SWEN30006 Software Modelling and Design
Project 2: Lucky Thirteen
- Project Specification -

School of Computing and Information Systems
University of Melbourne
Semester 1, 2024

1 Background

The JQK Computer Game Company (JQK) observes a steep increase in a popularity of card games. The company foresees a bright future of digital card games and aims at exploring this opportunity. While there exist many card games already, the company decides to create yet another card game called **Lucky Thirteen**.

A software team at JQK has started to develop a framework for this new digital card game. The code does work to some extent. However, it was implemented in a rush and was not properly designed. JQK has recruited you and your team to revise the existing design and codebase as well as extending the framework to also include additional logic and features to make the game even more exciting.

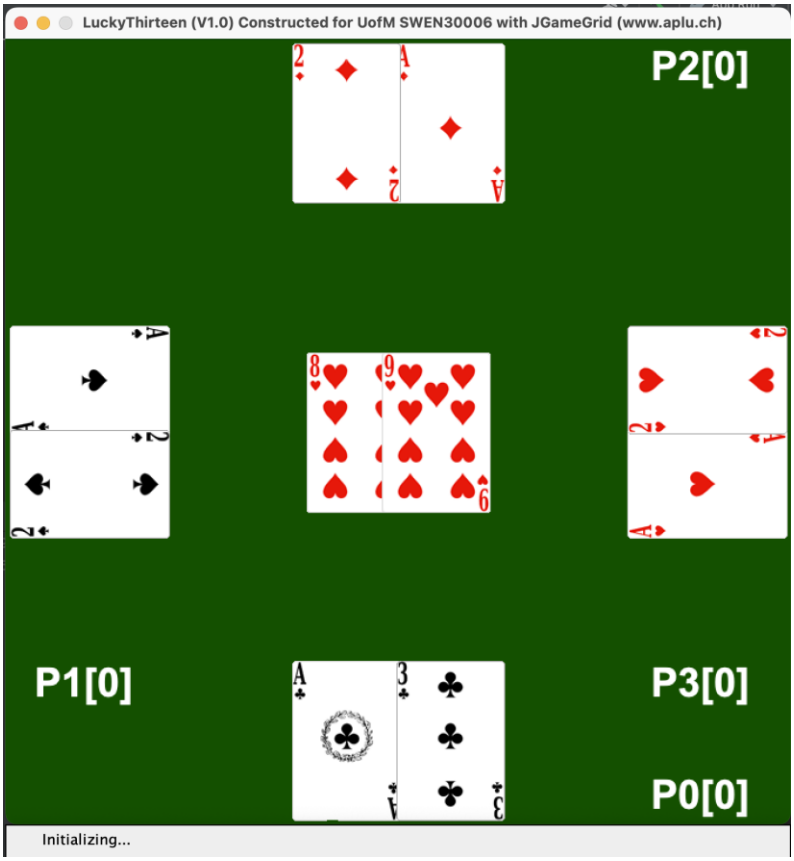

Figure 1: The GUI of LT game

2 Rules for the Lucky Thirteen Games

The Lucky Thirteen (LT) game play is briefly described below. The current program developed by JQK already supports this behaviour.

- The game is played with a standard 52-card deck with four *suits*, each having thirteen *ranks*.
- Each rank is associated with a numerable value where
  - For summing during the game: Number cards represent as is (i.e., 2 is two). For picture cards, 'A' can represent either zero or one, while 'J', 'Q', 'K' can be 10, 11, 12, or 13.
  - For scoring after the game: Number cards score according to its numerable value (i.e., 2 is worth two points). For picture cards, 'A', 'J', 'Q', 'K' score 1, 11, 12, 13 points respectively.

  (See Table 1 for a summary)
- The goal of a player is to have the sum of the value of the cards to thirteen (see below for more details about the summation).
- The game involves four *players* who play independently.
- All cards are dealt face down. Note that the software version deals the cards face up to make it easy for developers to see what is going on. However, computer players should not be able to "see" the cards of the other players.
- At the start of the game, all 52 cards are shuffled. Two cards are dealt from the pile and put face up (every player can see these cards). These two cards are referred to as ***public cards***. Then, two cards are distributed to each player (players cannot see each other cards), these cards are called ***private cards***. A total of ten cards are dealt from the pile at this step.
- The game consists of four *rounds*. Each round, each player deals one card from the pile and discards one card of their choice. Note that a player can choose to discard any of the three private cards in their hand, and every player can see the discarded card.
- Player 0 always starts first. The game then proceeds clockwise.
- At the end of the fourth round, each player sums the value of the private cards (regardless of the suit) in their hand or combines with the public card(s) dealt at the beginning of the game to achieve the summation of thirteen. The summation can only be one of the following two scoring options (see also Table 2):
  - **Option1:** Two private cards in their hand.
  - **Option2:** One private card in their hand and one card from the two public cards.

