Course Project Description

In this project, each group needs to apply the knowledge learned in the course to develop a command-line-based Jungle game (hereinafter referred to as "the game"). Appendix B gives more information about the game.

Group Forming: Form your groups of 3 to 4 students from the same or different class sessions on Blackboard before 14:00, 8th October 2025 (Wednesday). Afterward, students with no group will be randomly grouped, and requests for group change are only approved if written agreements from all members of the groups involved are provided before 14:00, 13th October 2025 (Monday).

Object-oriented vs. Procedural-oriented: You are suggested to design and implement the game in an object-oriented way because OO techniques often help produce software systems with better quality in terms of, e.g., code cohesion and coupling, understandability, and maintainability. That, however, does not mean you will be penalized for choosing the other way. We will consider the capabilities and limitations of each design and programming paradigm when grading, so if you feel more comfortable working with the procedure-oriented paradigm, or if you don't have enough time to learn the OO techniques for this project, feel free to adopt the procedure-oriented way. Particularly, you can program the game in either Java or Python. If you want to use another programming language, consult your instructor first.

Deliverables: Each group needs to submit the following deliverables, compressed into a single ZIP archive, before 14:00, 24th November 2025 (Monday). Unless explicitly stated otherwise, all the required documents in the deliverables should be in DOC, DOCX, or PDF format.

1. **Software requirements specification (SRS). (6 points)**
   You may decide on the specifics regarding how users will interact with the game, but all the rules given in Appendix B should be obeyed, and the user stories in Appendix C should be supported. Your requirements should be valid, consistent, complete, realistic, and verifiable.

   a. *Document structure*: See slides "The Structure of A Requirements Specification (1) & (2)" from "Lecture 04: Requirements Engineering" for the overall structure of a requirements specification; Note that chapters "System models", "System evolution", "Appendix", and "Index" are not needed in your requirements specification.
   b. *Contents*: All the important functional and non-functional requirements should be included.
      - See slides "Natural Language Specification" and "Example Natural Language Specifications" from the same lecture for guidelines on and examples of writing natural language requirements.
      - To facilitate unit testing (see Deliverable 4), you should put all the code that belongs to the model of the game into a separate package named "model". You may refer to the Model-View-Controller (MVC) pattern [a)] to help you understand what constitutes the model of the game. We will discuss the MVC pattern in "Lecture 6: Architectural Design".
      - It is not recommended to add obviously additional features, e.g., a GUI or support for online use, to the game. Your efforts to design and implement the additional features will not get you extra credit.

2. **Design document. (5 points)**
   The deliverable should contain 1) a diagram to describe the game's architecture, 2) diagrams to show both the structure of and the relationship among the main code components of the game, and 3) a diagram to outline what happens during a user's turn of playing the game, i.e., between two adjacent users rolling the dice.
      a. Architecture: Explain which generic architectural pattern you picked for the game, why you chose this pattern, and how the components of the pattern were

[a)] https://en.wikipedia.org/wiki/Model-view-controller

instantiated in the context of the game. ("Architectural Design" will be discussed in Lecture 6.)

b. *Structure of and relationship among main code components*: If you adopt object-oriented techniques, explain the classes in your design and their fields and public/protected methods; If you adopt procedural-oriented techniques, explain the user-defined data types and functions in your design. For each method/function, the explanation should include information about the return type, the method/function name, the argument names and types, and possible exception declarations. Explain how the major code components are related to each other in the game.
c. *Example use*: Demonstrate the collaboration among the code components using a sequence or activity diagram.
d. *Necessary textual explanations*: Each diagram should be accompanied by necessary textual explanations to facilitate understanding.

