SWEN30006 Software Modelling and Design
Project 2: Oh Heaven

School of Computing and Information Systems
University of Melbourne
Semester 1, 2022

Overview
Off the back of their overwhelming success with Snakes and Ladders on a Plane (sales went into double digits!), NERDI are back to work on their new game. While NERDI have learnt some lessons (sadly, none related to object oriented design), they still need the help of you and your team to assist them in getting their *Oh Heaven* card game market ready.

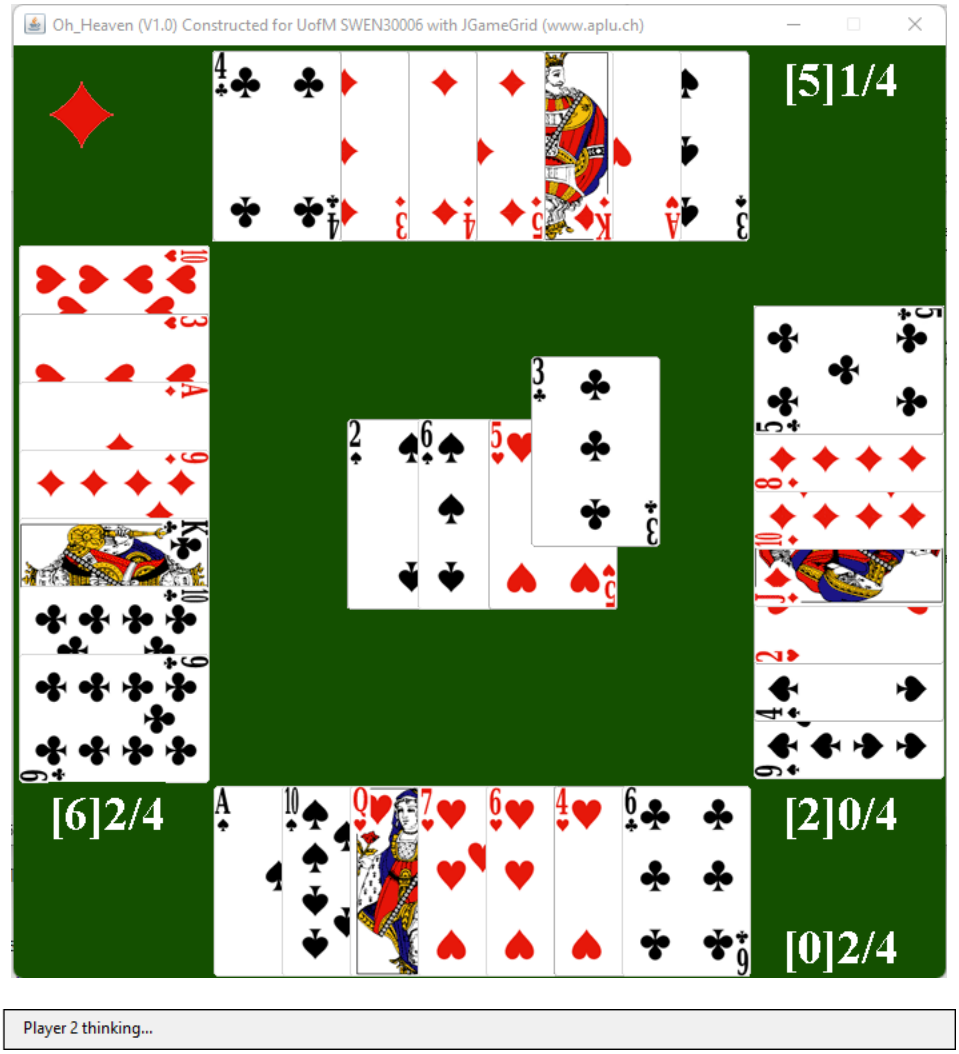

Figure 1: Advanced Oh Heaven CGI

NERDI have a fully running version of the game, but the design is not very extendible or maintainable, it is limited in terms of configurability, and needs more NPC (Non-Playable Cardplayer) options.

The Oh Heaven game play is briefly described below. Note that some of the elements listed below as specific values are configurable. The current program developed by NERDI already supports this behaviour.

