

Project Report
Group 10
COMP2021 Object-Oriented Programming (Fall 2018)

Cheung Ka Ho (17047931d)

Ngan Ting Cheuk (17061434d)

Yan Ho Wang (17057122d)

Yau Yuet Man (17056093d)

1 Introduction

This document describes the design and implementation of the Jungle game project by Group 10. 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 and functions in the game.

2 The Jungle Game

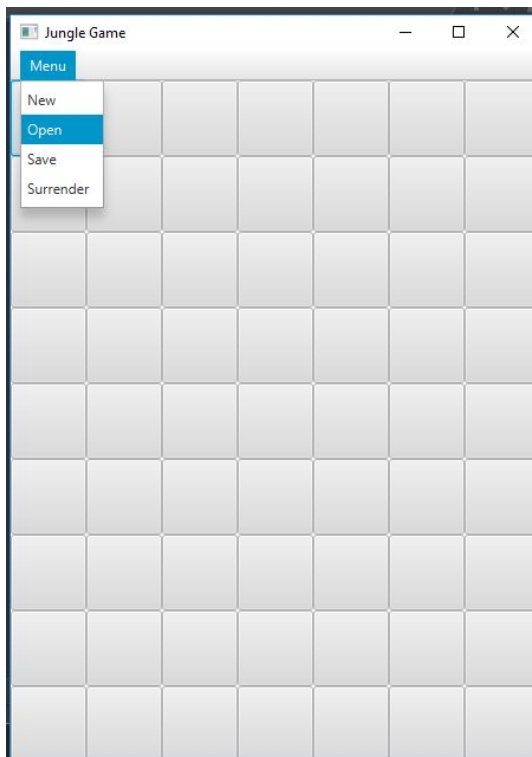
The Jungle Game is an Chinese traditional game with 7 kinds of animals in both side, such as elephant, lion, tiger, leopard, wolf, dog, cat and rat. Animals with higher ranks can eat other animals. For exception, rat with the lowest rank can capture the elephant with the highest rank, but elephant cannot.

A Player will win the game if his piece step on the enemies' den or his enemies' pieces are all eaten.

2.1) Requirements

REQ01:

When the program is launched, users should be able to choose between starting a new game and opening a saved game. The user can choose either open a new game or load a game save file from if there's any, from the menu.



Supporting software elements:

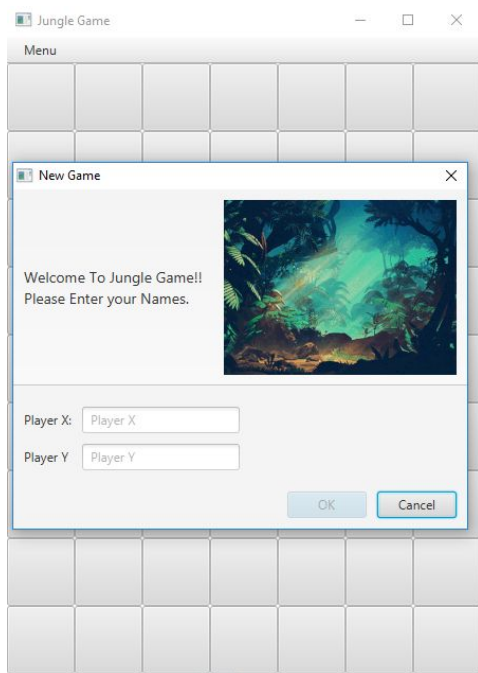
Class name: JungleGame, JungleGameGUI

Methods: StartNewGame(String nameX, String nameY) ,OpenSavedGame()

REQ02:

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 make his turn;



Supporting software elements:

Class name: JungleGameGUI(), JungleGame(), Player()

Methods: Player(String name)

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

- save [filePath]: To save the current game into file at [filePath].
- 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.

When the command mode still exist before GUI is developed, user was able to enter a String command in the command line program.

