command with optional filename. (US7)
- **UR-F49**: System ***shall*** save move history to ".replay" file. (US7)
- **UR-F50**: System ***shall*** include player names in replay file. (US7)
- **UR-F51**: System ***shall*** accept "watchReplay" command from main menu. (US8)
- **UR-F52**: System ***shall*** enter replay mode after loading replay file. (US8)
- **UR-F53**: System ***shall*** accept "next" command in replay mode. (US8)
- **UR-F54**: System ***shall*** advance replay forward one move on "next". (US8)
- **UR-F55**: System ***shall*** accept "prev" command in replay mode. (US8)
- **UR-F56**: System ***shall*** reverse replay backward one move on "prev". (US8)
- **UR-F57**: System ***shall*** accept "exit" command to leave replay mode. (US8)
- **UR-F58**: System ***shall*** return to main menu on replay exit.

5.2. Non-Functional User Requirements (UR-NF)

- **UR-NF1**: System ***should*** validate moves within 500 milliseconds.
- **UR-NF2**: System ***should*** load saves within 2 seconds.
- **UR-NF3**: System ***should*** render board display within 200 milliseconds.
- **UR-NF4**: System ***should*** enable gameplay after 5 minutes manual reading.
- **UR-NF5**: System ***should*** provide specific reason for each move rejection.
- **UR-NF6**: System ***should*** fit board display in standard terminal window.

6. System architecture

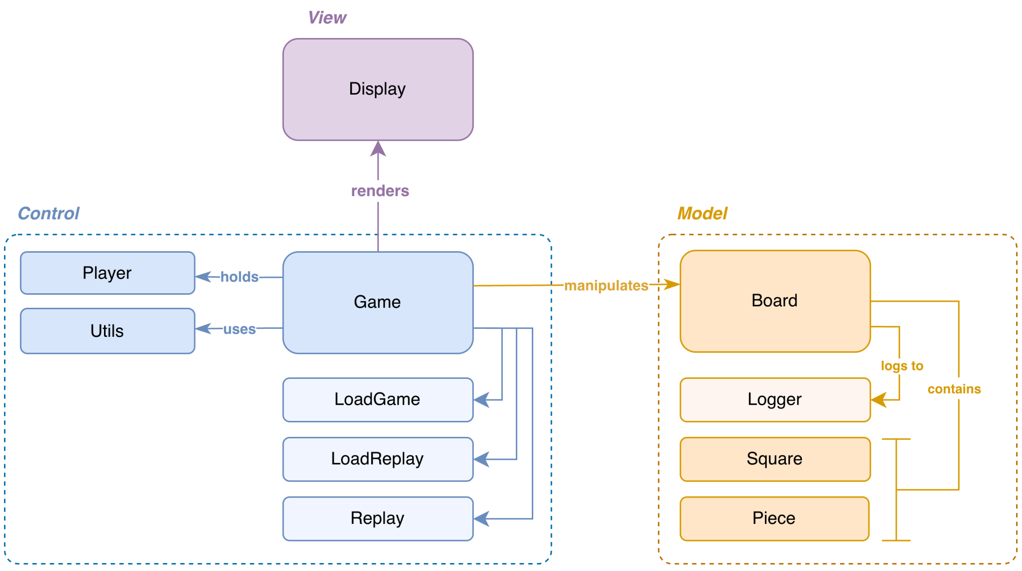

Figure 1: Our code structure demonstrating extensibility and modularity. The diagram illustrates the high-level separation of responsibility in our *Jungle Game* implementation. The **Control layer (Blue)** contains the Game controller that orchestrates gameplay, alongside utility components like LoadGame, LoadReplay, and Replay that handle different execution modes. **View layer (Purple)** consists solely of the stateless Display component, with a unidirectional rendering connection. **Model layer** (Yellow) contains domain logic within Board, Logger, Square, and Piece classes. Arrows denote control flow.

6.1. Main Architectual Pattern

Our implementation for the *Jungle Game* adopts **Model-View-Controller (MVC)** pattern. This pattern separates the application into three interconnected components. The MVC pattern was selected because the game requires multiple way to view game state (normal gameplay mode vs. replay mode). Its components are instantiated as follows in the *Jungle Game* context:

Model Within the `Model` package. This layer contains all domain logic and game state. The `Board` class acts as the central component, maintaining a 7×9 grid of `Square` objects, each potentially containing a `Piece` object. The `MovingValidator` enforces movement rules specific to the *Jungle Game*, such as terrain constraints (river, trap, den), piece capture mechanisms, and specific abilities (rat swimming, tiger/lion leaping); at each round, the `Board` query the validator with specific move attempts. The `Logger` component records all game events as reversible operations, allowing for withdraw and replay.

