



PROJECT REPORT

COMP2021 Object-Oriented Programming

(Fall 2019)

1. Introduction

This document describes the design and implementation of the Jungle game by group 36. 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 slightly similar to Chinese Chess. The game involves 2 players, playing against each other. Each player has a set of 8 distinct pieces. The game is played on a board of 7 x 9 squares. The objective is to take over as many opponent's pieces and land one piece on the opponent's den. Whichever player lands the first piece on the opponent's den or takes over all the opponent's pieces wins.

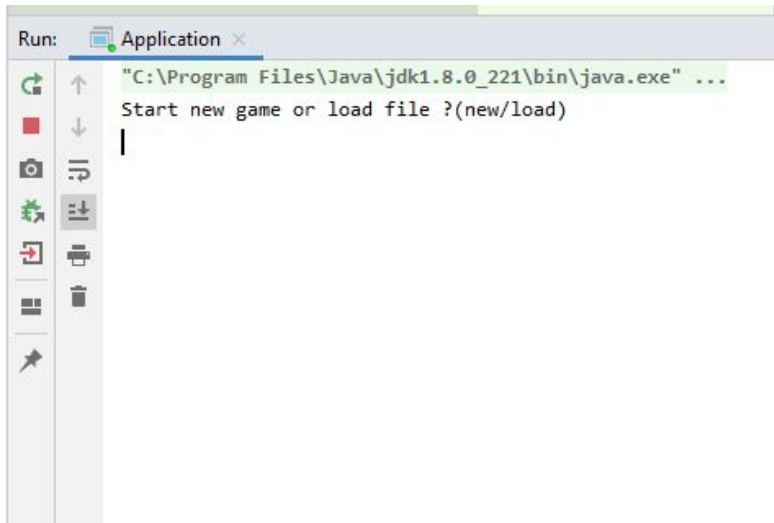
The players take turns moving their pieces. Each player can only move one piece at a time and only one box in any direction.

Each piece has a rank in order Elephant, Lion, Tiger, Leopard, Wolf, Dog, Cat, Rat with Elephant having the highest and the Rat having the lowest rank. A piece can take over a lower rank or the same rank piece, except Rat can take over Elephant.

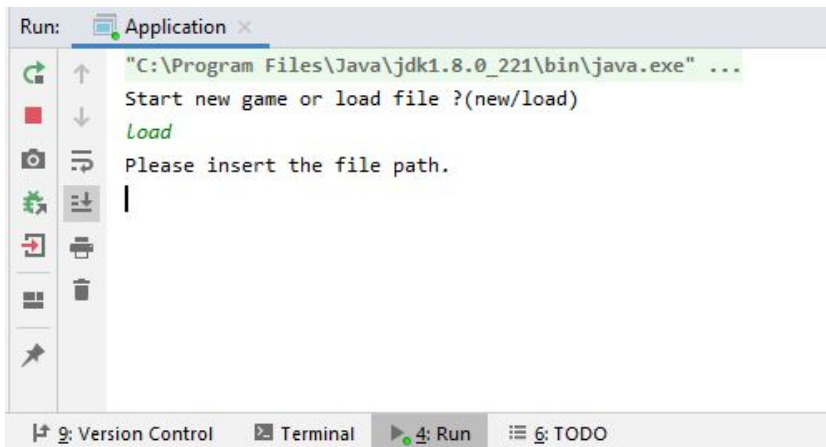
There are 2 different types of obstacles in the game, rivers and traps. The obstacles are to assist some pieces but to slow down others. Only a Rat can enter the river, but it can neither capture other piece nor can be captured by other pieces while in the river. While Lion and Tiger can jump both vertically and horizontally over the river squares unless there is no Rat in any of the river squares, and they can take over other pieces using this movement. Lastly, the trap can help capture any piece regardless of the rank. A player can simply take over a piece in its trap by moving to that square.

2.1 Requirements

REQ1. The requirement is to give the user a choice to choose between starting a new game or loading an already saved game. The user is prompted to input his/her choice. This is done in the Application class in the main program. The prompt is shown below.



