Course Project Description

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

Group Forming: Form your groups of 3 to 4 students on Blackboard before or on Sept. 30, 2022. 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 involved groups are provided before or on Oct. 12, 2022. No group change will be allowed after Oct. 12, 2022. It is okay for students from different class sessions to team up.

Object-oriented vs. Procedural-oriented: You are suggested to design and implement the game in an object-oriented way because it makes certain desirable code properties, e.g., high cohesion and low coupling, easier to achieve. That, however, doesn't mean your grade will be lower if you choose the other way. We will consider the limitations of each design and programming paradigm when grading, so if you feel more comfortable 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. You may 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 produce the following deliverables. While different deliverables have different deadlines, make sure you first read the whole description carefully to get an overview of what you need to do in the project.

1. **A software requirements specification for the game. (8 points)**
   You may decide on the specifics regarding how users play the game, but all the rules given in Appendix B should be respected, and the requirements should be valid, consistent, precise, complete, 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” and “System evolution” are not needed in your requirements specification.
   b. *Contents*: All the important requirements, both functional and non-functional, 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 3), you are strongly recommended to adopt the Model-View-Controller (MVC) pattern [a)] for your game. Since MVC is a generic pattern, you need to explain how you would apply that generic pattern in the context of the game. Particularly, in the System Architecture chapter, you need to describe the overall structure of the game, the reasons for choosing that structure, the major components in that structure, and how those components interact with each other. (“Architectural Design” will be discussed in Week 6.)
      - The rules in Appendix B reflect some, but not all, important requirements of the system, and they are not written in the form of software requirements.
      - You are not recommended to add obviously additional features, e.g., a GUI or support for playing online, to the game. Your efforts to design and implement the additional features will not get you any extra points.
   c. *Deadline*: Oct. 12, 2022.

2. **An API design document of the game. (7 points)**
   The deliverable should contain 1) a collection of code components like class and method declarations, 2) diagrams to demonstrate the relation and collaboration among the code components, and 3) a brief description of the important design decisions. It is okay to prepare the code components in the form of source code files, while the diagrams and

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

description should be put into a single Word/PDF file. Compress all the files into a ZIP archive and submit the ZIP archive.

a. *Code components*: If you adopt object-oriented techniques, the classes in your source files should contain major fields and public/protected methods; If you adopt procedural-oriented techniques, your source files should include user-defined data types and functions. Each method/function should contain a return type, the method/function name, argument names and types, possible exception declarations, and an empty body. Note that you don't need to implement the methods/functions yet.
   Since Python is dynamically typed, you cannot include return and argument types in Python method declarations. In that case, you could annotate your Python method declarations with type hints (https://docs.python.org/3/library/typing.html).
b. *Diagrams*: The diagrams should demonstrate the relations and the collaborations among the code components. The relations between components can be modeled using class diagrams, while the collaborations can be modeled using activity and/or sequence diagrams. Note that the classes in a class diagram represent realistic or conceptual things in software systems, so class diagrams are also useful even if you plan to develop your game in a procedure-oriented way.
c. *Important design decisions*: Any decision you make when designing your game is a design decision, and these decisions eventually determined the details in your design. Here you can decide by yourself which aspects of your design are important. Such aspects may be, e.g., parts that you think were difficult to design, parts where you spent the most time to come up with the design, parts where you think your design was smart and worth mentioning, or parts that are critical for the game.
d. *Deadline*: Oct. 21, 2022.

3. **A suite of unit tests for the game model based on the API design. (7 points)**
   Writing unit tests based on the API design is an effective way to check whether the APIs defined are consistent and sufficient for implementing the requirements. Each test should clearly state what functionalities they exercise (e.g., in comments), how the functionality is exercised step by step, and what the expected results are (e.g., using assertions).
   a. *Unit testing framework*: Since all the unit tests should be automatically executable, you are strongly recommended to prepare the tests using unit testing frameworks. Here are two example unit tests, in JUnit for Java and unittest for Python, respectively, for your reference.

```
// 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()
```

   You may also use other unit testing frameworks, but it is important that your unit tests, once combined with the implementation of your game, should be executable. Please refer to the JUnit and unittest documentations/tutorials for more information about writing unit tests based on those frameworks.
   b. *APIs to test*: As stated above, you only need to write unit tests for APIs defined in the model, but not the view or controller, of the game.
   c. *Revisions to the API design*: If you need to revise your API design when preparing the tests, you should also include in the deliverable a separate document to explain the revisions you made and the reasons for the revisions.
   d. *Deadline*: Nov. 4, 2022.

