COMP3211
Software Engineering
Project Submission1
The Jungle Chess Game
Software Requirements Specification

Members:
Yalu PAN
QI Shihao
Fengkai YU
Ming WANG

1 Preface

This requirement document is completed by Yalu PAN, Fengkai YU, Ming WANG, and Shihao QI for the group project of COMP3211-Software Engineering, Department of Computing, The Hong Kong Polytechnic University. It is assumed based on the scenario, that our group is a game company developing a two-player game, Jungle Game. The document is shared on Google Docs and was written by Microsoft Word. Our first and current version of the game is JungleGame V1.0, and the latest version of the game will be available at our GitHub repo at https://github.com/YaluPAN/SE-Project (the project is currently private). The structure is completed, based on "The Structure of A Requirements Specification (1) & (2)", Lecture 4, COMP3211, as the project requirement. Later continuous updates and development of the game will be strictly based on this requirement document we have written. The document will be visible to course professors, all tutors, as well as all group members of our team. Any commercial use or distribution is prohibited, modification and reposting elsewhere is not allowed as well.

2 Introduction

The Jungle game is a popular two-player strategy game. The game is played on a host, or in other words, which is a single game in which two players use one computer to play. The system's goal is to make the game **ease-use, have many functions to choose from, and be fun for players.**

The jungle game is designed using an OOP method and using an MVC model and layer architecture to construct. **The software is designed in an agile method.**

The document describes the goal and structure of the system, including the functional requirement and non-functional requirements. Besides, the document describes the design and implementation of the system. **You can see detail in part5 and part6**.

The **key** features of the system are:

1	**Fun, easy-use, stable, and low latency** (many functions involved, see part 5 functions for details)
2	The interface is command-line-based (**CLI**)
3	Worked on a host and two players **using one keyboard**
4	Besides playing the game, the players can also **WM** (withdraw a chess move) if the other player agrees.

5	Allow them to **admit defeat** and **request a draw** if another player agrees. Even if the game doesn't end.
6	Many tips to help players play, and there is a time limit for each chess move. And there will be a **reminder** for the last ten seconds to warn users.
7	Use **different colors** to specify which piece belongs to which players.
8	Using the **OOP** method to construct a class and using its instance to simulate the players.
9	To help users to know which step they make a mistake in the game and improve their game technology, **a chess resumption function(CRF) is created**. When a step is allowed, then **it will be recorded into the txt file , and the user can study even after the game is finished.**

The system architecture may change according to future requirement changes.

3 Glossary

CML	Command line
KBL	Keyboard listener
OOP	Object-Oriented Programming
SR	Software Requirement
MVC	The Model View Controller Architecture
CLJG	The command-Line Jungle Game
GIS	Game initial stage
GB	Game Board for players
SRD	Software Requirement Document
WM	Withdraw a chess move
URI	User requirement Index

UML (User case diagram)	describe the relationship between the characters and instances to give the user a brief and clear view.
CRF	a chess resumption function
RD	One user has three chances to request a draw, and if the other user agrees, then the game end and this is a tie; else, game continues.

4 User requirements definition

Index	Description
R1	Users should receive game instructions before the start of the game(GIS).
R2	The game should accommodate TWO players only.
R3	Two players should operate the game turn by turn in the same terminal.
R4	Two players have their standard initialized chess board and that will be printed friendly at the start of the game.
R5	Players will be able to update their chessboard status by moving the chessboard, and the status should be respectively updated and immediately visible to both users on the game board displayed.
R6	The index indicating the role of players should be shown on the game board, and only authorized players should be able to take their turns referring to the respected index.
R7	All the operations of players (moving the chess) should be controlled by the system via command line input.
R8	Players are only allowed to move the chess under the rules of the game. The system will check and decide whether a chosen move is legal or not by providing the category of the chess (Elephant, Tiger, Lion, etc.) and the game rule.
R9	If a player has made an illegal move, his operation should not be updated until a legal move has been made. The illegal move message will be displayed for the user’s reference in the command line.