During game, command accepts different formats of save/load command in JungleGame.class

| | | |
|---------------------------|----------------------|---|
| No path | : “save” | (Save to game directory / path opened from) |
| Full path / relative path | : “save, C://Users/” | (Save / Load from exact path) |
| Informal path | : “save, C://Users ” | (Path without “/” at the end of string) |

Supporting software elements:

Class: JungleGame, method SaveGame() / OpenSavedGame()

String path = ""; // if command (save / load) has parameters after it, reads the path to this string.

String filename = "/JungleSave.ser"; // Name of save file

String command;

- 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.

As same as above, command move shares the same command reading system like other commands.

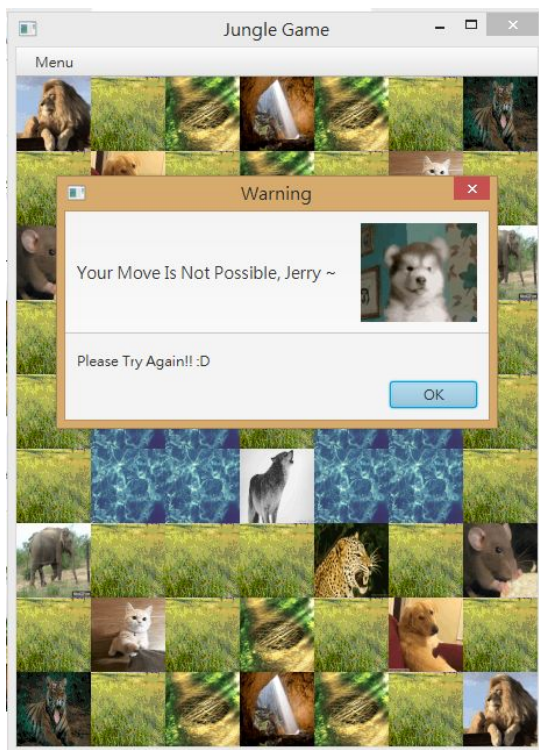
The program accepts a String of command then split them into array using “,”.

step(Player, x, y) in Board class is then called after parsing the coordinates and check if the move is valid utilizing overridden methods in animal classes (e.g. AnimalNotJump, AnimalJump).

Supporting software elements: Class name: Board() Methods: step()

REQ04:

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;



Supporting software elements:

Check Invalid move:

Class name: Board()

Methods: step (Player player, Position start, Position end),

```
validIndex (Position start, Position end)
hasAnimal (Position p)
```

```
isOwnAnimal (Player player, Position p)
```

