**Project Report**

Group 19

COMP2021 Object-Oriented Programming (Fall 2018)

TENG Yun 16098399D

WEN Xi 1608574D

XU Suyan 16097128D

FU Xinyi 17083458D

**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**

The Jungle Game is a modern Chinese board game with an obscure history. The game is played on a 7×9 board, and The Jungle gameboard represents a jungle terrain with dens, traps "set" around dens, and rivers. Each player controls eight game pieces representing different animals of various rank. Stronger-ranked animals can capture ("eat") animals of weaker or equal rank. The player who is first to maneuver any one of their pieces into the opponent's den wins the game. An alternative way to win is to capture all the opponent's pieces.

In our game, we will simulate all the processes and rules of this tradition game. Also, we will add some more functions like save and load game to make it more like a computer games.

**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**

When the program is launched, the user could choose to start a new game or open a saved game.

**Software elements**

This is achieved by the chooseMenu() and the first half part of start() function.

The chooseMenu() function prompt a notification and ask the user to input one command between save game and load game. The function checks the validity: if not valid, it prompts a notification and let the user input until a valid command; if valid, it will check and return the type of the command, whether “START” or “LOAD”.

The start() function is the core function of TextView class. It switch the output of start() function: if “START”, it will ask the user to input two player name and initialize a new game; if “LOAD”, it will ask the user to input the file path and load the game.

**Req02**

**Description**

At the beginning of a new game, a notification will be prompted to let the user to input two player names. If the name is null, a error message will be printed and let the user to input again until a non-empty player name entered. After two players’ entering their names, the game will be initialized and the board will be printed. The first player, player X will be prompted to input a command.

**Software elements**

This is achieved by start() function in the TextView class. Inside start() function start game part, a while loop is used until the user input a non-empty player name. After the two user input their correct names, model.initializeGame(); will be user to initialize a game and pass the player name. A board will be printed by printBoard() function.

**Req03**

**Description:**

During the game, a notification will be prompted to let the user choose save command, open command, or a move command:.

save [filePath]: To save the current game into [filePath].

open [filePath]: To load a saved game from [filePath].

move [fromPosition] [toPosition]: To move the piece at [fromPosition] to [toPosition].

**Software elements:**

This requirement is achieved by. For the ‘save’ command, the saveGame() method of the TextView class with a parameter which is supposed to be the filepath, saveGame() method checks if there’s already a namesake file in the directory and decide whether to prompt the player to delete the original file or give the file a new name. The loadGame() method in the TextView class first check if the savestate variable in the JungleGame class is set to true, then examine if the file exist in the target directory, finally it calls the loadCheck() method to judge whether the file string is readable or not. For the ‘move’ command, the input string is supposed to have the length of ten, the haddleCommand() method in the TextView class divide the string into two parts as well as utilize them as the parameters of the movePosition() method.

**Req04**

**Description:**

1. Invalid command doesn’t affect the game state.

2. Invalid move position input will be detected and therefore rejected

**Software elements:**

1. movePosition() method examines the two string format position, if an invalid input is encountered, value false will be returned immediately. Therefore, game state will not be affected.

2. The requisition is achieved by the movePosition() method in the JungleGame class as well as the eat() method in the Board class. First and foremost, it checks if there’s a piece in the former square input and it belongs to the current player. Thereafter, it checks the latter square input, which is not supposed to be the current player’s own dens.

For the case that either a lion or tiger hopes to jumps across the river, movePosition() inspects that if the rank of the piece in the former square is higher than that in the latter (if any). Besides, a rat-piece is not expected to appear in the river overlapping the jumping route.

For the case that a rat hopes to walk across the river, if either the former or latter square is part of the river, when there’s a piece inside the latter square, move command is evaluated as invalid.

As for the general case, piece in the latter input square (if any) is expected to have a smaller rank than that inside the former.

After the all these requisition above are met, the eat() method is called in case there is any piece in the latter square. Finally, piece position, piece number remained and winning state are updated, player’s turn is switched, the value true is returned as well.

**Req05**

**Description:**

1. After each valid move, the updated game board will be printed

2. The game checks if a player has achieved the goal

3. The game is over, if a winner rises

4. The program will exit after printing the name of the winning player

5. The current player's turn is terminated and the other player should be prompted to input

next command, in case neither of the players win

**Software elements:**

1. This is implemented by the printBoard method in the View class, using notation “~” as river, “+” as trap, “.” as dens, “|” as the height of a single square and “-” as the length of a square.