R10	The system should update the location information of a chess player once its legal move has been made, and the game board should be refreshed again for players to see based on the new location after the update.
R11	Each chess player should always maintain their binary status (Alive or dead), and players should only be able to see their live cheeses on the game board in their respective locations.
R12	Once one player meets the definition of losing the game(based on their chess status), the game should be immediately terminated for users based on the track of their status in the system.
R13	Players will be able to see their results after one game is terminated; they will be either winners or losers, respectively, referring to their chess status.

5 System architecture

5.1 Architectural Design Decisions

Architecture design decisions have a profound impact on the ability of a system to meet critical requirements. To decide which architectural pattern we're going to use, the following questions are considered as the key points that help with the decision.

Question1

Is there a generic application architecture that can act as a template for the system that is being designed?
Ans: Yes. The generic application architecture is the Model-view-controller pattern (MVC). Using the Model-view-controller pattern, the system is structured into three logical components that interact with each other.

Question2

What architectural patterns or styles might be used?
Ans: The Model-view-controller pattern (MVC), Layered Architecture Pattern

Question3

What will be the fundamental approach used to structure the system?
Ans: In software engineering, structured analysis (SA) and structured design (SD) are methods for analyzing business requirements and developing specifications for converting practices into computer programs, hardware configurations, and related manual procedures. To structure the system, first, think of the user manual and user requirements. Then, the corresponding functions will come out.