3. **Implementation. (6 points)**
The deliverable should include 1) the complete source code for the game, 2) a developer manual explaining how to compile the code and build the project, 3) a user manual describing how to play the game, 4) a short video (no more than 4 minutes) of the game in play, and 5) a short report on requirements coverage achieved by your implementation.
   a. *Source code:* The code implementing the model of the game should be placed in a separate package named "model", and the other major code components should also be easily identifiable in the source code. Your implementation should only use Java/Python standard libraries.
   b. *Developer manual*: Here, you may assume the readers know how to properly set up a Java/Python development environment, and you don't have to explain that in the manual. You only need to prepare the manual for one platform, e.g., Windows, macOS, or Ubuntu. You need to explain which version of JDK/Python was used for the development, which IDE should be used to open your project, which commands (and parameters, if any) should be used to build your project, and/or how to launch the game in the debugging mode.
   c. *User manual*: In the user manual, you need to explain things like which input commands are supported, what their formats are, what happens if an input command is invalid, and how to interpret the output of the game. Everything a user needs to know to play the game effectively should be explained in the user manual.
   d. *Video*: The video should be in MP4 format. Given the limited length of the video, you only need to demonstrate the game's most important features. A feature can be important for the user because it is a basic operation or it greatly influences the user's experience, and it can be important for you, as the developer, because, e.g., you are proud of the design behind the feature. You don't need to verbally explain everything in the video since most interactions between the game and the players should be straightforward, but feel free to add narratives and/or texts to the video to describe the things that are less obvious and/or worth extra attention.
   e. *Requirements coverage report*: Report in a table which user stories from Appendix B and which requirements in the SRS have/have not been implemented. Put the report in the root folder of your source code.

4. **Unit tests for the game model. (4 points)**
Each unit test (that's right, you only need to write unit tests in this project) should clearly state the functionalities it exercises (e.g., using comments) and the expected results (e.g., using assertions).
   a. *Unit testing framework*: All the unit tests should be automatically executable and should pass when executed against your game implementation. You are strongly recommended to prepare the tests using unit testing frameworks. Two example unit tests, in JUnit for Java and unittest for Python, respectively, are given in Figure 1 for your reference, but feel free to use other unit testing frameworks if you so desire. Please refer to the corresponding documents and/or tutorials for information about

writing unit tests based on different unit testing frameworks.

```
// JUnit for Java
...
public class JUnitProgram {
  @Test
  public void test_JUnit() {
    String str1="testcase ";
    assertEquals("testcase ", str1);
  }
}
```

```
# unittest for Python
import unittest
class Testing(unittest.TestCase):
    def test_string(self):
        a = 'some'
        b = 'some'
        self.assertEqual(a, b)
if __name__ == '__main__':
    unittest.main()
```

Figure 1. Example unit tests in Junit for Java and unittest for Python, respectively.

b. *Test coverage:* Report the line coverage achieved by your tests on the game model in a separate file and place the report in the root directory of your tests.

5. **Presentation slides and recording (4 points)**
   The project presentation will be delivered in the last week of the semester. Each group has 4.5 minutes for the presentation and 1 minute for Q&A. More information about the organization of the presentations will be announced later.
   a. *Presentation contents*:
      - The game UI design regarding 1) the supported user commands, 2) the output of the game status, and 3) the error handling mechanism, respectively.
      - The overall design of the game, i.e., the combination of Deliverables 2.a and 2.b.
      - One important lesson learned from the project regarding requirements engineering, API design, or unit testing.
   b. *Presentation slides and recording:* The slides should be in PDF format, and the recording should be in MP4 format.

Appendix A. Grading Criteria

1 is for the lowest quality, coverage, etc.; 5 is for the highest.

