**THE HANG SENG UNIVERSITY OF HONG KONG**
**DEPARTMENT OF COMPUTING**
**COM3101 Project Description**

# Mini-Monopoly

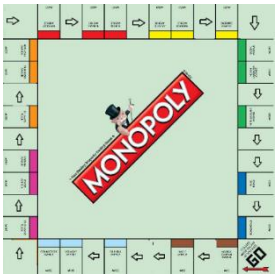
**Important Dates**

- **Project Documentation (Draft): April 10th, 23:59**
- **Live Demonstration: Week 14 (MS Teams)**
- **Project Documentation (Final): April 30th, 23:59**
- **Demo video: April 30th, 23:59**

## Project Team

Work in teams of two or three students.

## A Simplified Monopoly Game




In this project, you will design and develop a mini version of the famous Monopoly board game.

## The Basic Rules

The game is played by four players using a playing board. A possible board is shown in the above picture.


1. There are 44 spots placed in various parts of the board. Twenty-two of these spots are coloured and each one represents a piece of land that can be bought and sold. Each piece of land has a unique *slot number* (i.e., 1 to 22), a *slot name* and a *land price*. However, the slot names and the land prices are not always fixed and may vary from version to version. For example, in one version, the last spot (slot 22) has the slot name "Repulse Bay" and the land price is $3500.
2. Each piece of land has ownership. Initially, all lands are owned by no one. (For simplicity, we assume all other uncoloured slots on the board are skipped in this simplified game).
3. The players start the game on the GO position (slot 0) on the bottom right corner of the board. After that, each player takes turn to roll a dice and moves along the board.
4. Each player collects some starting money ($2,000) at the beginning of the game.
5. When a player arrives at a slot that is not owned by other players. He has the option to buy it. To buy the land, he pays (i.e., returning) the amount of money that is the same as the land price. After the payment, the ownership of that piece of land is transferred to the owner.

6. Every time a player goes pass the GO position again (after travelling around the board once), he receives a GO-money of $2000.

7. When a player lands in a slot that is already owned by another player. He needs to pay a rental fee to the owner. In this simplified game, the rental fee is fixed at 10% of the land price.
8. If a player cannot play a rental fee, he must declare bankruptcy. All his lands will be returned (ownership reset to "owned by no one"). He must then stop playing the game.
9. The last remaining player wins the game.

**The Advanced Rules (Optional)**

In addition to the basic game play, some advanced players also have the following rule:

10. A player may buy or sell a piece of land from/to another player, provided that both parties agree to trade. The price of such transactions is not fixed, and can be negotiated by the seller and the buyer. Once an agreement is reached, the buyer pays the agreed amount of money to the seller and the land's ownership is transferred from the seller to the owner.

There are no other rules in this simplified version of the game, that is, no houses, no hotels, no tax, no chance, no railway stations, no jail (yeah!).

**What you need to implement**

Traditionally, the game is played on a game board with toy money. In this project, you need to implement a computer version of it as follows.

Handling of the Basic Rules

As a basic requirement, you need to implement the basic rules as described above. Note the following, however.

1. The software will be run on stand-alone Windows platforms using Java.
2. The software should use a data file for storing the slot names and land-price information. The following are some sample data.

| Slot | Slot name | Land price |
|---|---|---|
| 0 | Go | - |
| 1 | Ping Chau | $1000 |
| 2 | Tai O | $1200 |
| … | … | … |
| 22 | Repulse Bay | $3500 |

You are free to decide its file format and how it can be edited (e.g., provide a program function or simply rely on external applications for editing).

3. There are 23 slots in the game. The first one (slot 0) is the Go Slot, which has no land value and cannot be bought.
4. The software is used by all four players during game play. Each player may use it when it is his/her turn to play.
5. All player starts at slot 0 (the Go Slot)
6. At any time, the program should keep track of the following information:
   i. The amount of money currently owned by each player (i.e., his/her balance)
   ii. The status of each player (i.e., active or bankrupt)
   iii. The ownership of each piece of land
   iv. The position of each player
   v. Who's turn is it to move

7. When it is a player's turn to move, he may start moving by using a "roll dice" function. The function will generate a random number (*r*) between 1 and 10, and the player will go forward by *r* slots accordingly.