View Implemented through the `Display` class. It renders the board state to the console using Unicode box-drawing characters. The view is stateless and purely reactive; in other words, it accepts a `Board` reference and produces formatted output without maintaining any game states. The `Display` class have a few methods worth mentioning. The `displayBoard` method traverses the board grid and foramt each square according to tis type (normal, river, trap, den) and occupying piece. The players are distinguished by uppercase vs. lowercase letters.

Controller The `Game` class functions as the primary controller. It orchestrates the game loop and mediates user input and model operations. The `Game` class has a reference to the `Board` model and two `Player` objects. Upon receiving user commands, the `Game` object delegates the validation and execution to the model layer, then triggers view update after successful state modifications by the model.

6.2. Complementary Architectual Pattern

Beyond MVC, our implementation also incorporates a **Repository Pattern** via the `Logger` class. The logger maintains a centralized event storage that all game state modification flows through. The serialization mechnaism for saving game is effectively a simple file-based repository, persisting the entire object graph to a `.jungle` file.

7. System Requriements Specification (SRS)

7.1. Functional Requirements (SRS-FR)

- **SRS-FR1**: System ***shall*** create 7-column by 9-row grid on game start.
- **SRS-FR2**: System ***shall*** place lower den at coordinate (D, 1).
- **SRS-FR3**: System ***shall*** place upper den at coordinate (D, 9).
- **SRS-FR4**: System ***shall*** place lower traps at (C, 1), (D, 2), (E, 1).
- **SRS-FR5**: System ***shall*** place upper traps at (C, 9), (D, 8), (E, 9).
- **SRS-FR6**: System ***shall*** place river squares at columns B-C, rows 4-6.
- **SRS-FR7**: System ***shall*** place river squares at columns F-G, rows 4-6.
- **SRS-FR8**: System ***shall*** initialize remaining squares as normal land.
- **SRS-FR9**: System ***shall*** place lower elephant at (A, 3).
- **SRS-FR10**: System ***shall*** place lower lion at (G, 1).
- **SRS-FR11**: System ***shall*** place lower tiger at (A, 1).
- **SRS-FR12**: System ***shall*** place lower leopard at (E, 3).
- **SRS-FR13**: System ***shall*** place lower wolf at (C, 3).
- **SRS-FR14**: System ***shall*** place lower dog at (F, 2).
- **SRS-FR15**: System ***shall*** place lower cat at (B, 2).
- **SRS-FR16**: System ***shall*** place lower rat at (G, 3).
- **SRS-FR17**: System ***shall*** place upper elephant at (G, 7).
- **SRS-FR18**: System ***shall*** place upper lion at (A, 9).
- **SRS-FR19**: System ***shall*** place upper tiger at (G, 9).
- **SRS-FR20**: System ***shall*** place upper leopard at (C, 7).
- **SRS-FR21**: System ***shall*** place upper wolf at (E, 7).
- **SRS-FR22**: System ***shall*** place upper dog at (B, 8).
- **SRS-FR23**: System ***shall*** place upper cat at (F, 8).
- **SRS-FR24**: System ***shall*** place upper rat at (A, 7).
- **SRS-FR25**: System ***shall*** rank elephant 8, lion7, tiger 6, leopard 5, wolf 4, dog 3, cat 2, rat 1.
- **SRS-FR26**: System ***shall*** accept piece names case-insensitively.
- **SRS-FR27**: System ***shall*** accept direction characters U/D/L/R case-insensitively.
- **SRS-FR28**: System ***shall*** compute target by applying direction to current position.
- **SRS-FR29**: System ***shall*** increment row by 1 for U direction.
- **SRS-FR30**: System ***shall*** decrement row by 1 for D direction.
- **SRS-FR31**: System ***shall*** decrement column by 1 for L direction.
- **SRS-FR32**: System ***shall*** increment column by 1 for R direction.
- **SRS-FR33**: System ***shall*** reject moves outside columns A through G.
- **SRS-FR34**: System ***shall*** reject moves outside rows 1 through 9.
- **SRS-FR35**: System ***shall*** reject moves of captured pieces.
- **SRS-FR36**: System ***shall*** reject moves of opponent's pieces.
- **SRS-FR37**: System ***shall*** reject moves into current player's own den.
- **SRS-FR38**: System ***should*** display "Your move out of board" for boundary violations.
- **SRS-FR39**: System ***should*** display "Cannot move captured piece" for dead pieces.
- **SRS-FR40**: System ***should*** display "Cannot move in your den" for own-den entry.
- **SRS-FR41**: System ***shall*** accept move to empty land square.
- **SRS-FR42**: System ***shall*** accept move to empty trap square.