  Note that the game automatically checks these two options.
- There are three cases to assign scores to a player. Note that each case considers both scoring options mentioned above.
  - **Case1:** If there is only one player who achieves the sum of exactly thirteen, that player wins the game and gain a score of 100 points. In other words, only one player receives a score. The other three players' score remains zero.
  - **Case2:** If no player has the sum of thirteen, each player receives a score of the sum of the two private cards in their hand according to the following calculation:
    - Each suit corresponds to a multiplication factor as follows: spade (♠) x 4, heart (♥) x 3, diamond (♦) x 2, club (♣) x 1.
    - A number card score as is. Picture cards score as follows: A = 1, J = 11, Q = 12, K = 13.
    - Example 1: A player with 8-heart and 5-club would receive a score of 8x3 + 5x1 = 29 points.

  Note that in this case, all players receive a score.

- **Case3:** If more than one player achieves the sum of thirteen, those players receive a score from the cards that add up to thirteen according to the score calculation above. If the public card is used, the multiplication factor is always 2 regardless of the suit. If a player can sum the cards in different ways to thirteen, the calculation should maximize the score (see Example 3a below).
  - Example 2: The score of 7-club (in hand) together with 6-club (from public cards) is 7x1 + 6x2 = 19.
  - Example 3a: A player has J-diamond and A-heart. The public cards are 5-spade and 2-diamond.
    - [Option1] Two cards in hand:
      - Summing: J as 12 and A as 1 -> 12 + 1 = 13
      - Scoring: J is 11 and A is 1 -> 11x2 + 1x3 = 25
    - [Option2] One card in hand and one public card:
      - Summing: J as 11 -> 11 + 2 = 13
      - Scoring: J is 11 -> 11x2 + 2x2 = 26
    - The player earns the score of 26 since it is the highest.

  Note that in this case only the player(s) whose sum is thirteen may receive scores. The score of the other players remains zero.
- The winner and the score obtained are displayed on the screen.

The LT program currently supports two types of players: *human* interactive player who selects the card to play through a double-left-mouse-click, and *random* computer player who selects a card to discard at random. The current configuration is always one human at position 0 and three random players at the other positions.

	Number card (2, ..., 10)	**Picture card 'A'**	**Picture cards 'J', 'Q', 'K'**
Summing	Value as is	can be 0 or 1	can be 10, 11, 12, or 13
Scoring	Value as is	worth 1	J is worth 11 Q is worth 12 K is worth 13

Table 1: Summing and scoring for each card

Public cards: A♠ and 2♦ **In-hand cards:** J♦ and A♥		**Summing**	**Scoring**	
Current support	Two cards in hand	J as 12 A as 1	J is 11 and A is 1	11x2 + 1x3 = 25
	One card in hand and one public card	J as 11 (2 is 2)	J is 11 and 2 is 2	11x2 + 2x2 = 26
Additional feature (your task)	Two cards in hand and two public cards	J as 11, both A's as 0 (2 is 2)	J is 11, both A's is 1, and 2 is 2	11x2 + 1x3 + 1x2 + 2x2 = 31

Table 2: Examples of three options to sum the cards to thirteen

3 Your Task

You task is to improve the design of LT game, refactor the existing codebase, and implement additional logic/features as follows:

