SWEN30006 Software Modelling and Design
Project 2: Whist

School of Computing and Information Systems
University of Melbourne
Semester 1, 2020

Overview
You and your team of independent Software Contractors have been hired by *New, Exciting and Revolutionary Designer Games* (aka *NERD Games*) to provide some much-needed assistance in developing their innovative *Whist* card game to be market ready.

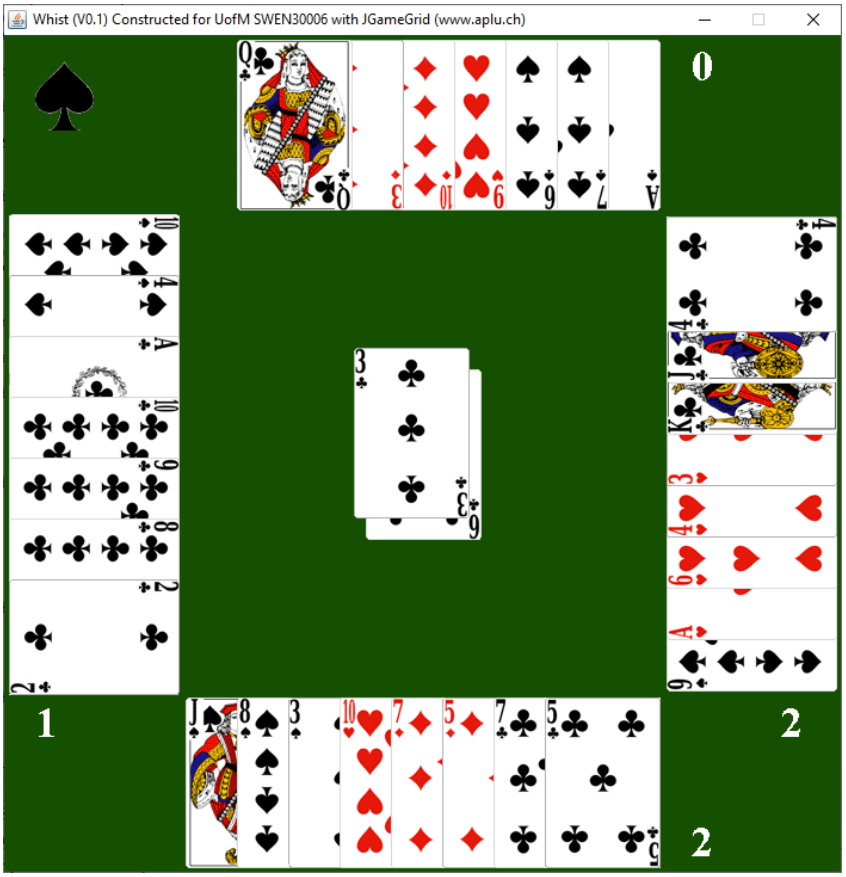

Figure 1: Advanced Whist CGI

NERD Games have a fully running version of the game, but it is limited in terms of configurability and the available NPCs (Non-Playable Cardplayers).

Your task is simple. Modify the design and implementation to improve configurability, to add new NPCs, and to facilitate the addition of other future changes.

The Whist game play is briefly described below. Note that some of the elements listed below as specific values are configurable.

1. Whist is played with a standard fifty-two card deck with four *suits* and thirteen *ranks* in each suit.
2. The game involves four *players* who play independently, i.e. there are no teams. The first player to score twenty points wins.
3. The game starts with a *hand* of thirteen cards being *dealt* to each player, a *trump* suit being randomly selected (displayed in the upper left of Figure 1), and a player being randomly selected to start or *lead*. The game play then proceeds as follows:
    a. The player taking the *lead* can play any card they wish from their hand to the centre; this card provides the basis for a *trick*.
    b. Play proceeds clockwise from the lead with each player playing one card in turn to *follow* the lead.
    c. Following players must play a card of the same suit as that lead if they have one. If not, they may play any card they wish.
    d. Once every player has played one card, the winner is the player who has played the highest card of the trump suit if any, or the highest card of the lead suit if not.
    e. The winner receives one point for winning the trick, and then leads for the next round starting a new trick. Play ceases as soon as a player has won the game (received twenty points).
4. If the players have played all their cards without anyone winning, play continues exactly as described from (3) starting with new deal.

The Whist program currently supports two types of players: *human* interactive players who select the card to play through a double-left-mouse-click, and one type of NPC, *random* players who select a card to play from their hand randomly without regard for the rules.

Your job is to support the following program and design enhancements:

1. Additional NPC types are required immediately. Others might be added in the future. Note that here "legal" means consistent with the rules.
    a. *Legal*: plays a random selection from all cards that can legally be played.
    b. *Smart*: a player that records all relevant information and makes a reasonable, legal choice based on that information. A Smart player must produce smarter play than a legal player and have the necessary information available in a suitable form so that an extremely good NPC player could be developed based on the Smart player. A Smart player should assume that other players are playing legally.
2. Parameters: currently all elements of Whist are set in the code. The system should be made more configurable through a property file. You will have to make a judgement as to which parameters are worth making configurable.
3. Three property files, all supporting repeatable runs (subject to interactive player choices) with a fixed seed "30006".
    a. "original.property": one interactive player (0) and three random NPCs, and game settings as per the original package.
    b. "legal.property": four legal NPCs, with nbStartCards = 4 and winningScore = 6.
    c. "smart.property: one interactive player (0), one smart NPC (1), and two random NPCs, and game settings as per the original package.

