Requirements Coverage Report
Jungle Game Implementation

COMP3211 Software Engineering Project
Group: Sun Hansong, Fu Guanhe, Wang Yuqi

1. User Stories Coverage

Table 1: User Stories Implementation Status

ID	User Story	Status	Implementation Notes
US1	As a player, I want to start a new game.	✓ Implemented	Command: `startNewGame` in Main.java (line 28-30). Creates new Game instance via `Game.newGame()` factory method.
US2	As a player, I want to end an ongoing game.	✓ Implemented	Command: `close` in Game.java (line 253-254). Calls `closeGame()` method which sets `isActive=false`, causing game loop to exit.
US3	As a player, I want to name the players with input or randomly generated strings.	✓ Implemented	Prompts for names in `Game.newGame()` (lines 304-309). If empty input, Player constructor generates 5-character random name via `Utils.getRandomString()`.
US4	As a player, I want to play via command line console.	✓ Implemented	All interactions via CLI. Scanner reads from System.in, output to System.out. No GUI components.
US5	As a player, I want to see game status with pieces, positions, and next player.	✓ Implemented	`Display.displayBoard()` shows full board state with Unicode box-drawing. Pieces shown as r/R, c/C, etc. Player prompt shows "upperPlayer (Name) >" or "lowerPlayer (Name) >".
US6	As a player, I want to take back at most 3 moves.	✓ Implemented	Command: `withdraw` in Game.java (lines 188-212). Player class maintains `withdrawQuota=3`, decremented on each use. Board.tryWithdraw() reverses last 2 move events.
US7	As a player, I want to record game info to file with extension ".record".	✓ Implemented	Command: `saveReplay` in Game.java (lines 234-250).
US8	As a player, I want to replay game from ".record" file.	✓ Implemented	Command: `watchReplay` in Main.java (line 36-37). LoadReplay.java loads replays and Replay.java provides next/prev/exit commands.
US9	As a player, I want to save game to ".jungle" file.	✓ Implemented	Command: `saveGame [filename]` in Game.java (lines 215-231). Serializes entire Game object to `./archive/<filename>.jungle`. Auto-generates timestamp filename if none provided.
US10	As a player, I want to load game from ".jungle" file.	✓ Implemented	Command: `loadGame` in Main.java (lines 31-35). LoadGame.java scans `./archive/` for `.jungle` files, displays numbered list, deserializes selected file.

2. System Requirements Specification - Functional Requirements

The SRS document defines 336 functional requirements (SRS-FR1 through SRS-FR336). Due to the large number of requirements, here we organize them into functional categories for clarity.

2.1. Board Initialization (SRS-FR1 to SRS-FR24)

Table 2: Board Initialization Requirements

Req ID	Requirement Summary	Status
SRS-FR1-8	Board structure: 7x9 grid, den positions (D,1) and (D,9), trap positions at (C,1), (D,2), (E,1), (C,9), (D,8), (E,9), river squares at B-C and E-F rows 4-6	✓
SRS-FR9-24	Initial piece placement: All 16 pieces (8 per player) placed at correct starting positions per traditional Jungle game rules	✓

Implementation Location: `Board.setDefaultBoard()` (Board.java, lines 24-84)

2.2. Piece Attributes and Movement Commands (SRS-FR25 to SRS-FR31)

Table 3: Piece and Command Requirements

Req ID	Requirement Summary	Status
SRS-FR25	Piece ranking: Elephant=8, Lion=7, Tiger=6, Leopard=5, Wolf=4, Dog=3, Cat=2, Rat=1	✓
SRS-FR26-31	Accept case-insensitive piece names and directions (U/D/L/R), compute target coordinates by applying direction offsets to current position	✓

Implementation Location:

- Piece ranking: `PieceType` enum ordinal values (PieceType.java, lines 3-11)
- Case-insensitive input: `equalsIgnoreCase()` in Utils.java and Game.java
- Direction computation: `Coordinate.moveByChar()` (Coordinate.java)

2.3. Move Validation (SRS-FR32 to SRS-FR82)

Table 4: Move Validation Requirements

Req ID	Requirement Summary	Status
SRS-FR32-38	Boundary validation: Reject moves outside columns A-G and rows 1-9, reject captured pieces, reject opponent's pieces, reject own den entry	✓
SRS-FR39-46	Square occupation rules: Accept moves to empty land/trap/den, reject friendly piece collision, validate capture rules	✓
SRS-FR47-62	Water square rules: Reject non-rat/lion/tiger to water, allow rat swimming, validate lion/tiger leap with blocking detection	✓
SRS-FR63-82	Capture rules: Rank-based capture, rat captures elephant, elephant cannot capture rat, trap allows any capture, prohibit water-land capture	✓

Implementation Location: `MovingValidator.attemptMove()` and `attemptCapture()` (MovingValidator.java, lines 86-196, 44-71)

2.4. Move Execution (SRS-FR83 to SRS-FR90)

Table 5: Move Execution Requirements

Req ID	Requirement Summary	Status
SRS-FR83-90	Move execution: Remove piece from original square, place on target, update position reference, mark captured piece dead, record events, toggle turn, trigger display	✓

Implementation Location: `Board.attemptMove()` (Board.java, lines 109-133)