Software requirements specification	1	2	3	4	5
Deliverable completeness	☐	☐	☐	☐	☐
Document quality (in terms of contents, structure, and writing)	☐	☐	☐	☐	☐
Requirements quality (in terms of validity, consistency, completeness, realism, and verifiability)	☐	☐	☐	☐	☐
Design document					
Deliverable completeness	☐	☐	☐	☐	☐
Document quality (in terms of contents, structure, and writing)	☐	☐	☐	☐	☐
Design quality (in terms of modularity, efficiency, extendibility, justifiability, etc.)	☐	☐	☐	☐	☐
Diagram quality (in terms of completeness, well-formedness, understandability, etc.)	☐	☐	☐	☐	☐
Implementation					
Deliverable completeness	☐	☐	☐	☐	☐
Document quality (in terms of contents, structure, and writing)	☐	☐	☐	☐	☐
Code quality (in terms of reliability, readability, etc.)	☐	☐	☐	☐	☐
Requirements coverage	☐	☐	☐	☐	☐
Unit tests for the system model					
Deliverable completeness	☐	☐	☐	☐	☐
Test quality (in terms of readability and granularity)	☐	☐	☐	☐	☐
Test coverage	☐	☐	☐	☐	☐
Presentation slides and recording					
Deliverable completeness	☐	☐	☐	☐	☐
Presentation quality (in terms of contents, length, and delivery)	☐	☐	☐	☐	☐
The presentation is properly timed.	☐	☐	☐	☐	☐

Individual grading. In the project, each group member is expected to actively participate and make fair contributions. In general, group members will receive the same grade for what their group produces at the end of the project. Individual grading, however, could be arranged if any member believes "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 contributions to the project, and each member will be asked to provide evidence for his/her contributions to facilitate the grading. **Should you opt for individual grading, you need to send the request in writing to your instructor before 14:00, 24th November 2025 (Monday).** Requests sent afterward will not be entertained.

Use of GenAI tools. Using GenAI tools in the project is allowed, but you should properly acknowledge all the contents in your deliverables that were created with the help of those tools. More specifically, each group must fill out the "**Honour Declaration for Group Project**" and include it within the final submission, even if the group has not used GenAI in the project. You may refer to https://www.polyu.edu.hk/ar/docdrive/polyu-students/Student-guide-on-the-use-of-GenAI_revised_250214.pdf for more guidance on the use of GenAI. **Failure to submit the "Honour Declaration" or submitting a false declaration will result in a penalty of up to 40% of the total points.**

Appendix B. The Jungle Game

The concise introduction to the Jungle game here is based on the description at https://en.wikipedia.org/wiki/Jungle_(board_game).

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 the 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 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 in the center of the board: each comprises six squares in a 2X3 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.

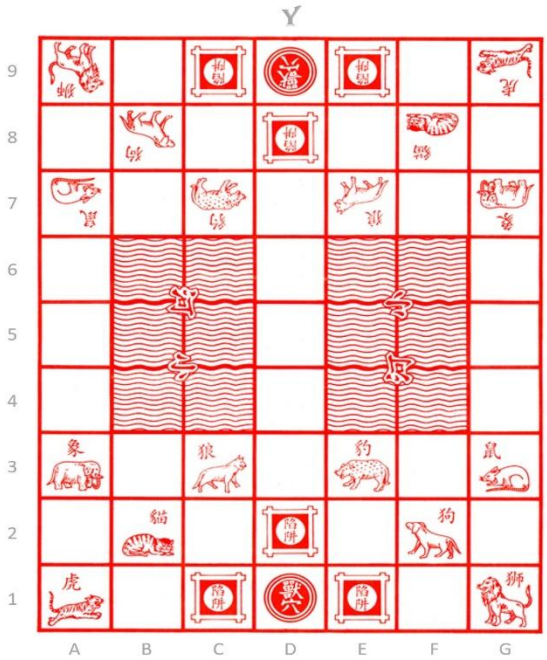

Figure 1: A typical Jungle gameboard showing the locations of starting squares, the den, rivers, and traps.

Figure 2: The den highlighted in green.

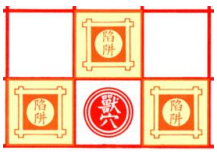

Figure 3: The traps highlighted in yellow.

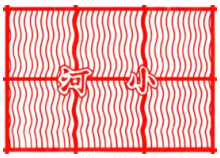

Figure 4: One of the rivers.

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.

Table 1: Pieces and their ranks.

Rank	Piece (en)	Piece (cn)
8	Elephant	象
7	Lion	獅
6	Tiger	虎
5	Leopard	豹
4	Wolf	狼
3	Dog	狗
2	Cat	貓
1	Rat	鼠

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 type of 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.

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.

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 the opponent's pieces.