Note that, despite the graphics in Figure 1, an NPC must not be able to see another player's card until they use it to lead or follow in the game play. Player's must not share information directly and must store their own information about the game they are playing, i.e. they must not access a common pool. As well as cards played, a player can see which player played the card, and the scores. Note that inferred information is also useful, for example, when a player doesn't follow suit, that player must have no remaining cards in that suit.

The Sample Package

You have been provided with an Eclipse project export representing the current system. This system runs in a fixed configuration like the property file described above in 3a.

To begin:

1. Import the project zip file as you did for Workshop 1.
2. Make sure that the JGameGrid.jar file is in the Java Build Path for the project (Project > Properties > Java Build Path); if not, re-add it (using Add Jars …).
3. Try running by right clicking on the project and selecting **Run as... Java Application**.
4. You should see a window like that in Figure 1; you can then play the game as the interactive player.

The current system should be used as a starting point. Please carefully study it and ensure that you are confident you understand how it is set up and functions before continuing. You will need to preserve elements of the system in making changes, that is, you will want to refactor parts of the system. However, you are free to make whatever changes you wish, as long as the game play is preserved and still easily recognisable. If you have any questions, please make use of the discussion board, or ask your tutor directly as soon as possible; it is assumed that you will be comfortable with the package provided.

Note: By default, the simulation should run without a fixed seed, meaning the results will be random on every run. To have a consistent test outcome, you need to be able to specify the seed in the configuration file.

Project Deliverables

As well as your updated system, including design and code reflecting your design, you need to provide the following:

- A **report** describing the changes you have made to the system and providing a rationale for these changes **using design patterns and principles**. Note that to do this well, you should include discussion of other reasonable options which you considered, and why you did not choose these options. Around 4-6 pages should be enough to provide a concise rationale. You should strongly consider including and referring to diagrams in your report (as part of the report or separately) to illustrate aspects of the changes.

Note: There are three required tasks here – design, code, and report – which are interrelated. You should work on these tasks **as a team**, not separately. Every team member needs to be able to demonstrate their understanding of and contributions to all deliverables.

Testing Your Solution

We will be running your application manually using the property files and will need to be able to build and run your program without using an integrated development environment. The entry point must remain as “Whist.main()”.

Note: It is **your responsibility** to ensure that you have thoroughly tested your software before you submit it.

Marking Criteria
This project will account for 20 marks out of the total 100 available for this subject (note that scaling to project marks will apply as announced on the LMS). It will be assessed against the following rubric:

- H1 [16-20]: Excellent extendable design and implementation. Clear, consistent, and helpful rationale for design and other changes in terms of patterns. Few if any minor errors or omissions (no major ones).
- H2B-H2A [13-15.5]: Good extendable design and implementation. Generally, clear, consistent, and helpful rationale for design and other changes in terms of patterns. Few errors or omissions (no major ones).
- P-H3 [10-12.5]: Reasonable design and implementation. Fairly clear, consistent, and helpful design rationale with some reference to patterns. Some errors or omissions.
- F+ [5-9.5]: Plausible design and implementation. Plausible attempt at design rationale with some reference to patterns. Includes errors and/or omissions.
- F [0-4.5]: No plausible implementation. Design rationale provided but not particularly plausible/coherent.

Note: We reserve the right to award or deduct marks for clever or very poor code quality on a case by case basis outside of the prescribed marking scheme. Further, we expect to see good variable names, well commented functions, inline comments for complicated code. We also expect good object-oriented design principles and functional decomposition. For UML diagrams you are expected to use UML 2+ notation.

On Plagiarism: We take plagiarism very seriously in this subject. You are not permitted to submit the work of others under your own name. This is a **team** project. More information can be found here: https://academichonesty.unimelb.edu.au/advice.html.

Submission
Detailed submission instructions will be posted on the LMS. You should submit all items shown in the checklist below. You must include your team number in all your pdf submissions, and as a comment in all changed or new source code files provided as part of your submission.

Submission Checklist

1. Your complete updated source code and library package reflecting your new design and implementation (all Java source with top-level folder called “swen30006”, plus all non-standard libraries on which your source depends).
2. Three property files, named as above, for configuring the system as described above.
3. Design Analysis Report (pdf) and any external diagrams (pdf or png).

Note: Only one member of the team to submit.

Submission Date

This project is due at **11:59pm on Friday June 5**. Any late submissions will incur a 1-mark penalty per day unless you have supporting documents. If you have any issues with submission, please email Peter at peter.eze@unimelb.edu.au **before** the submission deadline.