Project Description: The Jungle Game

Department of Computing

September 10, 2018

1 Overview

Jungle is a traditional Chinese board game. In this project, you will work in groups on the design and implementation of a Jungle game to be played in the command line console.

2 The Jungle Game

The Jungle game is a traditional Chinese board game played between two players. A concise description of the game and its rules is available in Appendix A.

The game you develop is to be played in the command line console in the following way:

REQ01 [2 points] When the program is launched, a user should be able to choose between starting a new game and opening a saved game;

REQ02 [4 points] At the beginning of a new game, the two players `X` and `Y` should be prompted to input their names. Then the initial board should be printed and player `X` should be prompted to input a command;

REQ03 A command can be a `save` command, an `open` command, or a `move` command:

- [4 points] `save [filePath]`: To save the current game into file at `[filePath]`.
- [4 points] `open [filePath]`: To load a saved game from `[filePath]`. If the current game is not saved yet, prompt the player to save the current game first.
- [7 points] `move [fromPosition] [toPosition]`: To move the piece at `[fromPosition]` to `[toPosition]`. For example, `move C7 C3` means to move the piece at position `C7` to position `C3`. Refer to Figure 1 in Appendix A for the encoding of positions on the game board.

REQ04 [5 points] If a command is valid, in the sense that it can be executed successfully, the command is executed. An invalid command should not affect the game state.

Continuing the previous example, if the player making the move has no piece at location `C7` or the piece is not allowed to directly jump to `C3`, the move is invalid and will not change the state of the game.

REQ05 [7 points] After each valid move, the updated game board should be printed, and the game checks if a player has achieved the goal (Appendix A.5): If yes, the game is over and the program should exit after printing the name of the winning player; Otherwise, the current player's turn is terminated and the other player should be prompted to input the next command;

REQ06 [7 points] Upon an invalid command, an error message should be shown and the same player should be prompted to input another move. If the invalid command is a `move` command, the current game board should also be printed.

The requirements above have been left incomplete on purpose. You need to decide on the missing details when designing the program.

The game may also be extended with the following *bonus* features:

Bon01 The game should have a full-fleged GUI mode.

Bon02 Two players should be able to play the game with each other on different computers.

3 Project Group

This is a group project. Each group may have 3 to 4 members. Please team up on Blackboard/Groups before or on October 3, 2018. Afterwards, we will randomly team up the ones with no group.

Groups will be announced no later than October 10. Afterwards, groups become official, and whoever requests for changing the group needs to

1. Send an email to the instructor explaining the desired change,
2. Seek agreements from all the members in both his/her current group and the group that he/she wants to join, and
3. Inform the instructor in email of the agreements from the other members. The email should be cc-ed to all the relevant members. Upon receiving the second email, the instructor will make the corresponding change.

4 Code Inspection

The inspection tool[1] of IntelliJ IDEA checks your program to identify potential problems in the source code. We have prepared a set of rules to be used in grading (`COMP2021_PROJECT.xml` in the project material package). Import the rules into your IntelliJ IDEA IDE and check the source code of your game against the rules. Test code does not need to be checked.

The inspection tool is able to check for many potential problems, but only a few of them are considered errors by the provided rule set. 2 Points will be deducted for each *error* in your code reported by the tool.

5 Statement Coverage of Unit Tests

Statement coverage[2] is a measure used in software testing to report the percentage of statements that have been tested: the higher the coverage, the more reliable the test results tend to be.

You should design your game using the Model-View-Controller (MVC) pattern[3]. Put all Java classes for the model into package `hk.edu.polyu.comp.comp2021.jungle.model` (see the structure in the sample project) and write tests for the model classes.

During grading, we will use the coverage tool[4] of IntelliJ IDEA to collect statement coverage information on your tests for the `model` subpackage. Your will get 15 base points for a statement coverage between 95% and 100%, 14 base points for a coverage between 90% and 94%, and so on. You will get 0 base points for a statement coverage below 25%. The final points you get for statement coverage of unit tests will be calculated as your base points multiplied by the percentage of your requirement coverage. For example, if you only implement 60% of the requirements and you achieve 95% statement coverage in testing, then you will get 9 = 15 * 60% points for this part.