If the user chooses new, a new board is created. If the user chooses load, a prompt is shown to ask for the path for the file (as shown below). This command uses the method `executeGame()` from Class `LoadCommand`.



REQ2. This requires the players to input their names. Once the user is asked to input their choice for the game they are prompted to input their names. Player x is prompted to input a name and then player y is prompted to input a name. The second part of this requirement is to print the board and to prompt player x to input a command, it could either be a move, save, or an open command.

```
Run: Application x
"C:\Program Files\Java\jdk1.8.0_221\bin\java.exe" ...
Start new game or load file ?(new/load)
new
Player X please input your name.
abc
Player Y please input your name.
xyz

      Y
    + - - - - - +
  9 | L   ^  #  ^   T |
  8 |   D   ^       C |
  7 | R   l   W   E   |
  6 |   ~ ~   ~ ~   |
  5 |   ~ ~   ~ ~   |
  4 |   ~ ~   ~ ~   |
  3 | E   W   l   R   |
  2 |   C   ^       D |
  1 | T   ^  #  ^   L |
    + - - - - - +
      A B C D E F G
      X
Player x: Please input command.
|
```

REQ3. This requirement is to create 3 different types of input commands: save, open, and move. The user is prompted to input a command. The choices are taken in the Application class. If the user chooses to save, a prompt is shown to input the file path for the destination of the file. This task calls the method execute() from class SaveCommand. If the user chooses to open. A prompt is shown to input the location of the file the user wants to open. This task calls the method executeGame() from class LoadCommand. The prompts are shown below.

```
Run: Application x
"C:\Program Files\Java\jdk1.8.0_221\bin\java.exe" ...
Start new game or load file ?(new/load)
new
Player X please input your name.
abc
Player Y please input your name.
xyz

      Y
+ - - - - - +
9 | L  ^ # ^  T |
8 |  D  ^    C  |
7 | R  I  W  E  |
6 | ~ ~ ~ ~ ~ |
5 | ~ ~ ~ ~ ~ |
4 | ~ ~ ~ ~ ~ |
3 | E  W  I  R  |
2 |  C  ^    D  |
1 | T  ^ # ^  L  |
+ - - - - - +
  A B C D E F G
      X

Player x: Please input command. (save / open/ move)
open
Do you want to save current game? (1/0)
0
Please insert the file path.
|
```

```
Run: Application x
"C:\Program Files\Java\jdk1.8.0_221\bin\java.exe" ...
Start new game or load file ?(new/load)
new
Player X please input your name.
abc
Player Y please input your name.
xyz

      Y
+ - - - - - +
9 | L  ^ # ^  T |
8 |  D  ^    C  |
7 | R  I  W  E  |
6 | ~ ~ ~ ~ ~ |
5 | ~ ~ ~ ~ ~ |
4 | ~ ~ ~ ~ ~ |
3 | E  W  I  R  |
2 |  C  ^    D  |
1 | T  ^ # ^  L  |
+ - - - - - +
  A B C D E F G
      X

Player x: Please input command. (save / open/ move)
save
Please select the file path.
|
```

If the user chooses move command, he/she will have to input the initial and final position of the piece after move. And if the command is valid, the command is executed and the new board is printed. This is done using execute() isValid() methods in MoveCommand. And the updated board is displayed after each valid command.

```
+ - - - - - +
9 | L  ^ # ^  T |
8 |  D  ^    C  |
7 | R  I  W  E  |
6 | ~ ~ ~ ~ ~ |
5 | ~ ~ ~ ~ ~ |
4 | ~ ~ ~ ~ ~ |
3 | E  W  I  R  |
2 |  C  ^    D  |
1 | T  ^ # ^  L  |
+ - - - - - +
  A B C D E F G
      X

Player x: Please input command. (save / open/ move)
move a1 a2

      Y
+ - - - - - +
9 | L  ^ # ^  T |
8 |  D  ^    C  |
7 | R  I  W  E  |
6 | ~ ~ ~ ~ ~ |
5 | ~ ~ ~ ~ ~ |
4 | ~ ~ ~ ~ ~ |
3 | E  W  I  R  |
2 | T  C  ^    D  |
1 |  ^ # ^  L  |
+ - - - - - +
  A B C D E F G
      X

Player y: Please input command. (save / open/ move)
|
```