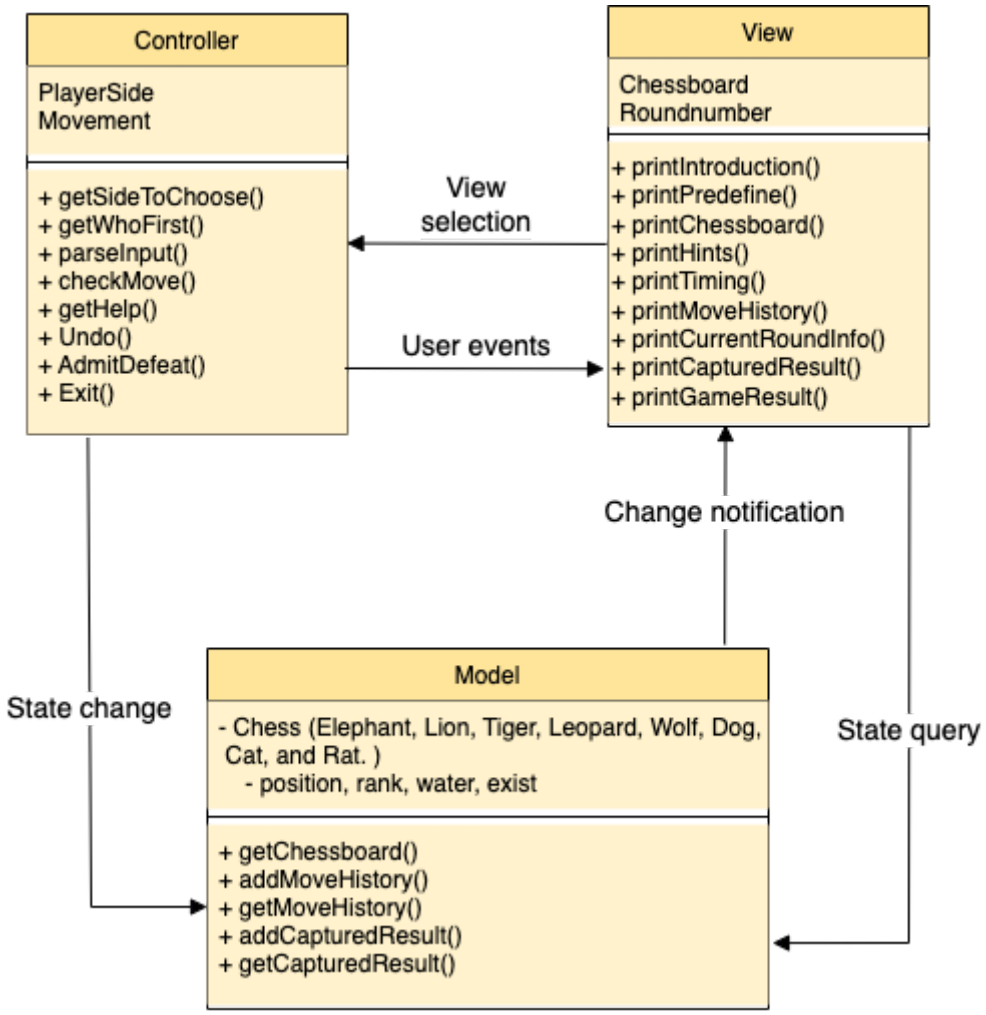
Question4

How will the structural components in the system be decomposed into sub-components?
Ans: To decompose a big component into smaller tasks, the Product Flow Chart is recommended to follow. The pattern focuses on users' input and output and what they will interact with during the experience. A step-by-step processing model will be built to analyze what operations are preferred by users, such as a Hint function and Timing function.

Question5

How should the architecture of the system be documented?
Ans: Architecture should be documented in great detail and presented in a way that is accessible to a wide range of stakeholders.
A code diagram, Container diagram, Component diagram, or System Context diagram is the best way to record.

5.2 Model-view-controller pattern (MVC)



MVC pattern form
(The Jungle Game System using the MVC pattern)

- **The Model Component**
  The model component manages the system data and associated operations on that data. In our command line jungle game (CLJG), the model component stores the data structure used to record chess movements and positions and contains the function which can return animal chess position and existence.

  Class: Chess
  subclass: Elephant, Lion, Tiger, Leopard, Wolf, Dog, Cat and Rat
  attribute: position, rank, water, exist

Index	Function	Purpose
1	def getChessboard()	get every animal's attribute information from the data structure.
2	def addMoveHistory()	add the player's movement decision into a txt. file in every round chess resumption function
3	def getMoveHistory()	get the player's movement history from the txt. file chess resumption function
4	def addCapturedResult()	add every captured result into another txt. file once the player captures any enemy pieces
5	def getCapturedResult()	get all previous captured results from the txt. file

- **The View component**
  The view component defines and manages how the data is presented to the user. In order to maximize player satisfaction, we have designed various functions in the view component.

Index	Function	Purpose
6	def printIntroduction()	Display a brief system introduction and game rules before the game starts.
7	def printPredefine()	Ask players' preferences about chess color and language (English or Chinese)
8	def printChessboard()	Print and display the animal chessboard after every update