8. After a player passes the last slot (slot 22), he will return to slot 0 and so on.

9. A player can receive a GO money of $2000 each time he goes pass slot 0.

10. If a player lands in an un-occupied slot, he will be given an option to buy the land. If he chooses to do so, the amount of money corresponding to the land price is deducted from his balance. The balance cannot be negative, however. If the transaction is successful, the land ownership will be changed to the player.

11. If a player lands in a slot owned by another player, he needs to pay the owner a rental fee (10% of the land price). The balance of the two players will be updated accordingly. If the payer's balance is less than the rental fee, the payer's status will be changed to "bankrupt" and he must skip all his future turns. The remaining balance will of the player will be transferred to the landowner. His lands will be reset as unoccupied.

12. At any time, the program should clearly display the following game data.
    - The position of each player
    - The balance or each player
    - The status of each player
    - Who's turn is it to move

    Note: for a start, you do not need to follow the original game board design in your UI. Instead, you may choose any reasonable method to show the above data, with or without using a game board.

13. You should also provide a function for showing which land is owned by which user (e.g., upon requests by the player). Again, you do not need to follow the method used in the original Monopoly game. Instead, you may use any reasonable method to show the information.

A Game Editor Function (For Testing)

- To facilitate testing (Unit Test, SIT and UAT), please also implement a *Game Editor* function that allows a tester to modify the game status at any time. Using this function, the tester may:
    - Modify the ownership of a land slot
    - Modify the balance of a player
    - Modify the location of a player
    - Modify the status of a player (active or bankrupt)
    - Modify who's turn is it to move.
- Both the internal representation and the UI should be changed accordingly.
- This function is a temporary one to support testing. It will be hidden (unless the user knows the secret) before the software is released in the future (not part of this project).

Handling of the Advanced Rules (10% bonus)

You may also implement the advanced rule if you have the time:

14. When it is a player's turn to move, and before he uses the roll-dice function to advance his position, the player may use a *trade* function to buy/sell land from/to another player.
15. After selecting the *trade* function, the player needs to indicate the operation type (buy or sell), the slot number of the land he is interested to buy or sell, and the agreed amount. If the transaction is successful, the agreed amount will be transferred from the buyer to the seller and the land ownership would be changed to the buyer. (You can assume that the buyer and the seller have already agreed to trade before using this function.)

## Project Documentation

### Part A: Requirement Analysis

**(a) Use case diagram**

Identify the actors and the goals from the above descriptions and draw a use case diagram like the ones in the lecture notes accordingly. (Hints, there should be more than one actor if you include the user of the Game Editor function.)

**(b) Use case descriptions**

For each of the goal you have shown in the use case diagram, write a use case description like the ones given in the lecture notes. For each one, you need to state the

- Use case name
- The actor
- The case details: i.e., what actions are expected from the actors and what are the processing and what responses are expected from the system.

### Part B: System Design

Create the **class diagrams** for the Monopoly Bank System, including:

- Major classes
- Major attributes and their data type
- Major methods (getters and setters and constructors can be omitted for simplicity)
- The associations between the classes (e.g., one to one, one to many, etc.)

**Project Documentation (Draft Version) Submission: 10th April 2022, 23.59**

## Implementation

**Intermediate check point: week 12 (Note: not needed to be handled in. Not for scoring)**

It is suggested that you should have completed the following by end of this week:

- Loading of data file.
- Internal representation of game status, e.g.
    - Land information (name, price, and ownership)
    - Position of each player
    - Who's turn is it to move.
- UI for showing the above
- The Roll Dice function, and the advancement of the player's location.
- UI indication of the location of each player.
- The Game Editor Function.

**Live Demonstration: (week 14)**

Demonstrate your completed project to the instructor on MS Teams.

**Final Submission: 30th April 2022, 23.59 (Sat, week 14)**

- Your program files (source code and data files)
- A short demonstration video (around 3 minutes). Upload it online and submit a link to the video. (Make sure you grant me access right if it is on Google Drive).
- Project status report (will be available on Moodle)
- Project Documentation (Final Version)
- (Optional) If necessary, prepare a user guide on how to setup/run your application, and how to use it.

## Note

You should, as far as possible, follow the practices you have learned in this module. This includes:

- MVC Design pattern
- Object-oriented design principle
- The good coding practices

## Mark Distribution

Project documentation: draft version 10%, final version 20%

Project file submissions and demonstration (live and video): 70%

**Have fun, and good luck!**