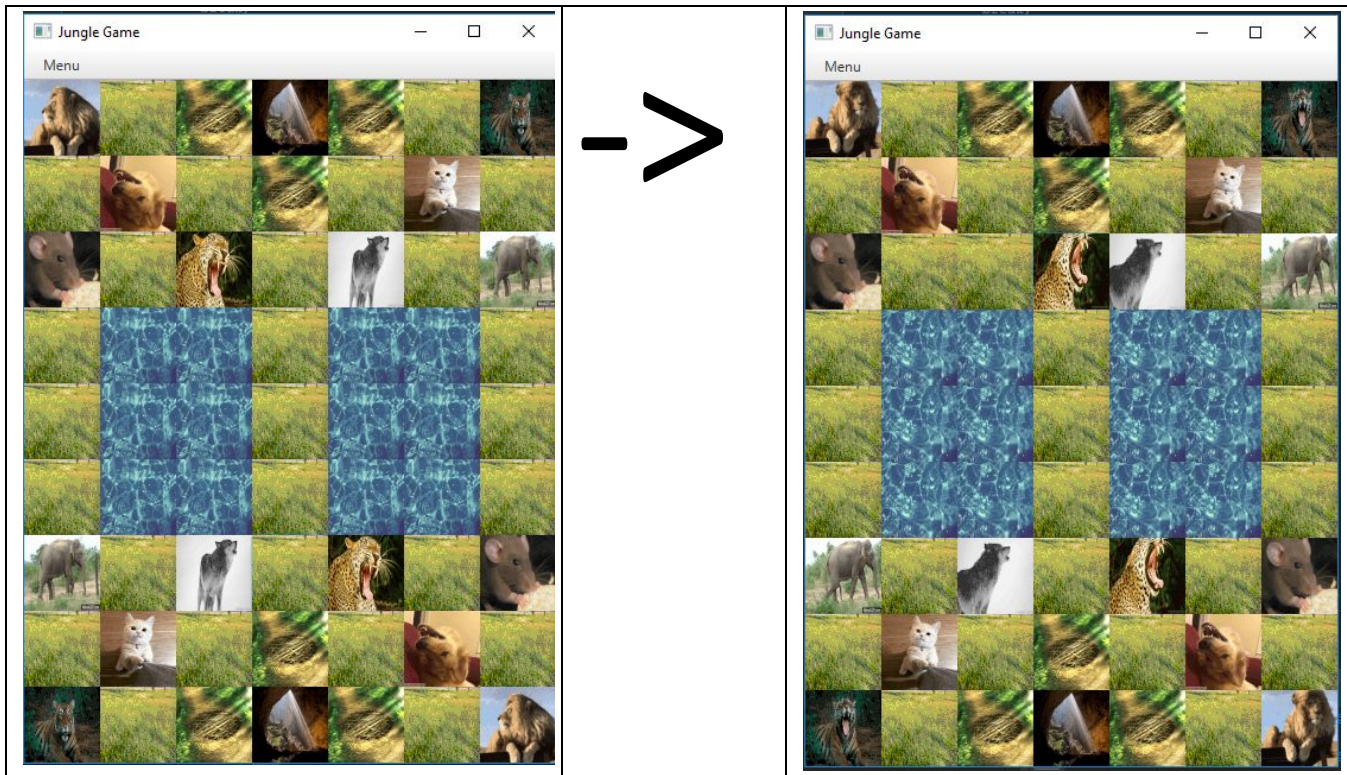
Class name: Animal():

Methods:

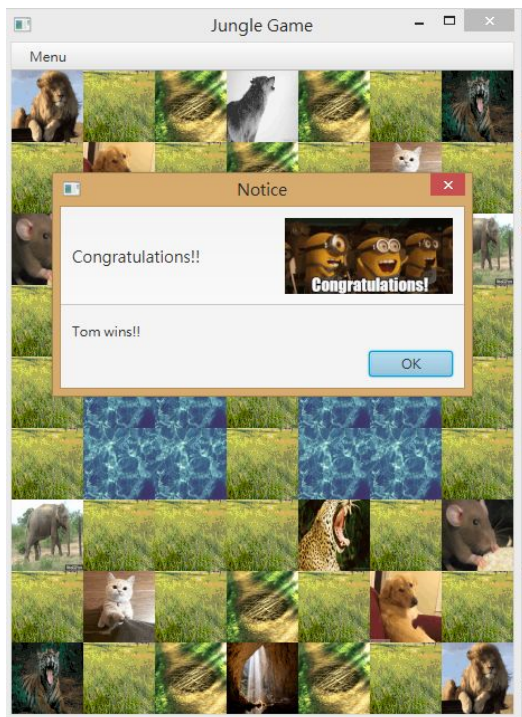
```
isOneMove (Board.Cell[][] board, Position start, Position end)  
canSwim(Board.Cell end)  
isOwnDen (Board.Cell endCell, Animal animal)  
canCapture (Board.Cell endCell)  
ratCheck (Position start, Position end)
```

REQ05: 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;

Printing a valid move in GUI:



Check if a player has achieve its' goal:



Supporting software elements:

Class name: Board()

Methods: isDen (Position end, Player player), move (Position start, Position end, Animal animal)

Next player turn:



Class : JungleGameGUI()

method :SetupButtons()---handle(ActionEvent event1)

REQ06: Upon an invalid command, an error message should be shown, **the same player** should be prompted to make another move. If the invalid command is a move command, the game board will not be updated;

Supporting software elements:

Class name: JungleGameGUI()