1. Oh Heaven 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. You can assume exactly four players. The aim is to get the highest score.
3. The game is made up of three rounds. Each round is made up of the following sequence:
    a. A *hand* of thirteen cards being *dealt* to each player, a *trump* suit is randomly selected (displayed in the upper left of Figure 1) for the round, and a player being randomly selected to bid first and then start or *lead*.
    b. Each player then makes a *bid* as to how many *tricks* they think they can win. They will get 10 extra points if they make exactly this many *tricks*. The total bid cannot equal the thirteen, that is, the last player must bid such that it is not possible for every player to achieve their bid.
    c. The game play then proceeds as follows:
        i. 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*.
        ii. Play proceeds clockwise from the lead with each player playing one card in turn to *follow* the lead.
        iii. 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.
        iv. 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.
        v. The winner will receive one point for winning the trick; if any cards remain, they go back to step i by leading a card to start a new trick.
    b. At the end of the round, any player who made exactly the number of tricks that they bid will receive an additional 10 points.
4. At the end of all rounds, the player (or players) with the highest score wins.

The Oh Heaven 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. However, it always has one human and three random players.

Tasks

Your job is to improve the design of Oh Heaven, and to change it 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 Oh Heaven 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, however you need to at least support all the parameters found in the test property files (see below).
3. There is no requirement to change the bidding behaviour in the code. Your report, however, should comment on how the design could be extended to support NPCs making smarter bids.

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. Players 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, the number of tricks taken by a player, and the scores. Note that inferred information is also useful, for example, it can be assumed when a player doesn't follow suit, that the player must have no remaining cards in that suit.

Property Files

The following three property files (provided) should all support repeatable runs (subject to interactive player choices) with a fixed seed "30006":

1. "original.properties": one interactive player (0) and three random NPCs, and game settings as per the original package. Running with this file should preserve the current behaviour of the system.
2. "legal.properties": four legal NPCs, with nbStartCards = 4.
3. "smart.properties: one interactive player (0), one smart NPC (1), and two legal NPCs, and game settings as per the original package.

There is a further property file "runmode.properties" which should tell the program which of the property files above should be used (in a similar fashion to the way property files work with Snakes and Ladders on a Plane).

Report

Your submission must include a report (pdf) with supporting diagrams (PDF or image).

The report should describe the changes you have made to the system and provide 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 (not including diagrams) should be enough to provide a concise rationale. You must include and refer to diagrams in your report to illustrate aspects of the changes: at least one UML design class diagram and at least one UML sequence diagram should be provided.

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 "Oh_Heaven.main()".

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

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 (Import>Existing Projects into Workspace>Archive File, *not Import>Projects from Folder or Archive*)
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.

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.

Submission Details

Only one team member needs to submit for the team. The team can submit any number of times, however staff will only look at the latest submission for a team.

Your submission should be a zip file (with .zip extension) containing only two folders:

1. "report": this folder should contain your report and any separate diagrams.
2. "src": this folder should contain the java source files for your submission, in whatever subfolder structure you choose.

Do not include sprites, property files, jar files, or version control repositories in your submission.

Submission Date

This project is due at **11:59pm on Friday May 27**. 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 Khang at khang.vo@unimelb.edu.au **before** the submission deadline.

Rubric

This project will account for 20 marks out of the total 100 available for this subject.

The design of the system is our key concern. We want a good extended or extendable design, based on GRASP and GoF Patterns. ***We are not looking for the "best design".*** A good design with a rational justification well based on *design principles and patterns* is sufficient. The design report, the design (implicit in the implementation), and the implementation are all related artefacts so there is no breakdown of marks in the rubric – they are all marked together. We expect appropriate UML diagrams to be provided to support the explanation provided in your report.

H1 [16-20]:

Excellent extendable design and implementation.

1. Clear, consistent, and helpful rationale for design and other changes in terms of patterns.
2. Preserves original behaviour and implements new capability appropriately.
3. Few if any minor errors or omissions (no major ones).

H3-H2B-H2A [13-15.5]:

Good extendable design and implementation.

1. Generally, clear, consistent, and helpful rationale for design and other changes in terms of patterns.
2. Preserves original behaviour and implements new capability appropriately.
3. Few errors or omissions (no major ones).

P [10-12.5]:

Reasonable design and implementation.

1. Fairly clear, consistent, and helpful design rationale with some reference to patterns.
2. Preserves original behaviour with at most minor variations in output (or similar behaviour to original)
3. Implements new capability appropriately.
4. Some errors or omissions.

F+ [5-9.5]:

Plausible design and implementation.

1. Plausible attempt at design rationale with some reference to patterns.
2. Plausible implementation, that is, required elements are present.
3. Includes errors and/or omissions.

F [0.0-4.5]:

No plausible design/implementation.

1. Design rationale provided but not particularly plausible/coherent.
2. No plausible implementation.

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.