Jungle Game Project

Dou Shou Qi Implementation

COMP3211 Software Engineering

Group96
Chow Kwan Ho, Alvin
Fan Ka Hei, Sam
SIU Yau Shing, Shiloh
Tsang Kai Yuen, Paul

Presentation Overview

- Game UI Design (Commands, Output, Error Handling)
- Overall System Architecture
- Key Lesson Learned: API Design

UI Design: User Commands

Movement Formats

- **Chess notation:** a0 b0
- **Coordinates:** 0,0 1,0
- **Parentheses:** (0,0) (1,0)
- **Verbose:** move from a0 to b0

Control Commands

- **undo** - Revert up to 3 moves
- **save/load** - .jungle files
- **record** - Move history
- **help/quit** - Assistance & exit

UI Design: Game Output

ASCII Board Display

- **Blue pieces:** Uppercase letters (E=Elephant, L=Lion, T=Tiger)
- **Red pieces:** Lowercase letters (e, l, t, r, c, d, w, p)
- **Terrain markers:** # (den), * (trap), ~ (water)

Game Status Information

- Current player turn with color indicator
- Move count tracking game progress
- Captured pieces display for both players
- Win condition messages

UI Design: Error Handling

Custom Exception Hierarchy

9 specific exception types:

InvalidMoveException

PieceNotFoundException

WrongPlayerException, etc

User-Friendly Messages

Clear feedback instead of raw exceptions

"Elephant cannot capture Rat"

Early Input Validation

CommandParser validates before game logic

Overall Design: MVC Architecture

Model

Pure game logic

- Game class
- Board class
- 8 Piece types
- Zero dependencies

View

Display layer

- GameView
- BoardRenderer
- ASCII output
- Message display

Controller

Coordination layer

- GameController
- CommandParser
- FileManager
- NameManager

Component Interaction Flow

- User types command (e.g., "a0 b0")
- CommandParser validates & converts to Position objects
- GameController calls Game.make_move()
- Game validates using Board & Piece methods
- Board state updates on valid move
- GameView displays updated board
- User sees new state & next turn

Lesson Learned: API Design Challenge

Initial Approach

Single method: is_valid_move(target) → boolean

Problems Encountered

- Poor error communication (no WHY)
- Limited reusability for different contexts
- Mixed responsibilities in one method

API Design: Our Solution

can_move_to(board, target)

Checks if piece can reach the square

can_capture(target_piece, board)

Validates rank-based capture rules

get_valid_moves(board)

Returns list of all legal moves

Benefits of Decomposed API

- **Single Responsibility:** Each method has one clear purpose (491 unit tests)
- **Composability:** Game.make_move() uses both methods in sequence
- **Extensibility:** Tiger/Lion override only can_move_to() for river jumping
- **Better Errors:** Know exactly which validation failed

Key Takeaways

- Flexible UI with multiple input formats
- Clear visual feedback & robust error handling
- Clean MVC architecture with tests
- Thoughtful API design enables maintainability

Thank You!

Questions?