REQ4. The player is required to input a command. The requirement is to check if the command is valid or not. The command should only be executed if it is valid, otherwise, it should not have any effect on the game. This requirement is met through execute() and isValid() methods in MoveCommand class. If the command is invalid the user is asked to input the command again.

```

+ - - - - - - - +
9 | L   ^ #   ^   T |
8 |   D   ^   C   |
7 | R   l   W   E   |
6 |   ~ ~   ~ ~   |
5 |   ~ ~   ~ ~   |
4 |   ~ ~   ~ ~   |
3 | E   W   l   R   |
2 | T C   ^   D   |
1 |   ^ #   ^   L   |
+ - - - - - - - +
  A B C D E F G
      X
Player y: Please input command. (save / open/ move)
move a3 a4
Wrong Site.
Player y: Please input command. (save / open/ move)
|

```

The command shown above is invalid because player y tried to move player x's piece, hence, is asked to input the command again.

REQ5. It requires the game board to be printed after each successful move and the new position of the pieces should be shown. This requirement is met by method printBoard() in class JungleGame. And the second requirement is to check if either of the players has won or not. For that, the method gameEnd is called from the same class.

REQ6. As required, after an invalid command, the same player will be prompted to input another command. And the current status of the board remains.

```

+ - - - - - - +
9 | L   ^ # ^   T |
8 |   D   ^   C   |
7 | R   l   W   E   |
6 |   ~ ~   ~ ~   |
5 |   ~ ~   ~ ~   |
4 |   ~ ~   ~ ~   |
3 | E   W   l   R   |
2 | T C   ^   D   |
1 |   ^ # ^   L   |
+ - - - - - - +
  A B C D E F G
      X

```

Player y: Please input command. (save / open/ move)

move a3 a4

Wrong Site.

Player y: Please input command. (save / open/ move)

|

2.2 Design

This Project uses a modular approach to satisfy all the requirements and to efficiently run the game. For the flexibility of the code design, the program adopted the Model view controller architecture and the command pattern. Developers are able to add any function more effectively. It comprises of 3 main packages: Command, Expection, and model. Each package contains several classes, each of which is responsible for different aspects of the program as explained below.



1. Model:

The model package is mainly responsible for the game. The board, pieces, their ranks, obstacles and the den are all constructed in this. It contains 7 classes as shown below.

1. Chess

Chess class contains the data for all the chess pieces like their names and ranks. The constructor is used to assign the name and rank. And a few methods are implemented to get rank and compare ranks. Each piece is identified using their rank but not the letter assigned as 2 pieces have the same letter.

2. Den:

This class is defined to check for the den. A method is implemented to check if a piece has reached the den.

3. Environment:

This is an abstract class, all the model in the game board will follow its structure. And only the type of environment object can save into the game board.

4. JungleGame:

This class is the main class for creating the game. This contains the methods for creating the board and initializing object fields such as player names. It implements methods to print the current status of the game board and to check after each move if a player has managed to reach the den.

5. Rank:

Rank is an enum class that is created to declare the ranks for each different type of piece and has a getter to pass the rank when required.

6. River:

This class is used to check for the location of the river on the board.

7. Trap

Similar to the River class this class checks for the location of the trap if a piece tries to move onto it.

2. Command:

This package contains the implementation for all the commands a user may input while playing the game, for example, move, load or open command. A separate class is implemented for each type of command.

1. LoadCommand:

This class uses a java API for object deserialization that makes creates an object from a byte stream. The already saved file is accessed from the path input by the user and a new object is loaded.

2. MoveCommand:

MoveCommand is one of the major classes in the project that implements the code for every move the user would like to make. This makes the game playable for the user. It checks for all the conditions for the command to be valid using a method isValid() and then execute() method is called in the main program to run the command. Conditional statements are used in both the methods to check for any possibility of change. For each command, the letter part is separated and converted to an integer index to be passed on for execution by the constructor.