- **SRS-FR43**: System ***shall*** accept move to empty opponent den.
- **SRS-FR44**: System ***shall*** reject move to square with friendly piece.
- **SRS-FR45**: System ***shall*** validate capture rules for occupied land squares.
- **SRS-FR46**: System ***shall*** validate capture rules for occupied trap squares.
- **SRS-FR47**: System ***shall*** validate capture rules for occupied den squares.
- **SRS-FR48**: System ***should*** display “Target square has your piece” for friendly collision.
- **SRS-FR49**: System ***should*** display “Cannot capture target” for illegal capture.
- **SRS-FR50**: System ***shall*** reject non-rat non-lion non-tiger moves to water.
- **SRS-FR51**: System ***shall*** accept rat move to empty water square.
- **SRS-FR52**: System ***shall*** validate capture for rat move to occupied water.
- **SRS-FR53**: System ***shall*** scan intervening water for lion leap attempt.
- **SRS-FR54**: System ***shall*** scan intervening water for tiger leap attempt.
- **SRS-FR55**: System ***shall*** reject leap if water square contains any piece.
- **SRS-FR56**: System ***shall*** accept leap to empty landing square.
- **SRS-FR57**: System ***shall*** validate capture for leap to occupied landing square.
- **SRS-FR58**: System ***should*** display “Cannot move piece to river” for invalid water.
- **SRS-FR59**: System ***should*** display “Cannot cross with piece blocking” for blocked leap.
- **SRS-FR60**: System ***shall*** permit capture if attacker rank >= defender rank.
- **SRS-FR61**: System ***shall*** permit rat capture of elephant on land.
- **SRS-FR62**: System ***shall*** prohibit elephant capture of rat under any circumstances.
- **SRS-FR63**: System ***shall*** permit any capture in attacker’s trap square.
- **SRS-FR64**: System ***shall*** prohibit water rat capture of land rat.
- **SRS-FR65**: System ***shall*** prohibit land rat capture of water rat.
- **SRS-FR66**: System ***shall*** permit water rat capture of water rat.
- **SRS-FR67**: System ***shall*** permit land rat capture of land rat.
- **SRS-FR68**: System ***shall*** prohibit capture between different terrain types.
- **SRS-FR69**: System ***shall*** remove piece from original square on valid move.
- **SRS-FR70**: System ***shall*** place piece on target square on valid move.
- **SRS-FR71**: System ***shall*** update piece position reference on valid move.
- **SRS-FR72**: System ***shall*** mark captured piece dead before move execution.
- **SRS-FR73**: System ***shall*** record captured event before move execution.
- **SRS-FR74**: System ***shall*** record move event after move execution.
- **SRS-FR75**: System ***shall*** toggle turn indicator after successful move.
- **SRS-FR76**: System ***shall*** trigger board display after successful move.
- **SRS-FR77**: System ***should*** print move confirmation after execution.
- **SRS-FR78**: System ***shall*** initialize turn to lower player.
- **SRS-FR79**: System ***shall*** alternate turns between lower and upper players.
- **SRS-FR80**: System ***shall*** not change turn on rejected move.
- **SRS-FR81**: System ***shall*** not change turn after successful withdraw.
- **SRS-FR82**: System ***shall*** check opponent den occupation before each turn.
- **SRS-FR83**: System ***shall*** check current player has legal moves before turn.
- **SRS-FR84**: System ***shall*** declare current player winner if opponent den occupied.
- **SRS-FR85**: System ***shall*** declare opponent winner if no legal moves exist.
- **SRS-FR86**: System ***shall*** mark game inactive upon victory detection.
- **SRS-FR87**: System ***shall*** save replay automatically upon victory.