2. If the current player wins after the movement, then either after the current move, player’s piece is in his counterpart’s dens, or the number of his counterpart’s piece equals to zero. This requisition is achieved through calling the board.isDens() and player.getPieceNum() method. The field winstate that is initialized as false will be updated to true in case someone wins.

3&4&5. Examination of current winning state is implemented by checking the winstate variable in the JungleGame class, if winstate is true, then model.iswin() is true, a while statement will keep prompting the players to input a valid command. Otherwise, the while statement will terminate and handle the remaining lines, then program ends after printing the current player’s name.

**Req06**

**Description:**

1. Error message should be printed and program keeps prompting player to input until encounters a valid command

2. Print the board in case of an invalid move command

**Bon01**

**Description:**

The game has a full-fledged GUI mode, which can be launch by uncommenting line 20 javafx.application.Application.*launch*(ApplicationUI.**class**);in Application.java

**Software elements:**

1. Application.java launches ApplicationUI.java under view folder, the later extends javafx.application.Application and thus is able to create a JavaFX GUI

2. ApplicationUI creates a new GameUI class, and sets the initial primary stage

3. GameUI class creates a new JungleGame class and 3 scenes, EntryScene, InfoScene, BoardScene, this class is also responsible to switching between scenes using the getPrimaryStage() method

4. EntryScene allows user to select between starting a new game and resuming an old game by clicking Buttons, the prior will then navigate to InfoScene, the later will generate a TextInputDialog that takes the file path as input

5. InfoScene will ask user for player names using TextField, a Button is used to start the game

6. BoardScene is the main scene for the game GUI, it uses Text to show welcome message, Button to select between save game, load game and exit. GridPane is used to create the 7\*9 board grid layout, each grid socket has a SquareUI, which has a StackPane private instance which allows PieceUI to be added to it or removed from it, each PieceUI has a Button private field, which allows the piece to be added and removed from a SquareUI

7. To select a Piece, chick once, to deselect it, click again, to move one piece to another square or on another piece, click piece first and then click the corresponding piece/square.

**Software elements:**

This requisition is fulfilled by calling the start() as well as the handleCommand() method in TextView class, the handleCommand() method examines whether the command is valid or not. As for an invalid move command, the printBoard() method is called after printing “Invalid move position.”. By the courtesy of the while loop in method start(), program will keep prompting players to input until it reads a valid command, while the error message “Invalid command. Please enter again.” will be printed.

**2.2 Design**

A screenshot of text

Description automatically generated

In this project, we use MVC design pattern: JungleGame (Board, Player, Piece) class is the model, View interface with TextView and GUI are the view, and Menu is the controller.

The controller is implemented relatively simple, launching the view and model and let them further interact with each other. View is an interface, implemented by TextView and GUI, doing the basic IO and command validation. JungleGame contains Board and Player, with general game information like nextTurn, winStatus, winnerID, and so on.

Board contains the basic information of board (COL\_NUM, ROL\_NUM, PIRECE\_NUM), piece location relationship storing in a HashMap data structure, and dens, river locations. The piece is the conceptually for the whole animals in Jungle Game, with basic information like position, rank and ownerID. Therefore, we could get piece location from board, and trace its owner by its owner ID. The player class contains Name, pieceNumber information.

So in the whole program, application calls controller, controller will calls view. View will prompt users necessary information and ask for command. It validates command and calls corresponding function in Jungle Game or Board. As Board, Player and Piece could be all traced by each other, we could get all information from one class and Jungle Game.

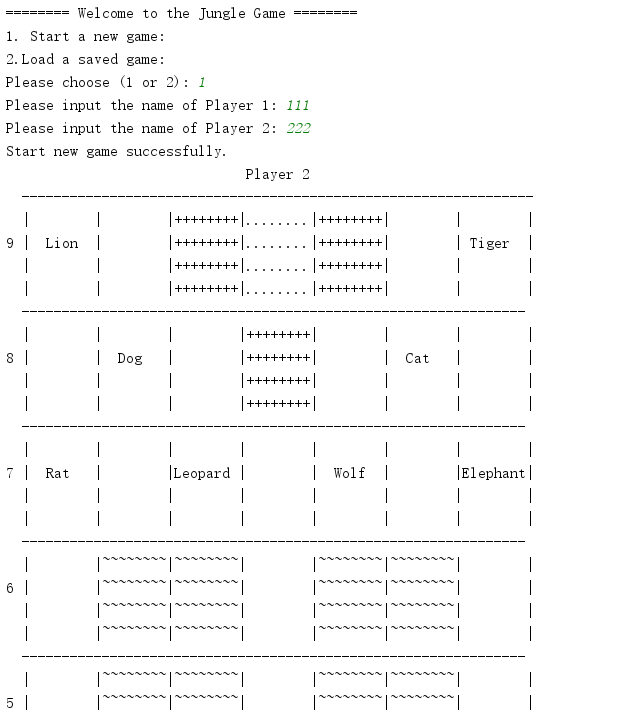
**2.3 Quick Start Guide**

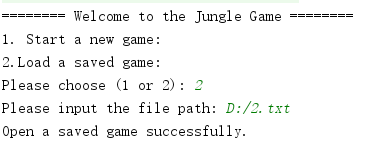
1. Base on Command Line Interface:

