Course Project Description

In this project, each group needs to apply the knowledge learned in the course to develop a command-line-based Monopoly 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 13:59, 27 September 2024 (Friday). 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 13:59, 4 October 2024 (Friday).

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 capability 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 13:59, 22 November 2024 (Friday). 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 PIM 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 or on 25 November 2024.** Requests sent afterward will not be entertained.

Use of GenAI tools. Using GenAI tools in the project is permitted, but you should adequately acknowledge how those tools were used. If you use GenAI tools in preparing any of the deliverables, you need to clearly state in a separate document 1) which deliverable(s) and which part(s) of the deliverable(s) were prepared with the help of GenAI tools, 2) how you used the GenAI tools, and 3) what you did to verify/revise/improve the results produced by the tools. Without such a document in your submission, we will consider all the deliverables as your original work. Failure to reference externally sourced, non-original work can be considered plagiarism. You may refer to https://libguides.lb.polyu.edu.hk/academic-integrity/GenAI_and_Plagiarism for more guidance on GenAI and plagiarism.

Appendix B. The Monopoly Game

The game is played with a board divided into 20 squares (as shown in Figure 1) and a pair of four-sided (tetrahedral) dice, and it can accommodate two to six players.

It works as follows:

- Players have money and can own properties. Each player starts with HKD 1500 and no property.
- All players start from the first square.
- Players take turns rolling the dice and advancing their respective tokens clockwise on the board. After reaching square 20, a token moves to square 1 again.
- Certain squares take effect on a player (see below) when her token passes or lands on the square. For example, they can change the player's amount of money.
- If, after taking a turn, a player has a negative amount of money, she retires from the game. All her properties become unowned.
- A round consists of all players taking their turns once.
- The game ends either if only one player is left or after 100 rounds. The winner is the player with the most money at the end of the game. Ties (multiple winners) are possible.

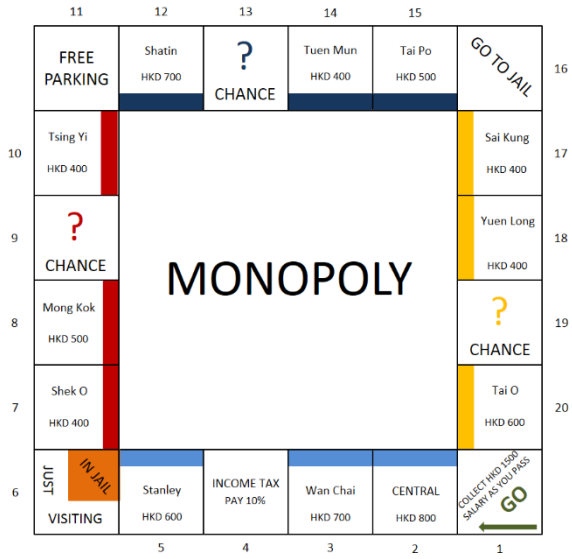

Figure 1. Monopoly board

There are seven kinds of squares on the board:

- **Property squares (marked by a colored stripe).** They contain the property's name and price and can be owned by players. If a player lands on an unowned property, she can choose to buy it for the written price or do nothing. If a player lands on a property owned by another player, she has to pay rent (rent amounts are listed in Table 1).
- **Go.** Every time a player passes through (not necessarily lands on) this square, she gets a salary of HKD1500.
- **Chance.** If a player lands on one of these squares, she either gains a random amount (multiple of 10) up to HKD200 or loses a random amount (multiple of 10) up to HKD300.
- **Income tax.** If a player lands on this square, she pays 10% of her money (rounded down to a multiple of 10) as tax.
- **Free parking.** This square has no effect.
- **Go to Jail.** If a player lands on this square, she immediately goes to the "**In Jail**" part of the "**In Jail/Just Visiting**" square.

Table 1: Properties

Pos	Name	Price	Rent
2	Central	800	90
3	Wan Chai	700	65
5	Stanley	600	60
7	Shek O	400	10
8	Mong Kok	500	40
10	Tsing Yi	400	15
12	Shatin	700	75
14	Tuen Mun	400	20
15	Tai Po	500	25
17	Sai Kung	400	10
18	Yuen Long	400	25
20	Tai O	600	25

In Jail/Just Visiting. If a player lands on this square, she is "**Just Visiting**": the square has no effect. However, if the player gets here by landing on "Go to Jail," she is in jail and cannot make a move. A player gets out of jail by either throwing doubles (i.e., both dice coming out the same face up) on any of her next three turns (if she succeeds in doing this, she immediately moves forward by the number of spaces shown by her doubles throw) or paying a fine of HKD 150 before she rolls the dice on either of her next two turns. If the player does not throw doubles by her third turn, she must pay the HKD 150 fine. She then gets out of jail and immediately moves forward the number of spaces shown by her throw.

Appendix C. User Stories for The Monopoly Game

US1. As a player, I want to start a new game with an existing gameboard.
US2. As a player, I want to name the players with input or randomly generated strings to better differentiate the players.
US3. As a player, I want to see the status of any specific player and all players.
US4. As a player, I want to see the status of the game, including the squares and the players' positions on the gameboard.
US5. As a player, I want to query the next player.
US6. As a player, I want to save the current game to a file.
US7. As a player, I want to load a game from a file and continue the game.
US8. As a gameboard designer, I want to design a new gameboard by creating and organizing squares of the seven types.
US9. As a gameboard designer, I want to load an existing gameboard and customize it by modifying its squares.
US10. As a gameboard designer, I want to save the gameboard I designed.