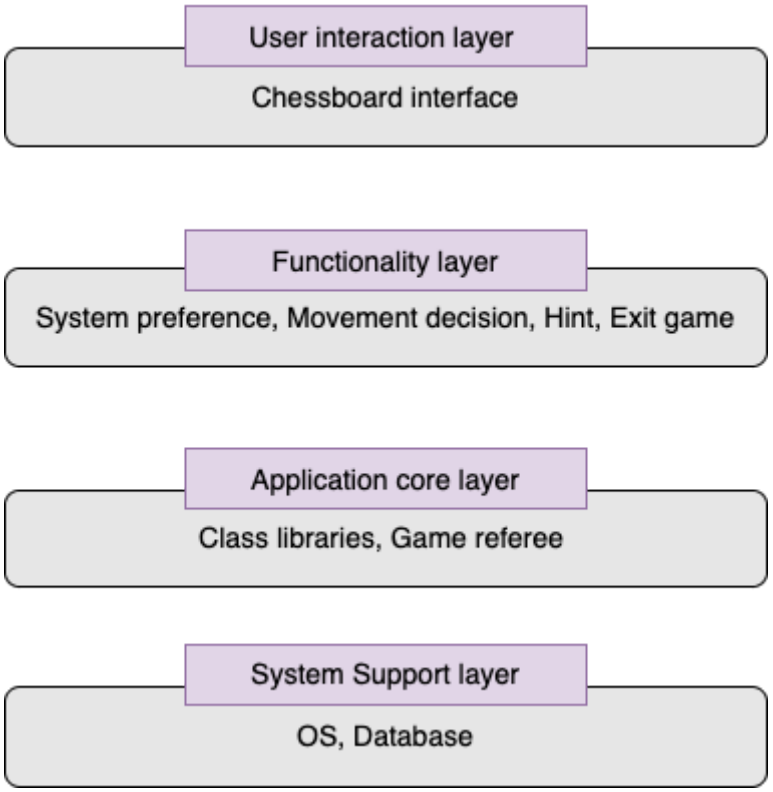
9	def printHints()	Print hints about players' movement: - **Hint1** "illegal syntax": the player is only allowed to input a sentence in a form like "animal name + move direction (u, d, l, r)." - **Hint2** "Moving out of board range" - **Hint3** "Can't move into the river": Except for mice, no other animals can enter the river area - **Hint4** "Lion/tiger is not allowed to jump over the river, because there is a rat on one of the rivers" - **Hint5** "The rat is entering the river, it's only allowed to capture the opponent rat when they're both in the river" - **Hint6** "The rat is entering the land; it's allowed to capture the elephants and rats(if the opponent rat is also on the land)" - **Hint7** "A piece may not move to its own den" - **Hint8** "You're going into a trap. Any opponent animal regardless of rank can capture you"
10	def printTiming()	To indicate the time that each player is left at every round
11	def printMoveHistory()	To print and display players' moving history. Included Information: movement order + animal moved + moving direction
12	def printCurrentRoundInfo()	Display the current round number and which player should operate in the screen after every update
13	def printCapturedResult()	To print each player's cumulative captured chess results after every update. Included information: captured order + captured animal + animal's rank
14	def printGameResult()	To print and announce the gaming result

- **The Controller component**
  The Controller component manages user interaction and passes these interactions to the View and the Model. The component has several parts. 1) Before the game starts,

the system will ask the players which side they want to pick. Players are expected to input their favorite side letter (A or B) in the command line. 2) The player below will play first. The player is expected to input the specific animal and movement (up, down, left, and right) in the command line. 3) The player is allowed to undo one movement by typing “undo” in the command line. 4) Both players are playing and deciding their movement on one computer, who will play chess in order.

Index	Function	Purpose
15	def ConfirmBeforePlay()	After printing a brief introduction and game rules to the users, he/she will be asked to confirm to play the game by entering “confirm”
16	def getSideToChoose()	Ask players which side they want to choose
17	def getWhoFirst()	Ask players to decide who starts first
18	def parseInput()	the player is expected to input the specific animal and movement (up, down, left and right) in the keyboard listener
19	def checkMove()	Check whether the syntax input is valid. Check whether the movement requested is valid
20	def getHelp()	The system will show help information about game rules once players input “help” in the command line
21	def Undo()	The player is allowed to undo one movement by typing “undo” in the command line
22	def AdmitDefeat()	The player is allowed to input “admit defeat” once he feels he is going to fail or end the game
23	def Exit()	exit the game

5.3 The Layered Architecture Pattern



Layered Architecture Pattern Form
(The Jungle Game System using the Layered Architecture Pattern)

Layered Architecture Pattern is one of the most common architectural design patterns in software engineering. Therefore we implemented this pattern into our system. There are four layers in the system, user interaction, functionality, application core, and system support layers. Every layer has a mission, which is a logical unit that separates a specific role from the responsibilities. So, unmanageable tasks can be divided into smaller ones that are easier to handle.

- User interaction layer
  The system displays a chessboard interface to players, which is the most basic user requirement.
- Functionality layer
  In order to give players the best user experience, the system provides basic operation (e.g. Movement decision), personalized choices (e.g.System preference), easy-to-play reminders (e.g. Hint), and exhaustive visualization (e.g. PrintHistory, Exit) to users.
- Application core layer
  The system has class libraries that store the user's movement data and a game referee mechanism to run over the game.
- System Support layer
  the system relies on the operating system and database to maintain.