4. **An implementation of the game. (9 points)**
   The deliverable should include 1) the complete source code for the game, 2) the final suite of unit tests and its coverage of the source code lines in the game model, 3) a short developer manual explaining how to compile the code and build the project, 4) a short user manual describing how to play the game, and 5) a short video (no more than 4 minutes) of the game in play. The implementation should support all the requirements listed in the requirements specification and make a playable game.
   a. *Source code and unit test suite*: If you need to revise your API design or tests during the implementation, you should include in the deliverable a separate document to explain the revisions and the reasons for them and discuss what you could have done differently to avoid such revisions. Add a separate TXT file for the test coverage results and place the file together with your test files.
   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 should be used to compile/build your project, and/or how the game can be started in the debugging mode. Besides, if you utilize extra libraries and/or tools in your development, you should also describe those libraries and/or tools and explain how they were used in the developer manual.
   c. *User manual*: In the user manual, you could refer to the game rules for the effect of most regular game operations. Meanwhile, you also need to explain other things that are not covered by the game rules. For example, you need to explain which input commands are supported, what their formats are, how users should input the commands, what happens if an input command is invalid, how to read the game board printed by the game, and how to interpret error messages. Essentially, everything that users need to know to be able to play the game, except for the game rules listed in Appendix B, should be included in the user manual.
   d. *Video*: Given the limited length of the video, you only need to demonstrate the most important features of your game. A feature can be important for the user because it is a basic operation or it greatly influences the user's playing 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 its 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. *Deadline*: Nov. 18, 2022.

5. **A presentation of the group work (4 points)**
   Each group will also need to present its work before the whole class in the last week of the semester. Details about the requirements for the slides and the organization of the presentations will be announced later.
   The presentation slides should be submitted before or on Nov. 21, 2022.

Appendix A. Grading Criteria

1 mark is for totally disagree, while 5 marks are for totally agree.

Requirements Document	1	2	3	4	5
The document exhibits a clear structure, and its contents are complete and pertinent.	☐	☐	☐	☐	☐
The text reads well (i.e., it is concise and grammatically correct).	☐	☐	☐	☐	☐
Requirements are valid and verifiable.	☐	☐	☐	☐	☐
Requirements are consistent and precise.	☐	☐	☐	☐	☐
Requirements are complete (i.e., no important requirements are missing).	☐	☐	☐	☐	☐
API Design Document					
The document exhibits a clear structure, and its contents are complete and pertinent.	☐	☐	☐	☐	☐
The text reads well (i.e., it is concise and grammatically correct).	☐	☐	☐	☐	☐
The code components are clearly defined and consistent.	☐	☐	☐	☐	☐
The design satisfies the requirements and supports the goals.	☐	☐	☐	☐	☐
The design decisions are reasonable or justified.	☐	☐	☐	☐	☐
Unit Test Suite					
The tests are indeed unit tests.					
The tests cover all the relevant APIs defined in the game model.	☐	☐	☐	☐	☐
The tests match with the APIs and represent typical usage of the APIs.	☐	☐	☐	☐	☐
The purposes of the tests are properly documented.	☐	☐	☐	☐	☐
The changes to the API design, if any, are reasonable and justified.	☐	☐	☐	☐	☐
Implementation					
The document exhibits a clear structure, and its contents are complete and pertinent.	☐	☐	☐	☐	☐
The texts read well (i.e., they are concise and grammatically correct).	☐	☐	☐	☐	☐
The implementation satisfies the requirements and makes a usable software.	☐	☐	☐	☐	☐
The code is in good style.	☐	☐	☐	☐	☐
The updated test suite covers 10/30/50/70/90 percentage of the source code in the software model.	☐	☐	☐	☐	☐
The changes to the API design and test cases, if any, are reasonable and justified.	☐	☐	☐	☐	☐
Presentation					
The contents are well-structured, sufficient, and technically correct.	☐	☐	☐	☐	☐
The presentation is delivered in a good style (English, voice, manner, etc.).	☐	☐	☐	☐	☐
The presentation is properly timed.	☐	☐	☐	☐	☐

Individual grading. In the project, each group member is expected to actively participate and make fair contributions. In general, members of a group 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 own contributions to the project, and each member will be asked to provide evidence for his/her contributions to facilitate the grading.

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.

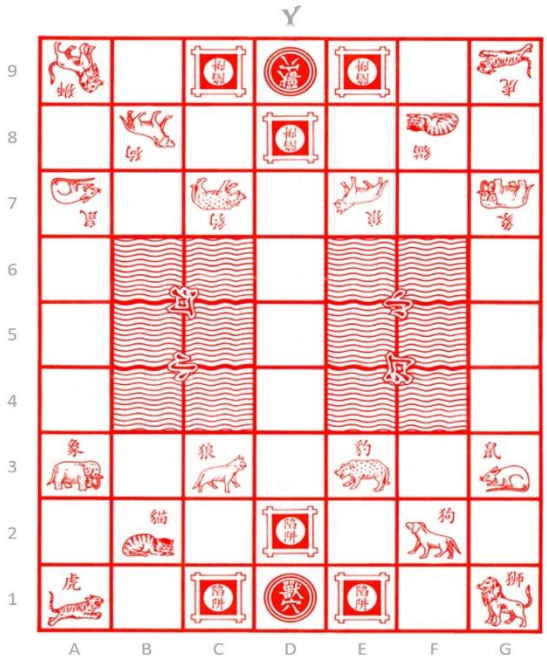

Figure 1: A typical Jungle gameboard showing the locations of starting squares, the den, rivers, and traps.

Figure 2: The den highlighted in green.

Figure 3: The traps highlighted in yellow.

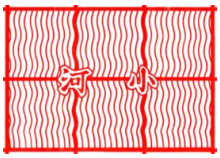

Figure 4: One of the rivers.

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.