- **SRS-FR88**: System ***should*** display victory message with winner name.
- **SRS-FR89**: System ***shall*** initialize each player withdraw quota to 3.
- **SRS-FR90**: System ***shall*** check withdraw quota before allowing withdrawal.
- **SRS-FR91**: System ***shall*** reject withdraw if quota is zero.
- **SRS-FR92**: System ***shall*** reverse last two move events on valid withdraw.
- **SRS-FR93**: System ***shall*** restore captured pieces during withdraw reversal.
- **SRS-FR94**: System ***shall*** decrement quota after successful withdraw.
- **SRS-FR95**: System ***shall*** record withdraw event to logger.
- **SRS-FR96**: System ***shall*** not toggle turn after successful withdraw.
- **SRS-FR97**: System ***shall*** reject withdraw if fewer than two moves exist.
- **SRS-FR98**: System ***should*** display "No quota remaining" when quota exhausted.
- **SRS-FR99**: System ***should*** display remaining quota after successful withdraw.
- **SRS-FR100**: System ***shall*** serialize entire game object on "saveGame" command.
- **SRS-FR101**: System ***shall*** generate timestamp filename if none provided.
- **SRS-FR102**: System ***shall*** create "./archive" directory if missing.
- **SRS-FR103**: System ***shall*** catch IOException during file write.
- **SRS-FR104**: System ***shall*** catch serialization errors gracefully.
- **SRS-FR105**: System ***should*** display "Save successful" on completion.
- **SRS-FR106**: System ***should*** display error details on save failure.
- **SRS-FR107**: System ***shall*** scan "./archive" for ".jungle" files.
- **SRS-FR108**: System ***shall*** display numbered list of available saves.
- **SRS-FR109**: System ***shall*** accept numeric selection from user.
- **SRS-FR110**: System ***shall*** deserialize game object from selected file.
- **SRS-FR111**: System ***shall*** return deserialized game to main loop.
- **SRS-FR112**: System ***shall*** catch IOException during file read.
- **SRS-FR113**: System ***shall*** catch ClassNotFoundException during deserialization.
- **SRS-FR114**: System ***should*** display "Game loaded successfully" on completion.
- **SRS-FR115**: System ***should*** display "File format invalid" for class errors.
- **SRS-FR116**: System ***should*** reprompt on load failure.
- **SRS-FR117**: System ***shall*** construct replay object with board and names.
- **SRS-FR118**: System ***shall*** include complete logger history in replay.
- **SRS-FR119**: System ***shall*** apply same validation as save game.
- **SRS-FR120**: System ***shall*** apply same error handling as save game.
- **SRS-FR121**: System ***shall*** auto-save replay with timestamp on game end.
- **SRS-FR122**: System ***shall*** scan "./replay" for ".replay" files.
- **SRS-FR123**: System ***shall*** display numbered list of available replays.
- **SRS-FR124**: System ***shall*** accept numeric selection from user.
- **SRS-FR125**: System ***shall*** deserialize replay object from selected file.
- **SRS-FR126**: System ***shall*** invoke startReplay method on loaded object.
- **SRS-FR127**: System ***shall*** apply same error handling as load game.
- **SRS-FR128**: System ***shall*** invoke logger initReplay on entering replay mode.
- **SRS-FR129**: System ***shall*** reset board to initial state for replay.
- **SRS-FR130**: System ***shall*** accept "next" command in replay mode.
- **SRS-FR131**: System ***shall*** advance by one turn on "next".
- **SRS-FR132**: System ***shall*** accept "prev" command in replay mode.

- **SRS-FR133**: System ***shall*** reverse by one turn on “prev”.
- **SRS-FR134**: System ***shall*** accept “exit” command in replay mode.
- **SRS-FR135**: System ***shall*** return to main menu on “exit”.
- **SRS-FR136**: System ***shall*** throw exception if “next” at end.
- **SRS-FR137**: System ***shall*** throw exception if “prev” at start.
- **SRS-FR138**: System ***shall*** handle withdraw events during replay navigation.
- **SRS-FR139**: System ***should*** display event message during replay step.
- **SRS-FR140**: System ***shall*** display welcome message on application launch.
- **SRS-FR141**: System ***shall*** present four main menu options.
- **SRS-FR142**: System ***shall*** accept “startNewGame” from main menu.
- **SRS-FR143**: System ***shall*** accept “loadGame” from main menu.
- **SRS-FR144**: System ***shall*** accept “watchReplay” from main menu.
- **SRS-FR145**: System ***shall*** accept “exit” from main menu.
- **SRS-FR146**: System ***shall*** terminate application on “exit” command.
- **SRS-FR147**: System ***shall*** return to menu after game completion.
- **SRS-FR148**: System ***should*** display “Invalid command” for unrecognized menu input.
- **SRS-FR149**: System ***shall*** trim whitespace from all input.
- **SRS-FR150**: System ***shall*** split input on whitespace boundaries.
- **SRS-FR151**: System ***shall*** ignore empty input lines.
- **SRS-FR152**: System ***shall*** validate argument count for each command.
- **SRS-FR153**: System ***shall*** catch IllegalArgumentException from model layer.
- **SRS-FR154**: System ***shall*** display exception message to user.
- **SRS-FR155**: System ***shall*** reprompt after displaying error.
- **SRS-FR156**: System ***should*** display “Too many arguments” for excess args.
- **SRS-FR157**: System ***shall*** render board using Unicode box-drawing characters.
- **SRS-FR158**: System ***shall*** use │ ─ ├┤ ┼ for grid.
- **SRS-FR159**: System ***shall*** display column labels A-G at top and bottom.
- **SRS-FR160**: System ***shall*** display row labels 1-9 on left and right.
- **SRS-FR161**: System ***shall*** use lowercase for lower player pieces.
- **SRS-FR162**: System ***shall*** use uppercase for upper player pieces.
- **SRS-FR163**: System ***shall*** represent pieces: r/R, c/C, d/D, w/W, p/P, t/T, i/I, e/E.
- **SRS-FR164**: System ***shall*** represent river with ≈ symbol.
- **SRS-FR165**: System ***shall*** represent trap with ■ symbol.
- **SRS-FR166**: System ***shall*** represent den with ▲ symbol.
- **SRS-FR167**: System ***shall*** combine piece and terrain symbols in cell.
- **SRS-FR168**: System ***should*** include legend on first board display.