5.4 Architectural Components Reused

Reusing software components is a software engineering practice that involves using existing components to build new applications rather than creating them from scratch. Reusable components can be requirements specifications, source code, user interfaces, design documents, user documentation, or any other items related to software. We've tried to find as many reusable components as we can in order to have a good quality product in fast development. Below are software reused components:

	Reused Components	Detailed
1	Architecture design pattern	- Model-View-Controller Pattern (MVC) - Layered Architecture Pattern
2	Function libraries/packages	- Colorama: a Python function library used to render terminal text in color and different styles. - Time Module: it allows users to get the current time and calculate the time remaining.
3	User interfaces	A common chessboard playing interface will be reused
4	Requirements specification document	The structure of this document is based on "The Structure of A Requirements Specification (1) & (2)" from Lec4: Requirement Engineering

6 System requirements specifications

6.1 Functional Requirement

Main Target	URI	System requirement index	Priority
Instruction representation	R1	6.1.1 6.1.2 6.1.3	1
Game board	R2	6.1.4	1
Game board maintenance	R3, R4, R5, R6	6.1.5 6.1.6	1
Command Line input	R7	6.1.6	2
Pieces operation	R8, R9, R10	6.1.8	2
Piece capture	M4, M5,	6.1.9	1
Live or Death	R11, M20, M21	6.1.12 6.1.13	1
End Game	R12, R13	6.1.10 6.1.11	1

Related Requirement Index points to the user requirement definition(e.g. R1, R2) and model component function index(e.g. M1, M2)

Before a game starts

6.1.1

the system shall provide both brief system instructions and game rules specified in the welcome message

6.1.2

The system shall present a sequence of valid commands to the user and mark out several common improper command examples. A command quick-search function is also preferred during system instructions.

6.1.3

The system shall warn the user in instructions that the game won't be saved if the game is terminated, whatever in what conditions. Only step recording will be saved to a txt file for review.

6.1.4

After the game user finishes the instructions, the system shall come into the game's initial stage yet allow users to select:

(a) "Choose the language": the system shall provide two optional languages for users, which are English and Chinese. The system shall switch the corresponding settings on pieces and rivers.

(b) "Pick one side": the system shall clearly indicate two sides and ask users to enter which side they will hold. The initial game stage should be completely symmetric. After the user picks the side, the color of pieces of two sides should be represented in an easy discern way, such as by applying different colors.

(c) "Who starts first": the system shall regard users as hosts and require them to enter a number between 1 to 10. After receiving correct input, the system shall generate a random value between 1 to 10 and compute whose number is closer to the generated one, then decide which user should start first.

While gaming in the process

6.1.5

The system shall clearly state the situation of each side and also every piece's position. Presents an exhaustive and transparent game board.

6.1.6

During each turn, the system shall list:

- Game board after user input.

- User's pieces taken by opponent and opponent pieces taken by the user.
- Current round number and which user should operate.

6.1.7

The system shall count a limited consideration time for each user's turn because it's apparently unfair to allow a user to determine the next step without time limitation.

6.1.8

The system shall use certain data structures to record every step of users. At every turn, the system should provide the user an opportunity to undo his/her decision. An extra board will be shown to achieve this when one user wants to undo the decision. The undo/redo should be processed only after getting permission from both users.

6.1.9

The system shall strictly constrain the operation of different pieces and prevent possible logistic faults:

Rats	• Only rats are allowed to go into the water square; the system shall remind the user if it detects any prohibited pieces operations. • Rats are only allowed to devour the elephant. The system shall firmly restrict the rank rules of rats.
Capture	• The system shall judge the validity of the capture operation among different pieces and represent the result to the user. E.g., the system shall notify the user which pieces are eliminated in one capture operation since there is a possibility for the user to force low rank one to attack high-rank pieces.
Order checking and malicious operation warning	• The system shall prompt user operation where they try to force their pieces across the river in one step or spontaneously cause pieces out of bounds.
River	• The river will block the forward order, except for tiger, lion, and rat; other animals have to go around it.