1) Ensure that all player types (see below) follow the rules of LT listed above.
2) Add another option to sum the card to thirteen, namely,
   Option3: Two cards in their hand and two cards from the two public cards. Note that there is now a total of three options to sum the card to thirteen.
   Example 4: The score for 2-diamond and 4-heart (in hand) together with 5-spade and 2-club (from public cards) is 2x2 + 4x3 + 5x2 + 2x2 = 30.
3) If more than one player achieves the sum of thirteen (Case3), the score calculation must consider all three options to sum the cards to thirteen and a player will receive the maximum among those three (see also Table 2).
   Example 3b: A player has J-diamond and A-heart. The public cards are A-spade and 2-diamond.
   - [Option1] Two cards in hand:
     - Summing: J as 12 and A as 1 -> 12 + 1 = 13
     - Scoring: J is 11 and A is 1 -> 11x2 + 1x3 = 25
   - [Option2] One card in hand and one public card:
     - Summing: J as 11 -> 11 + 2 = 13
     - Scoring: J is 11 -> 11x2 + 2x2 = 26
   - [Option3] Two cards in hand and two public cards:
     - Summing: J as 11 and both A's as 0 -> 11 + 0 + 0 + 2 = 13
     - Scoring: J is 11 and both A's is 1 -> 11x2 + 1x3 + 1x2 + 2x2 = 31
   - The player earns the score of 31 since it is the highest.
4) Support four player types: human, random, basic, clever
   a) **Human** player's behaviour should not change from that supported by the current framework.
   b) **Random** computer player's behaviour should not change from that supported by the current framework.
   c) **Basic** computer player discards a card with the lowest value where A (=0) < 1 < 2 < 3 < ... < 10 < J (=11) < Q (=12) < K (=13) and club (=x1) < diamond (=x2) < heart (=x3) < spade (=x4). This means that if the basic player has 6-spade and 7-diamond, the player will discard 7-diamond since 7x3 = 21 < 6x4 = 24.
   d) **Clever** computer player must keep track of all the cards played and use this information together with the knowledge of the cards in its hand to try to maximize its score. The clever player must produce cleverer play than the basic and random players. (You will have to explain how your design supports the clever play.)
5) Load appropriate game parameters from the property files. You must support configuring the player types in the property file using identification of the form "players.0=clever" for the full set of player types {human, random, basic, clever} and player positions {players.0, players.1, players.2, players.3}. All properties you add must have default provided in the code (see setProperty() in main()).

4 Provided Package

4.1 Getting started

- The package is provided as an IntelliJ project in a GitHub repository: https://classroom.github.com/a/CZL3V7l2
  To access the base package and set up, you can follow the instructions in Workshop 0. Note that the project requires at least Java 19.
- Build and run the project: You will see the GUI appear and can play the current version of Lucky Thirteen using the mouse actions.

4.2 System design

- The classes in the provided 'app/src/main/java' package primarily handle the game behaviour where the GUI operates by using the JGameGrid library. You can refer to the classes and their functions in this framework in Java Doc.
  - The required changes for this project relate only to the game behaviour. You ***should not*** need to modify any code pertaining to the GUI.
  - Implementation hints: The JGameGrid library has provided many useful functions that you can reuse to implement the proposed extensions. The codebase should also provide examples for implementing the extensions.
- The system reads a properties file that specifies the player type for each of the four players, auto-play configuration (for testing purposes) and some of the cards the player will play in auto-play mode. You can edit the properties in these files for your own testing but make sure to preserve the default behaviour.
- The 'logResult' string in the LuckyThirteen class is for testing your system. Please see Section 4.3 Testing Your Solution for more details. Image files of characters and items are provided in the 'sprites' folder (app/src/main/resources/spites). The file names of these image files must not be changed.

4.3 Testing Your Solution

- We will be testing your application programmatically, so we need to be able to build and run your program with a gradle build script. This is the same structure as the Workshop 0. You need to follow the following guideline for gradle build to work:
  - You should use the gradle script that we have set upvalid for you to run the project.
  - The entry point must remain at Driver and the Driver file needs to remain at `src/main/java`.
  - Whenever you push to Github, Github should automatically build and run the test in your code with the test cases already provided by us in the project. You are not required to run the test in your local machine. When you are developing the app, please ensure that your code passes the testing when pushed to Github.
  - Initially, your code will pass test case Original and fail test cases Extension. You need to implement the requirement and pass all test cases. Your code needs to pass the automatic tests as well as the manual tests that teaching staff will perform on your final submission. For manual tests, we will run the app as a human player to play against random, basic and clever players.
  - We have provided the sample log that the project needs to generate to pass the test case. Your log needs to follow the format but does not need to match the sample log 100%.

- You must not change anything in the test directory as we will overwrite all your changes with our default values when testing is executed. You must also not change the build.gradle and settings.gradle file as we will. It means that any other keys or values that you add into your properties files in the app directory will not work in testing.
- Four properties files are provided as samples for development (gameX.properties) and five properties files are provided as the key resources for testing (testX.properties).
- The auto mode may control some player for some card selections, and then it may stop after a few turns or a few rounds. When the auto mode stops the card selection of a specific player, your program will need to take over and selects the cards for the player based on the player type (random, basic or clever).
- You need to ensure that any modification still preserves the original behaviour, i.e., logResult generated by your extended version can pass the testing that we provide.
- For your understanding, below are some key gradle build explanations. Note that you are not expected to change anything outside of your development directory and documentation directory:
    - `settings.gradle`: to set up the name of the project and the main development directory
    - `app/src`: where the development and testing code is
    - `app/src/build.gradle`:set up the dependencies, the language version and the main class
    - `app/src/main/java`: where your development code is
    - `app/src/main/resources`: where your properties files and images are
    - `app/src/test/java`: where the testing code is
    - `app/src/test/resources`: where the properties files and images are
    - To load a property or text file: you need to use `CLASS_NAME.class.getClassLoader().getResourceAsStream(propertiesFile)`. This is already done for you, so you only need to be aware of it if you need to read any other text file.