7.2. Non-Functional Requirements (SRS-NFR)

- **SRS-NFR1**: ystem ***shall*** validate move within 500 milliseconds.
- **SRS-NFR2**: System ***shall*** execute valid move within 500 milliseconds.
- **SRS-NFR3**: System ***shall*** load save file under 100KB within 2 seconds.
- **SRS-NFR4**: System ***shall*** render board display within 200 milliseconds.
- **SRS-NFR5**: System ***shall*** never enter inconsistent game state.
- **SRS-NFR6**: System ***shall*** preserve in-memory state during file errors.
- **SRS-NFR7**: System ***shall*** handle missing directories gracefully.

- **SRS-NFR8**: System ***shall*** handle permission errors gracefully.
- **SRS-NFR9**: System ***shall*** handle disk full errors gracefully.
- **SRS-NFR10**: System ***shall*** validate deserialized object types.
- **SRS-NFR11**: System ***should*** recover from IOException without crash.
- **SRS-NFR12**: System ***should*** recover from ClassNotFoundException without crash.
- **SRS-NFR13**: System ***should*** enable gameplay after 5 minutes manual reading.
- **SRS-NFR14**: System ***should*** provide specific reason for each move rejection.
- **SRS-NFR15**: System ***should*** fit board display in 80x24 terminal.
- **SRS-NFR16**: System ***should*** avoid line wrapping in standard terminal.
- **SRS-NFR17**: System ***shall*** isolate model package from view package.
- **SRS-NFR18**: System ***shall*** isolate model package from controller package.
- **SRS-NFR19**: System ***shall*** centralize movement rules in MovingValidator class.
- **SRS-NFR20**: System ***shall*** centralize capture rules in MovingValidator class.
- **SRS-NFR21**: System ***should*** include Javadoc for all public methods.
- **SRS-NFR22**: System ***should*** document parameters, returns, and exceptions in Javadoc.
- **SRS-NFR23**: System ***shall*** execute on Windows with JDK 11+.
- **SRS-NFR24**: System ***shall*** execute on macOS with JDK 11+.
- **SRS-NFR25**: System ***shall*** execute on Linux with JDK 11+.
- **SRS-NFR26**: System ***shall*** use only Java standard library classes.
- **SRS-NFR27**: System ***shall*** avoid third-party framework dependencies.
- **SRS-NFR28**: System ***shall*** use UTF-8 compatible Unicode in display.
- **SRS-NFR29**: System ***should*** render correctly in common terminal emulators.
- **SRS-NFR30**: System ***shall*** expose state getters for unit testing.
- **SRS-NFR31**: System ***shall*** throw IllegalArgumentException for rule violations.
- **SRS-NFR32**: System ***shall*** include descriptive messages in exceptions.
- **SRS-NFR33**: System ***shall*** provide Scanner injection mechanism for testing.
- **SRS-NFR34**: System ***should*** enable white-box testing of model classes.
- **SRS-NFR35**: System ***should*** enable exception-based test assertions.
- **SRS-NFR36**: System ***shall*** handle event list of 500+ moves.
- **SRS-NFR37**: System ***shall*** support 100+ save files without degradation.
- **SRS-NFR38**: System ***shall*** support 100+ replay files without degradation.
- **SRS-NFR39**: System ***should*** maintain $O(1)$ piece lookup performance.
- **SRS-NFR40**: System ***should*** maintain $O(1)$ move validation performance.