Tiger and Lion	• The system shall specifically allow piece tigers and lions to jump through the river horizontally or vertically. • The system shall ignore the "jump through" features of the tiger and lion when there is a rat in the current river.

6.1.10

The system shall announce the winning under two situations

"One user's den is captured": system shall declare the winner immediately after a piece steps into the opponent's den

"One user's all pieces are eliminated": system shall declare the winner after detecting one user has no pieces to move except the den.

"One user admits defeat": this is a special situation. Before each turn starts, the system shall check whether one user admits he/she defeats. The system will represent the winner after the opponent input "admit defeat".

After a game ends

6.1.11

The system shall state the winner, pieces eliminated by the opponent, rounds number, every user's total time used, and the total scores of each user.

6.1.12

The system shall record every determined(non-WM) step and save it into a txt file for later review. (CRF) The pieces' type, forward direction, and step shall be recorded

6.1.13

The system shall provide two options for the user:

- Continue: the system shall restart a new board and bring users back to requirement 6.1.3
- Exit: the system shall thank the user for using and wash they have a good day, then close the tab

6.2 System Non-functional Requirements

This section state system non-functional requirements applied to the system based on section 4.

6.2.1 Reliability

The system shall be operated whenever users want to use it. The only requirement shall be limited to programming language configuration.

The reflection of user requirements or backend processing shall be within 1-2 seconds to provide a comfortable experience.

6.2.2 Performance

Environment: the system operator interface shall be similar to the consult simulator, which contains the following:

- User input and output command area
- Graphic content with a comprehensive representation and easy-discern labels of two side pieces
- Every turn should be processed within 30 seconds

6.2.3 Usability

- Instructions shall be easy to understand and provide an exhaustive illustration of the whole game.
- The system shall be manipulated and operated by the user with no confusion or difficulty for the one who is familiar with the jungle game.
- The result shown after every turn is expected to present correct and impartial graphic content. No out-of-bound or improper pieces step should be allowed to exist.

6.2.4 Compatibility

The system shall implement the game without importing non-default libraries and avoid system crashes brought by the different main OS. So, the user can access the game from the command line smoothly and doesn’t need to be concerned about protentional bugs.

6.2.5 Security

- The system shall prevent access to any underlying configuration of the user’s computer or insert invalid plugs.
- The system shall protect the user’s operation won't be infected by another one from a computing perspective.

6.2.6 Interface

The interface should implement as 9 square long and 7 square wide areas, marked with pieces in either Chinese or English. It will change together with the game process. While the same time, the eliminated piece will be put on the right side of the board.

The implementation of the interface will strictly follow the definition from R2, R3, M2, and M3 to establish the representation framework.

6.2.7 Maintainability

- Using python to implement the game
- Pygame library will be applied

7 Appendices

These should provide detailed, specific information that is related to the application being developed, for example, hardware and database descriptions. Hardware requirements define the minimal and optimal configurations for the system. Database requirements define the logical organization of the data used by the system and the relationships between data.

Reference

1. command-line java
   https://github.com/jgladch/chesstron
2. command-line-chess-cpp
   https://github.com/valentynpatsera/command-line-chess-cpp
3. python
   https://github.com/marcusbuffett/command-line-chess
4. jungle game: (windows)
   https://www.mobygames.com/game/windows/jungle/screenshots/gameShotId,847087/

Jungle Game board rules

The jungle game board consists of seven columns and nine rows of squares. The pieces move on square spaces like chess, not on lines like Chinese chess. Pictures of eight animals and their names appear on each side of the board to indicate the initial positions 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: Nests are located in the center of the board's border row and are marked in Chinese. The traps are located on the sides and front of the nest and are also labeled in Chinese. There are two bodies of water or rivers in the center of the board: each consists of six squares in a 2X3 rectangle and is marked with the Chinese character for "river". There is a single row of plain land squares in the middle between the edge of the chessboard and the river.

8 Index

Several indexes to the document may be included. As well as a normal alphabetic index, there may be an index of diagrams, an index of functions, and so on.