[1] https://www.jetbrains.com/help/idea/code-inspection.html
[2] http://en.wikipedia.org/wiki/Code_coverage
[3] https://en.wikipedia.org/wiki/Model-view-controller
[4] https://www.jetbrains.com/help/idea/code-coverage.html

6 Project Grading

You can earn at most 100 points in total in the project from the following components:

- Requirement coverage (in total 40 points, as listed in Section 2)
- Code quality (10 points for good object-oriented design and 10 for good code style, as reported by the code inspection tool)
- Statement coverage of unit tests (15 points)
- Presentation (10 points)
- Final report (15 points)
- Bonus points (14 points for Bon-1 and 6 points for Bon-2, in total 20 points)

  Note: Bonus points can be used to reach, but not exceed, 100 points.

In the project, each group member is expected to actively participate and make fair contributions to the result. In general, members of a group will receive the same grade for what their group produces at the end. *Individual grading*, however, could be arranged if any member thinks one grade for all is unfair and files a request to the instructor in writing (e.g., via email). In individual grading, the grade received by each group member will be based on his/her own contribution to the project, and each member will need to provide evidence for his/her individual contributions to facilitate the grading.

7 General Notes

- Java SE Development Kit Version 8u181 and IntelliJ IDEA Community Edition Version 2018.2.3 will be used in grading your project. Make sure you use the same versions of tools for your development.
- Your code should not rely on any library that is not part of the standard API of Java SE 1.8.
- The game should handle incorrect user inputs gracefully. For example, it should not crash upon invalid user inputs.
- The project material package also include three other files:
  - `SampleProject.zip`: Provides an example for the organization of the project. Feel free to build your game based on the sample project.
  - `IntelliJ IDEA Tutorial.pdf`: Briefly explains how certain tasks necessary for the project can be accomplished in IntelliJ IDEA.
  - `COMP2021_PROJECT.xml`: Contains the rules to be used in code inspection.

8 What to Hand in

Archive the code of your classes, the tests, and the final report in PDF into a ZIP file, and submit the ZIP file on Blackboard.

If you use other IDEs for your development, make sure all your classes and tests are put into an IntelliJ IDEA project that is ready to be tested and executed. 50 points will be deducted from projects not satisfying this requirement or with compilation errors!

Appendix A The Jungle Game

This section gives a concise introduction to the Jungle game based on the description at
https://en.wikipedia.org/wiki/Jungle_(board_game).

A.1 Board

A board of the Jungle game consists of seven columns and nine rows of squares (Figure 1). Pieces move on the square spaces as in international chess, not on the lines as in Chinese Chess. Pictures of eight animals and their names appear on each side of the board to indicate initial placement of the game pieces. After initial setup, these animal spaces have no special meaning in gameplay.

There are several special squares and areas on the Jungle board: The dens (Figure 2) are located in the center of the boundary rows of the board, and are labeled as such in Chinese. Traps (Figure 3) are located to each side and in front of the dens, and are also labeled in Chinese. Two water areas or rivers (Figure 4) are located in the center of the board: each comprises six squares in a 2×3 rectangle and is labeled with the Chinese characters for "river". There are single columns of ordinary land squares on the edges of the board, and down the middle between the rivers.