1. Run our game’s main function and close the GUI window.

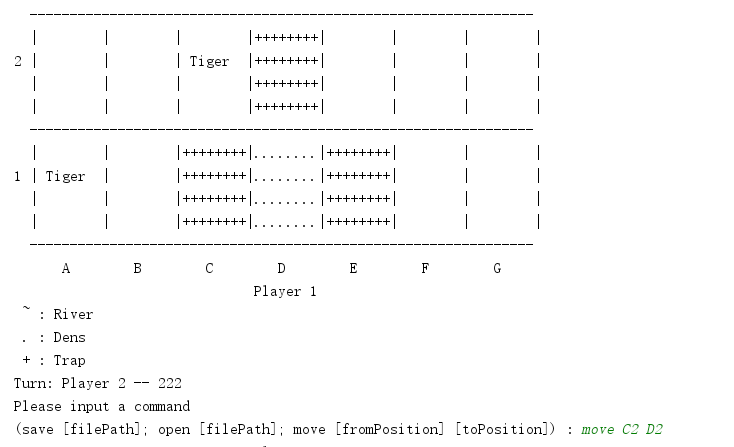
2. Then user can choose two options: start new game or load a saved game

3. If we choose start game and input two players name, a new game will start

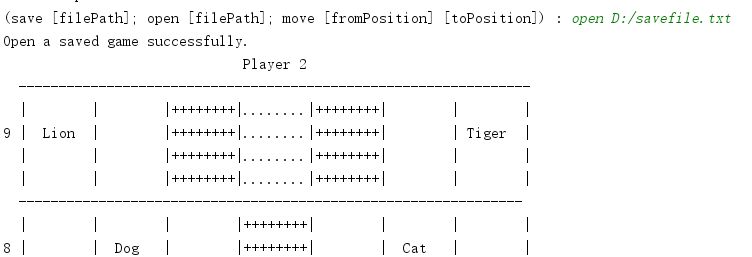


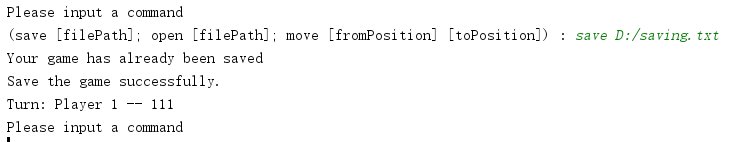
 4. If we choose load game and input the valid file path, the saved game will start:

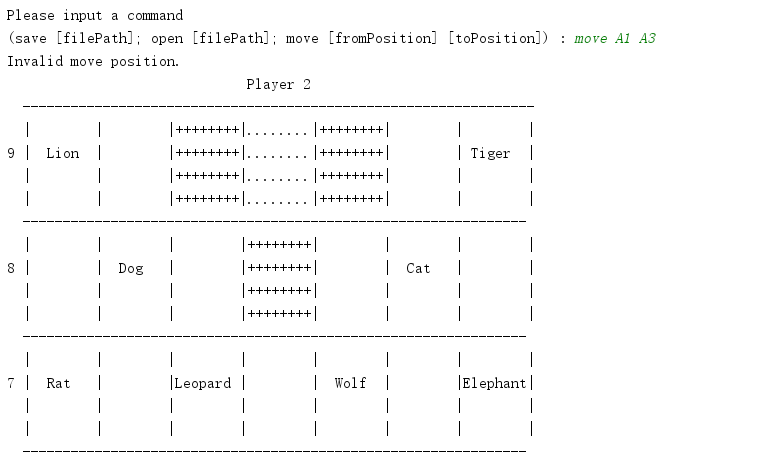
5. During the game, user can input move [fromPosition] [toPosition] to move pieces



6. During the game, user can input save [filePath] or open [filePath] tosave or load game





7. If user input any invalid command, system will push user to input again

2. Base on GUI:

1. All the command are similar. Using clicks instead of input command line. Below, we just show some snipping for the game and show how the interface looks like. GUI can be launch by uncommenting line 20 javafx.application.Application.*launch*(ApplicationUI.**class**);in Application.java