3. SaveCommand:

This class uses the same method for object serialization as LoadCommand that makes a writable byte stream that can be saved in a file. This saves the game object in a file to be loaded for later use.

4. Command:

This is an abstract class, all the command classes will follow the structure of command class.

3. Expectation:

This package holds the exceptions that may be thrown during the execution of the program.

1. InputException:

This class will handle the input error exception of the program. When the user inputs a wrong command, it will create an exception object which contains an error message.

2. siteException:

This class will handle the incorrect site exception. It will be created when user select other's chess.

The main program is implemented in the Application class. This class contains the code for execution of the other classes. User prompts for input of commands are printed in this class and it calls command methods to execute the user commands.

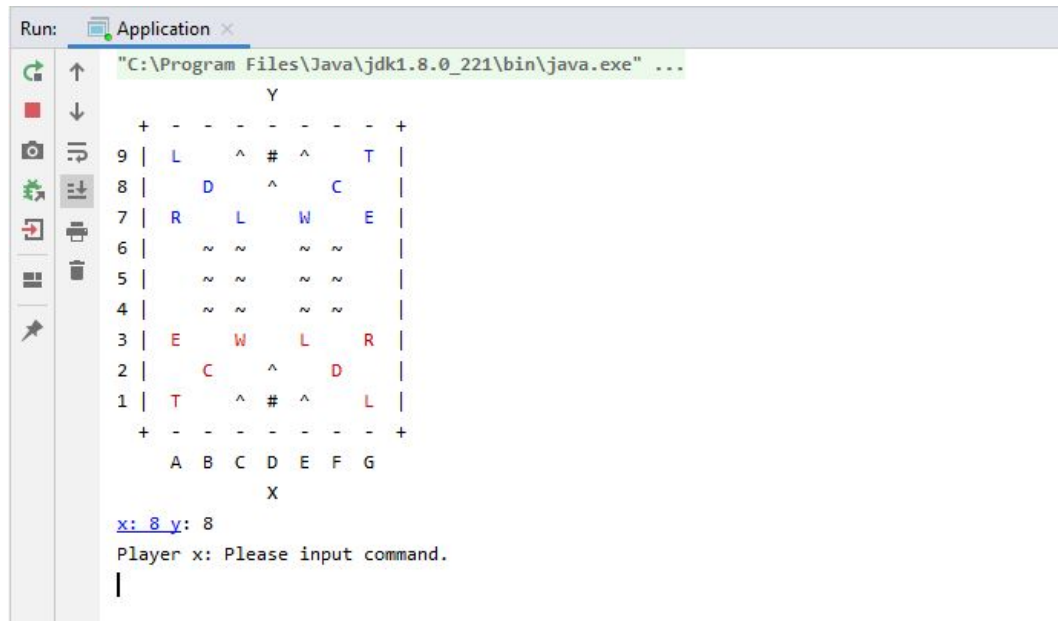
2.3 Quick Start Guide

After running the application, the user is prompted to enter a choice between starting a new game or loading an already saved game. If the user requests to load a saved game, a prompt is shown to input the file path. And the saved game is opened and the board is displayed. For a new game, the user is prompted to input each players name and the board is displayed.

After the current status of the board is displayed and Player x is prompted to input a command they would like to execute. There are 3 different types of commands a player can give: move, save, open. And players take alternate turns to move their pieces and the board is displayed after each valid command.

1. Move:

A move command is used to move a piece on the board as shown in the screenshot below.



```
Run: Application x
"C:\Program Files\Java\jdk1.8.0_221\bin\java.exe" ...

      Y
      + - - - - - +
9 | L   ^ # ^   T |
8 |   D   ^   C   |
7 | R   L   W   E   |
6 |   ~ ~   ~ ~   |
5 |   ~ ~   ~ ~   |
4 |   ~ ~   ~ ~   |
3 | E   W   L   R   |
2 |   C   ^   D   |
1 | T   ^ # ^   L   |
      + - - - - - +
      A B C D E F G

      X

x: 8 y: 8
Player x: Please input command.
|
```

Player name is mentioned at the beginning of the prompt, in this case, x. Player X can only move his/her own pieces.

