Course Project Description

Form your groups of 3 to 4 students on Blackboard before or on Oct. 5th, 2021. Afterwards, 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 Oct. 13th, 2021. No group change will be allowed after Oct. 13th, 2021.

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

You may program the game in either Java or Python. If you want to use another programming language, consult your instructor first. You are suggested to design and implement the game in an object-oriented way. You may opt not to do that but keep in your mind that desirable code characteristics like high cohesion and low coupling are more difficult to achieve in, e.g., procedure-oriented programming than in object-oriented programming.

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 document for the game. (8 points)
   Both functional and non-functional system requirements that you feel are important for the game should be clearly specified in the document. Note that, you may decide 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.
   See the slides of Lecture 04 Requirements Engineering for the overall structure of a requirements document and some example requirements written in natural language. (Deadline: Oct. 13, 2021)

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. Note that you should only specify the public and protected members of classes in your OO design. Also note that a method declaration contains the method return type, the method name, method argument names and types, possible exception declarations, and an empty method body. If a method return type is non-void, its body may also return a literal value of appropriate type so that the method declaration would compile successfully. (Deadline: Oct. 22, 2021)

3. A suite of unit tests for the game model [a)] 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. Since all the unit tests should be automatically executable, you are strongly recommended to prepare the tests using unit testing frameworks (e.g., JUnit if your use Java). 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).
   If you need to revise your API design when preparing the tests, you should include in the deliverable a separate document where you explain the revisions you made and the reasons for the revisions. (Deadline: Nov. 3, 2021)

4. An implementation of the game. (9 points)
   The deliverable should include 1) the source code, 2) the final suite of unit tests and

[a)] This implies that you need to adopt the MVC pattern (https://en.wikipedia.org/wiki/Model-view-controller) when designing your game.

its coverage of the code in the software 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.
If you need to revise your API design or tests during the implementation, you should include in the deliverable 6) a separate document where you explain the revisions and the reasons for them and discuss what you could have done differently to avoid such revisions. (Deadline: Nov. 18, 2021)

Besides, each group also needs to present its work before the whole class in the last week of the semester (4 points). The presentation slides should be submitted before or on Nov. 22, 2021. Details about the requirements for the slides and the organization of the presentations will be announced later.

Appendix A. Grading Criteria

Grade 1 is for totally disagree, while grade 5 is for totally agree.

Requirements Document	1	2	3	4	5
The document exhibits a clear structure and contains all needed and relevant information.	☐	☐	☐	☐	☐
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 contains all needed and relevant information.	☐	☐	☐	☐	☐
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 test suite covers the most important aspects of the software.	☐	☐	☐	☐	☐
The test suite matches with the API design.	☐	☐	☐	☐	☐
The test cases are indeed unit test cases.	☐	☐	☐	☐	☐
The purposes of the test cases are properly documented.	☐	☐	☐	☐	☐
The tests represent typical usage of the software.	☐	☐	☐	☐	☐
The changes to the API design, if any, are reasonable and justified.	☐	☐	☐	☐	☐
Implementation					
The documents are well structured and contain all needed and relevant information.	☐	☐	☐	☐	☐
The texts read well (i.e., they are concise and grammatically correct).	☐	☐	☐	☐	☐
The implementation satisfies the requirements and makes a reliable, usable software.	☐	☐	☐	☐	☐
The code is in good style.	☐	☐	☐	☐	☐
The updated test suite covers 10/30/50/70/90 percentage of the code lines in the software model [b)].	☐	☐	☐	☐	☐
The changes to the API design and test cases 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.	☐	☐	☐	☐	☐

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. Besides playing the game, Players should also be able to save and load a game.

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 ("**Go**").
- Players take turns in 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.

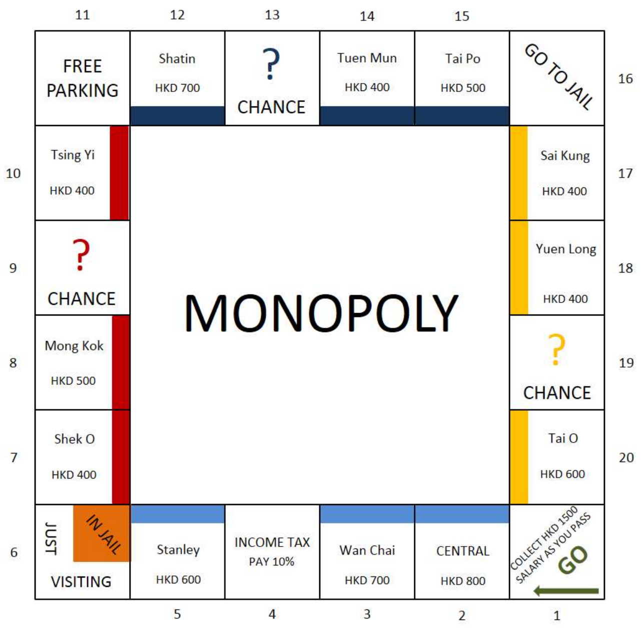

Figure 1. Monopoly board

- The game ends either if there is only one player 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.

There are seven kinds of squares on the board:

- **Property squares (marked by a colored stripe).** They contain the name and the price of the property 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 a rent (rent amounts are listed in Table 1).
- **Go.** Every time a player passes through (not necessarily lands on) this square, she gets HKD 1500 salary.
- **Chance.** If a player lands on one of these squares, she either gains a random amount (multiple of 10) up to HKD 200 or loses a random amount (multiple of 10) up to HKD 300.
- **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.
- **In Jail/Just Visiting.** If a player lands on this square, she is "**Just Visiting**": the square has no effect. However, if the player got here by landing on "**Go to Jail**", she is in jail and

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

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.