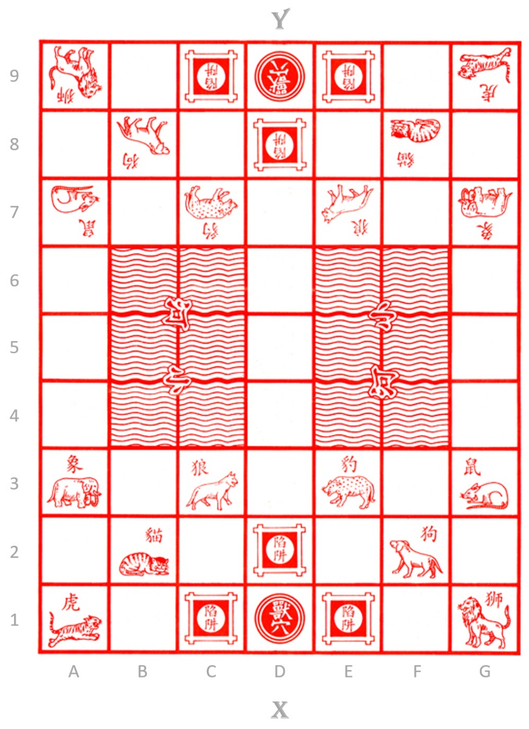

Figure 1: A typical Jungle gameboard showing the locations of starting squares, the den, rivers, and traps.

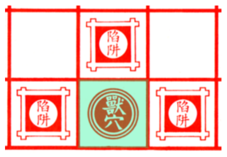

Figure 2: The den highlighted in green.

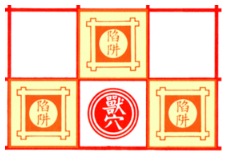

Figure 3: The traps highlighted in yellow.

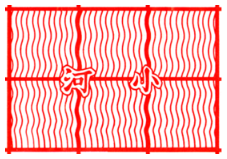

Figure 4: One of the rivers.

A.2 Pieces

Each side has eight pieces representing different animals, each with a different rank. Higher ranking pieces can capture all pieces of identical or lower ranking. However, there is one exception: The rat may capture the elephant, while the elephant may not capture the rat. The animal ranking, from highest to lowest, is as shown in Table 1. Pieces are placed onto the corresponding pictures of the animals which are invariably shown on the board.

Rank	Piece
8	Elephant
7	Lion
6	Tiger
5	Leopard
4	Wolf
3	Dog
2	Cat
1	Rat

Table 1: Pieces and their ranks.

A.3 Movement

Players alternate moves. During their turn, a player must move. Each piece moves one square horizontally or vertically (not diagonally). A piece may not move to its own den. There are special rules related to the water squares: The rat is the only animal that is allowed to go onto a water square. The rat may not capture the elephant or another rat on land directly from a water square. Similarly, a rat on land may not attack a rat in the water. The rat may attack the opponent rat if both pieces are in the water or on the land. The lion and tiger pieces may jump over a river by moving horizontally or vertically. They move from a square on one edge of the river to the next non-water square on the other side. Such a move is not allowed if there is a rat (whether friendly or enemy) on any of the intervening water squares. The lion and tiger are allowed to capture enemy pieces by such jumping moves.

A.4 Capturing

Animals capture the opponent pieces by "eating" them. A piece can capture any enemy piece which has the same or lower rank, with the following exceptions: A rat may capture an elephant (but not from a water square); A piece may capture any enemy piece in one of the player's trap squares regardless of rank.

A.5 Objective

The goal of the game is to move a piece onto the den on the opponent's side of the board, or capture all of the opponent's pieces.

Appendix B Project Report Template

Project Report

Group XYZ
COMP2021 Object-Oriented Programming (Fall 2018)
Name1
Name2
Name3

1 Introduction

This document describes the design and implementation of the Jungle game by group XYZ. The project is part of the course COMP2021 Object-Oriented Programming at PolyU. The following sections describe the requirements that were implemented and the design decisions taken. The last section describes the available commands in the game.

2 The Jungle Game

Give a short description of what the game is.

2.1 Requirements

Describe in this part the requirements (including the bonus ones, if any) you have implemented. Give a short description of each requirement. List the most important software elements (classes and/or methods) for supporting each requirement.

Req01	Description
	Software elements

2.2 Design

Use a class diagram to give an overview of the game design and describe in general terms how different components fit together. Feel free to elaborate on design patterns used (except for the MVC pattern) or anything else that might help others understand the design.

2.3 Quick Start Guide

Describe here all the commands that players can use.