Methods: `SetupButtons()`---`handle(ActionEvent event1)`

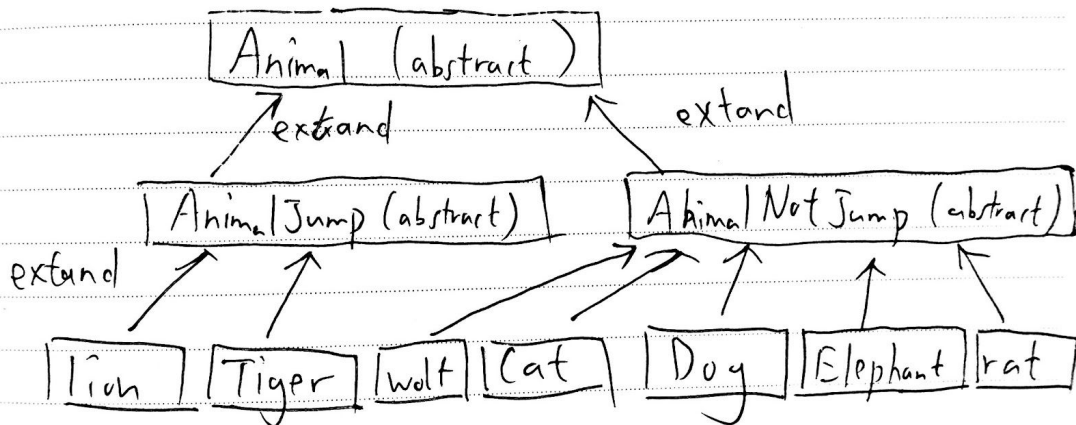
Bonus REQ01: The game should have a full-fledged GUI mode.

The JungleGameGUI is implemented with JavaFX, class extends application object. It provides a full graphical interface for users to interact with the Jungle game with a mouse.

Supporting software elements:

Class name: JungleGameGUI()

2.2) Design



In Jungle game, animal pieces are the most basic components that construct the game mechanics. Therefore class “Animal” is the parent of all the animal classes, representing the every pieces we have on our game board. To further classify the animals, it is classified into animal that can jump : “AnimalJump” and animal that cannot jump across the river : “AnimalNotJump” according to their game behaviour, extended from class “Animal”. The three parent classes are declared as abstract class since instances of parent classes are meaningless and unnecessary. Every specific animal classes (e.g.Rat, Cat, Dog) corresponding to each kind of animal pieces in game, inherit from either “AnimalJump” or “AnimalNotJump”. In the parent class “Animal”, we have abstract method e.g. canSwim() to identify if the animal have the ability to swim through method overriding. Alike ability identifications are implemented in a similar practice.

Besides from the board pieces, another essential element of the game is the game board, a place for the pieces to interact. A game board is a 9 x 7 grid field. The most logical idea is to imagine every grid on the board is a container for a animal piece, and each grid has its own property (e.g. normal, water, trap, enemies’ basement) to define the behaviour when the animals interact. Therefore, a container for animal

“Cell” class and an “Cell” array of 9 x 7 “Board” class will be representing as play field. The board is an electronic version of Jungle Game, it will handle the validation of the players’ move by asking the animal pieces if they are able to do certain actions and checking the move coordinates.

The “Position” enum class is a set of coordinate constants that support the translation between Alphabetical coordinates and pure numeric array indexes.

In the Jungle Game, there are 2 players beyond the board, so we have “Player” class to represent them. Each player has a name, knows how many pieces they left and can be the owner of an animal piece (attribute of Animal class).

The “JungleGame” class act as an observer of game or a stage for the players and the boards. When initializes, it helps setup the board (NewGame / OpenSavedGame) and assigns the players to their board. When the players want to quit their game, the observer will help saving their board intact, remembering whose turn was and wait for the players take it out again.

For each move, the program will check 10 conditions to determine whether a move is valid.