2.5. Game Flow and Victory Conditions (SRS-FR91 to SRS-FR102)

Table 6: Game Flow Requirements

Req ID	Requirement Summary	Status
SRS-FR91-96	Turn management: Initialize to lower player, alternate turns, don't change on invalid move or withdraw	✓
SRS-FR97-102	Victory detection: Check den occupation, check no legal moves, declare winner, mark game inactive, auto-save replay	✓

Implementation Location:

- Turn management: `Game.startGame()` loop (Game.java, lines 139-261)
- Victory: `Board.checkDen()` and `Board.haveValidMove()` (Board.java, lines 143-152), `Game.gameOver()` (Game.java, lines 284-289)

2.6. Withdraw Mechanism (SRS-FR103 to SRS-FR112)

Table 7: Withdraw Requirements

Req ID	Requirement Summary	Status
SRS-FR103-112	Withdraw system: Initialize quota to 3 per player, check quota before allowing, reverse last 2 move events, restore captured pieces, decrement quota, maintain turn	✓

Implementation Location:

- Quota management: `Player` class (Player.java, lines 7-32)
- Withdraw execution: `Logger.tryWithdraw()` (Logger.java, lines 48-70)
- Game integration: Game.java lines 188-212

2.7. Save and Load Game (SRS-FR113 to SRS-FR130)

Table 8: Save/Load Game Requirements

Req ID	Requirement Summary	Status
SRS-FR113-120	Save game: Serialize game object, generate timestamp filename if needed, create ./ archive directory, catch IOExceptions and serialization errors	✓
SRS-FR121-130	Load game: Scan ./archive for .jungle files, display numbered list, accept numeric selection, deserialize game, handle errors gracefully	✓

Implementation Location:

- Save: `Game.saveGame()` (Game.java, lines 54-74)
- Load: `LoadGame.mainLoop()` (LoadGame.java, lines 35-86)

2.8. Replay System (SRS-FR131 to SRS-FR152)

Table 9: Replay System Requirements

Req ID	Requirement Summary	Status
SRS-FR131-140	Save replay: Construct replay with board and names, include logger history, auto-save on game end	✓
SRS-FR141-152	Load and play replay: Scan ./replay directory, display list, accept selection, deserialize, invoke startReplay, accept next/prev/exit commands, handle replay navigation	✓

Implementation Location:

- Save replay: `Game.saveReplay()` (Game.java, lines 96-117)
- Load replay: `LoadReplay.mainLoop()` (LoadReplay.java, lines 35-88)

- Replay playback: `Replay.startReplay()` (Replay.java, lines 25-68)
- Navigation: `Logger.nextStep()` and `previousStep()` (Logger.java, lines 82-154)

2.9. Menu and Input Handling (SRS-FR153 to SRS-FR168)

Table 10: Menu Requirements

Req ID	Requirement Summary	Status
SRS-FR153-168	Main menu: Display welcome, present 4 options (startNewGame, loadGame, watchReplay, exit), accept commands, handle invalid input, return to menu after game	✓

Implementation Location: `Main.main()` (Main.java, lines 11-44)

2.10. Board Display (SRS-FR169 to SRS-FR181)

Table 11: Display Requirements

Req ID	Requirement Summary	Status
SRS-FR169-181	Display formatting: Use Unicode box-drawing (│ ─ ├ ┤ ┼), show column labels A-G, row labels 1-9, lowercase for lower player, uppercase for upper player, piece symbols (r/R, c/C, d/D, w/W, p/P, t/T, i/I, e/E), terrain symbols (≈ for river, ■ for trap, ▲ for den), include legend	✓

Implementation Location: `Display.displayBoard()` and `displayBoardWithLegend()` (Display.java, entire file)

3. System Requirements Specification - Non-Functional Requirements

Table 12: Non-Functional Requirements Coverage

Category	Requirements Summary	Status
Performance (SRS-NFR1-4)	Move validation < 500ms, move execution < 500ms, load save < 2s, board render < 200ms	✓
Reliability (SRS-NFR5-12)	Never inconsistent state, preserve state during errors, handle missing directories, permission errors, disk full, validate deserialized types, recover from IOException and ClassNotFoundException	✓
Usability (SRS-NFR13-16)	Enable gameplay after 5min manual reading, specific rejection reasons, fit 80x24 terminal, avoid line wrapping	✓
Maintainability (SRS-NFR17-22)	Isolate model from view/controller, centralize rules in MovingValidator, Javadoc for public methods with parameters/returns/exceptions	✓
Portability (SRS-NFR23-26)	Execute on Windows/macOS/Linux with JDK 11+, use only Java standard library, UTF-8 Unicode display, render in common terminals	✓
Testability (SRS-NFR27-31)	Expose state getters, throw IllegalArgumentException for violations, descriptive exception messages, Scanner injection for testing, enable white-box testing	✓
Scalability (SRS-NFR32-36)	Handle 500+ move events, 100+ save/replay files, O(1) piece lookup and move validation	✓

Verification Notes:

- Performance: Operations complete instantaneously in testing; well below specified thresholds
- Reliability: Comprehensive try-catch blocks throughout codebase (Game.java, LoadGame.java, LoadReplay.java)
- Usability: Clear error messages, intuitive commands, comprehensive legend display
- Maintainability: Clean MVC architecture, model package fully isolated (src/Model/)
- Portability: Uses only java.io, java.util, java.time standard libraries; tested on macOS
- Testability: Test package (Test/) includes unit tests for all major model components
- Scalability: HashMap-based piece lookup provides O(1) access; no performance degradation observed