5 Project Deliverables

5.1 Your version of the LT game

Submit your whole project (with any/all libraries used included, with all the gradle structure and gradle setting files)

- Ensure that your code is well documented and includes your team's name and team members in all changed or new source code files.

5.2 Report

A design analysis report detailing the changes made to the project and the design analysis including identifying design alternatives and justifying your design decisions with respect to design patterns and principles (GoF and GRASP). The report must also include your strategy for Clever computer player.

5.3 Software Models

To facilitate the discussion of your design, the following software models should be provided in the report and used along with the discussion. You can use sub-diagrams or provide additional diagrams where appropriate.

- A domain class diagram for capturing the covering the domain concepts relating to computer players
- A static design model (i.e., a design class diagram) for documenting your design relating to computer players
- Additional static and/or dynamic design models to support your explanation as you judge useful and appropriate.

6 Submission

All project deliverables should be included in the <u>one</u> project zip file with the specified structure (see Figure 2). **<u>Do not</u>** include the .git folder in your submission. **<u>Only one member</u>** in the team submits the file. **See more details about submission and evaluation on the project submission page.**

Note: It is your team's responsibility to ensure that you have thoroughly tested their software before submission.

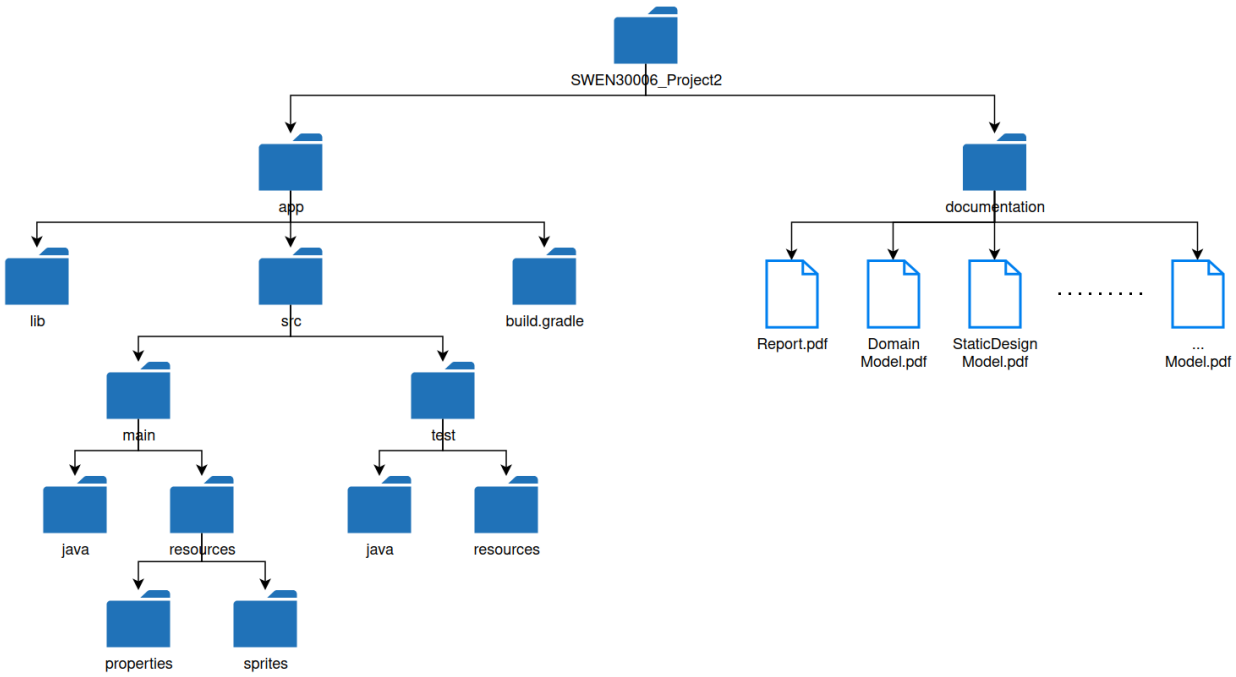

Figure 2: File structure for project submission