1. Check whether the player select different starting position and ending position (The player has move its piece?)
 - a. method signature: validIndex(Position, Position) in ‘Board’ class
 - b. input parameters: starting position, ending position
 - c. Pass when the starting position and ending position are not the same
2. Check whether the selected starting position has a piece
 - a. method signature: hasAnimal(Position) in ‘Board’ class
 - b. input parameters: starting position
 - c. Pass when the starting position has a piece

3. Check whether the player move his/her own piece
 - a. method signature: isOwnAnimal(Player, Position) in 'Board' class
 - b. Pass when the piece in the starting position is owned by the current player
4. Check whether the player move the piece in a correct way (What animal the piece is?)
 - a. method signature: isOneMove(Board.Cell[[]], Position, Position) (Abstract method in 'Animal' class)
 - b. Pass when the piece move in a correct way
 - i. Lion and tiger can jump over river (when no rat is in between the moving path) and checked using the method overridden in 'AnimalJump' class
 - ii. All other animals cannot jump over river and checked using the method overridden in 'AnimalNotJump' class
 - iii. All animal cannot move diagonally or move more than one square if they move from land to land
5. Check whether the piece can goto water square
 - a. method signature: canSwim(Board.Cell) in 'Animal' class
 - b. Pass when
 - i. Rat can always go onto water square (checked using the method overridden in 'Rat' class)
 - ii. All other animals cannot
6. Check whether the player moves his/her piece to his/her own den
 - a. method signature: isOwnDen (Board.Cell, Animal) in 'Animal' class
 - b. Pass when the player doesn't try to move his/her piece to his/her own den
7. Check whether the selected ending position has a piece
 - a. method signature: hasAnimal(Position) in Board class
 - b. If there is no piece in the ending position, the move is a valid movement to an empty cell
 - c. If there is a piece in the ending position, continues the checking
8. Check whether the player move his/her own piece
 - a. method signature: isOwnAnimal(Player, Position) in 'Board' class
 - b. Pass when the piece in the ending position is owned by the opponent

9. Check whether a piece can capture an opponent's piece
 - a. method signature: canCapture (Board.Cell) in 'Animal' class
 - b. Pass when any of the condition below is met in order
 - i. a piece try to capture an opponent's piece in current player's trap
 - ii. SPECIAL rule: an elephant cannot capture a rat (DO NOT pass)
 - iii. a rat try to capture an elephant
 - iv. the current player's piece has a higher rank than the opponent's piece
10. Check whether a rat try to capture an opponent's piece from land to water square or from water square to land
 - a. method signature: ratCheck(Position, Position) in 'Board' class
 - b. Pass when
 - i. The current piece is not a rat
 - ii. A rat does not try to capture an opponent's piece from land to water square or from water square to land

If the move is an invalid move, the current player need to input command again.

After each valid move, the program will check whether any win condition is met

1. If a piece move to an empty cell, check whether the piece move to opponent's den
 - a. method signature: isDen(Position, Player) in 'Board' class
2. If a piece capture an opponent's piece, check whether all opponent's piece have been captured
 - a. method signature: isZeroPieceCount (Player, Player) in 'Board' class

In the "Position" class, it is a enum class for defining each position and it's coordinate. eg. E2=4,1

For the "JungleGameGUI", it is a class for creating the user interface. We create "Manu" buttons or "New", "Open", "Save" buttons for operating the game mode. It also prints out the grid for every move. And it decide whose turn is it. The GUI also fills in the map after the pieces move. Eg, A1 has a lion, if

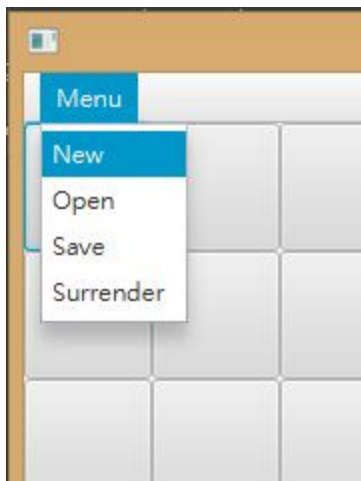
it moves to A2, the GUI will fill A1 as grass. It also handles the message like “Saving error”, “Save Successfully” “which player won the game” .