If the player intends to move a piece, they have to enter the command in the following format:

move (initial location) (final location)

For instance player x intends to move Tiger vertically to A2.

The player will have to enter:

move a1 a2

The example is shown in the picture below.

```

Run: Application x
9 | L ^ # ^ T |
8 | D ^ C |
7 | R L W E |
6 | ~ ~ ~ ~ |
5 | ~ ~ ~ ~ |
4 | ~ ~ ~ ~ |
3 | E W L R |
2 | C ^ D |
1 | T ^ # ^ L |
+ - - - - - +
  A B C D E F G
X
x: 8 y: 8
Player x: Please input command.
move a1 a2
Y
+ - - - - - +
9 | L ^ # ^ T |
8 | D ^ C |
7 | R L W E |
6 | ~ ~ ~ ~ |
5 | ~ ~ ~ ~ |
4 | ~ ~ ~ ~ |
3 | E W L R |
2 | T C ^ D |
1 | ^ # ^ L |
+ - - - - - +
  A B C D E F G
X
x: 8 y: 8
Player y: Please input command.
|

```

The status of the board will be displayed after each command is executed. Then, the other player is asked for the input.

2. Save:

For a save command the user is prompted to input the destination file path. The user inputs the file destination and game is saved as shown below.

```

Run: Application x JungleGameTest x
Y
+ - - - - - +
9 | L ^ # ^ T |
8 | D ^ C |
7 | R l W E |
6 | ~ ~ ~ ~ |
5 | ~ ~ ~ ~ |
4 | ~ ~ ~ ~ |
3 | E W l R |
2 | T C ^ D |
1 | ^ # ^ L |
+ - - - - - +
  A B C D E F G
X
Player x: Please input command. (save / open/ move)
move a3 a4
Y
+ - - - - - +
9 | L ^ # ^ T |
8 | D ^ C |
7 | R l W E |
6 | ~ ~ ~ ~ |
5 | ~ ~ ~ ~ |
4 | E ~ ~ ~ |
3 | W l R |
2 | T C ^ D |
1 | ^ # ^ L |
+ - - - - - +
  A B C D E F G
X
Player y: Please input command. (save / open/ move)
save
Please select the file path.
C:\Users\haris\Downloads\myGame.txt
SAVE GAME.

```

3. Open:

Similar to the save command when the user gives the command to open, first the user is asked if they would like to save the game.

```

Run: Application x JungleGameTest x
Y
+ - - - - - +
9 | L ^ # ^ T |
8 | D ^ C |
7 | R l W E |
6 | ~ ~ ~ ~ |
5 | ~ ~ ~ ~ |
4 | ~ ~ ~ ~ |
3 | E W l R |
2 | C ^ D |
1 | T ^ # ^ L |
+ - - - - - +
  A B C D E F G
X
Player x: Please input command. (save / open/ move)
open
Do you want to save current game? (1/0)
0

```

If the users chooses 0, he/she is asked for a file path to be input to load the game from as shown below.

```

Player x: Please input command. (save / open/ move)
open
Do you want to save current game? (1/0)
0
Please insert the file path.
C:\Users\haris\Downloads\myGame.txt
OPEN GAME.

      Y
+ - - - - - +
9 | L ^ # ^ T |
8 | D ^ C |
7 | R l W E |
6 | ~ ~ ~ ~ |
5 | ~ ~ ~ ~ |
4 | E ~ ~ ~ ~ |
3 | W l R |
2 | T C ^ D |
1 | ^ # ^ L |
+ - - - - - +
  A B C D E F G
      X
Player y: Please input command. (save / open/ move)

```

The game ends when either of the user wins by taking over all the opponent's pieces or one of the piece reaches the opponent's den. The game can also be stopped by saving the game.