2.3) Quick Start Guide

This quick start guide acts as a reference for those who use this software. Its describes the commands and functions of this“Jungle Game” software in general terms as well as the details of how to play the game. The following will be divided into several sections.

1) Start a new game

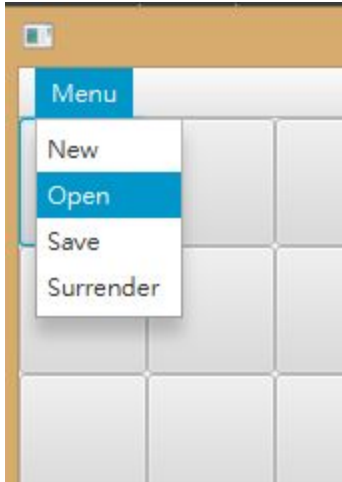
Users can start a new game by pressing the “New” button in Menu.



2) Open a saved game

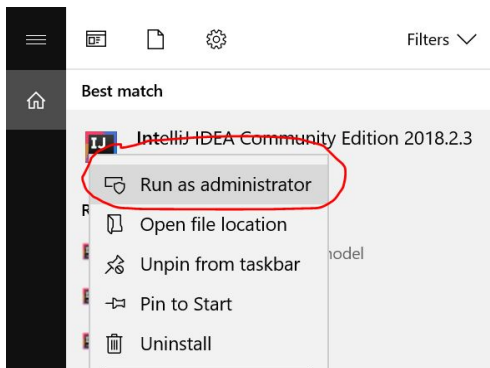
Users can open a saved game by pressing the “Open” button in Menu.

The game will open the saved game automatically.



3) Saved current game

Note: Before using the “Save” command, user should ensure the IntelliJ IDEA is ran as administrator or relevant admin permission is granted to the program.

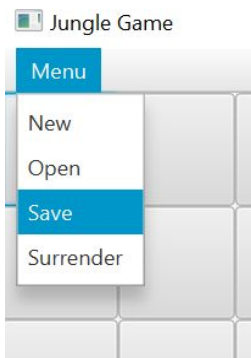


Users can save the current game by using the “Save” in the Menu

First, go to “Menu” at the upper-left corner



Then, click "Save"



The current game will be saved. Users are allowed to load the saved game and resume next time.

A pop-up window will appear afterwards which indicates whether the game has been saved successfully.

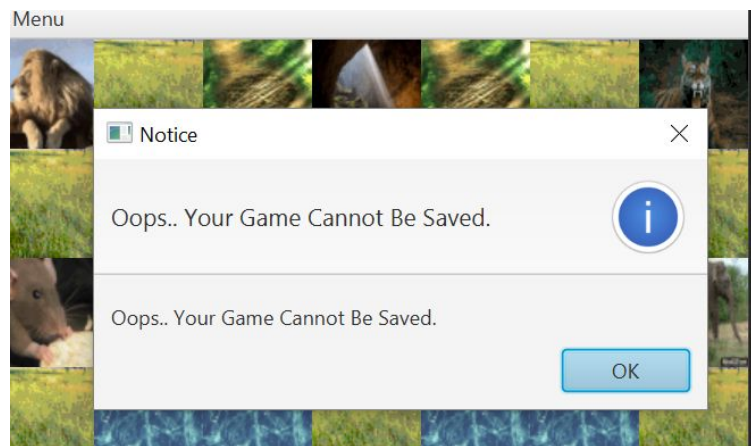


Figure 2a - an pop-up indicating a game is not saved successfully.

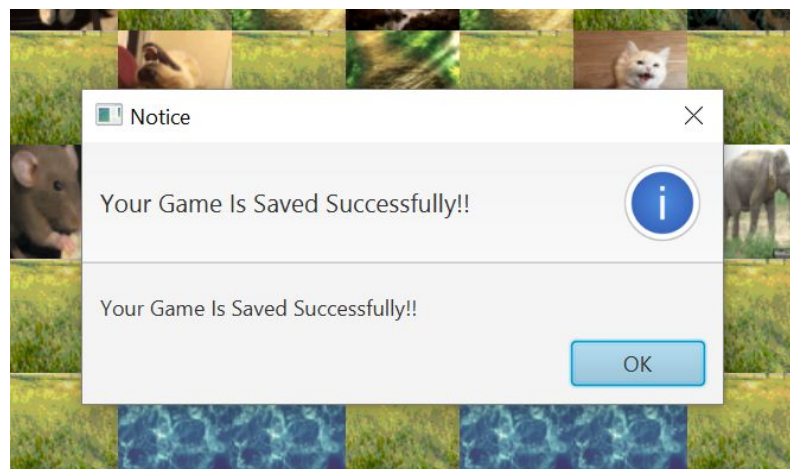
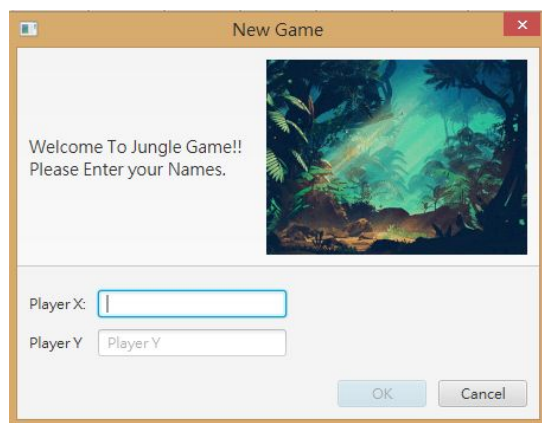
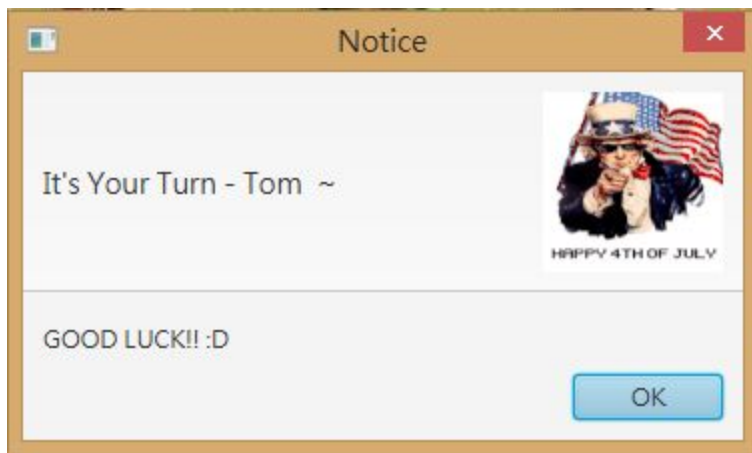


Figure 2b - an pop-up indicating a game is saved successfully.

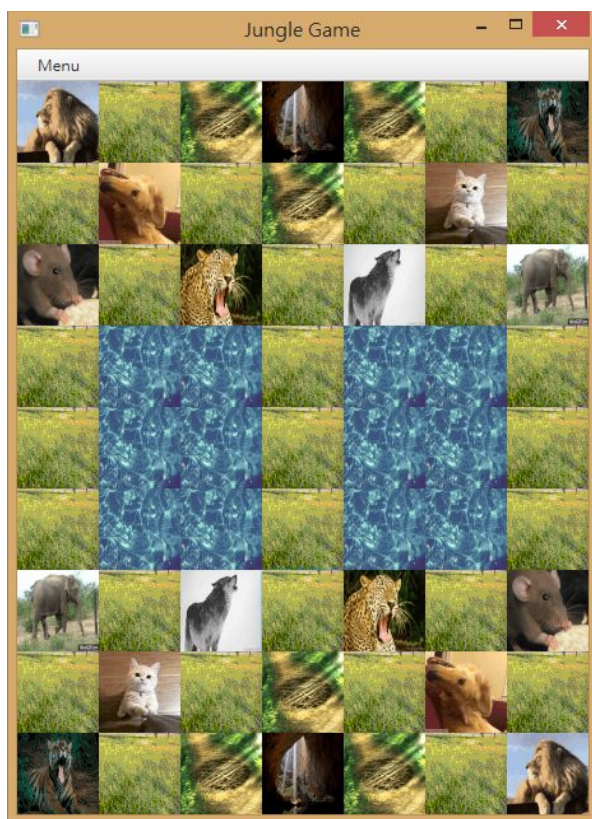
4) Input Player X and Player Y 's name if you add a new game



For each turn, it pops up a message to tell which turn is it.



The bottom is Player X side, and the Top is Player Y side, each time it start with Player X in every game.



5) Move the pieces

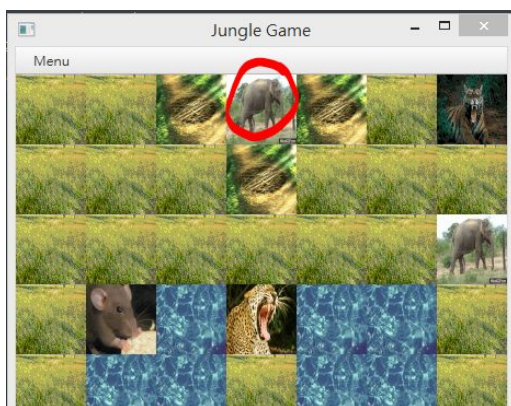
First, click the animal you want to move, then click to it's destination you want it to go.

If your move is not valid, it pops out the error message:

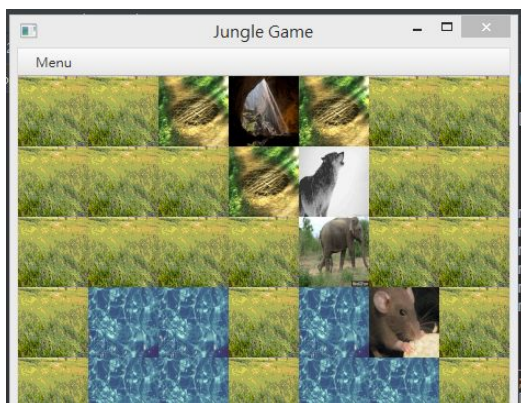


The game ends if:

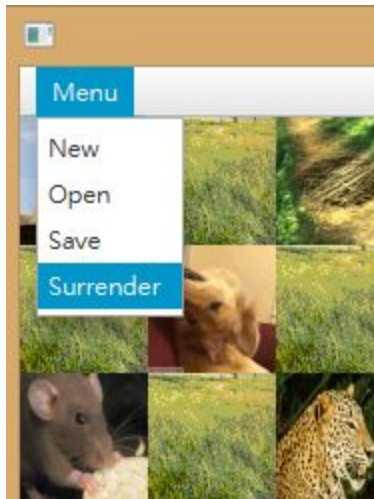
1. step into enemies' den



2. all enemies animals has been eaten



You can also surrender, your enemies will win immediately:



If you win, It pops up:



6) Appendix A

a) 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.

b) 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































| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 9 |  | |  |  |  | |  |
| 8 | |  | |  | |  | |
| 7 |  | |  | |  | |  |
| 6 | |  | | |  | | |
| 5 | |  | | |  | | |
| 4 | |  | | |  | | |
| 3 |  | |  | |  | |  |
| 2 | |  | |  | |  | |
| 1 |  | |  |  |  | |  |
| | A | B | C | D | E | F | G |

Figure 2: The den highlighted in green.



Figure 3: The traps highlighted in yellow.

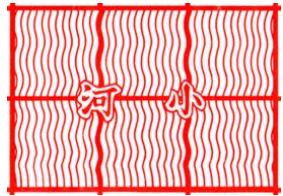


Figure 4: One of the rivers.

c) 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.

d) 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.

e) 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.