CS 247 Spring 2012
Straights Program Specifications (Part 2)

DUE: **Friday July 13, 2012** at **5:00PM**

You must submit all of your source code through Marmoset in a zip archive. It should include
- All .h files
- All .cpp files
- All images
- Makefile (should create an executable called `straights`)
- Bonus features text file

We will grade your program based on its design and programming style as well as its correctness. See the rubric for details on how your program's design and programming style will be marked.

This deliverable will NOT be marked automatically, so the output specifications are less strict. Instead, you will demo your submitted solution for a TA. Your solution and demo MUST run on one of the `student.cs.uwaterloo.ca` machines, so **be sure that your solution doesn't use libraries that are not installed on these machines.** Use UNIX command `locate gtk+` to check that the machine(s) you are using has GTK+ installed.

0. Starting a game

The user starts the program WITHOUT initializing the seed of the random number generator on the command line. Call the srand48 function with seed 0 to initialize the random number generator.

1. Setting the seed

At any time, the user can reinitialize the seed of the random number generator. The reinitialized seed is used when the user starts a new game.

2. Invite Players

At the beginning of **every game**, allow the user to determine whether each player (Player 1 through Player 4) will be played by a human or by the computer.

3. Shuffling and Dealing

Initially (before any game starts), the cards in the deck should be in the following order:

AC 2C 3C … QC KC AD 2D … QD KD AH 2H … QH KH AS 2S … QS KS

At the beginning of every round, call the provided shuffle function *once*. Do not shuffle the cards or reset the deck in any other way. After the shuffle, assume that the first 13 cards belong to Player 1, the next 13 cards belong to Player 2, the next 13 belong to Player 3, and the last 13 cards belong to Player 4.

4. Gameplay – Human Player

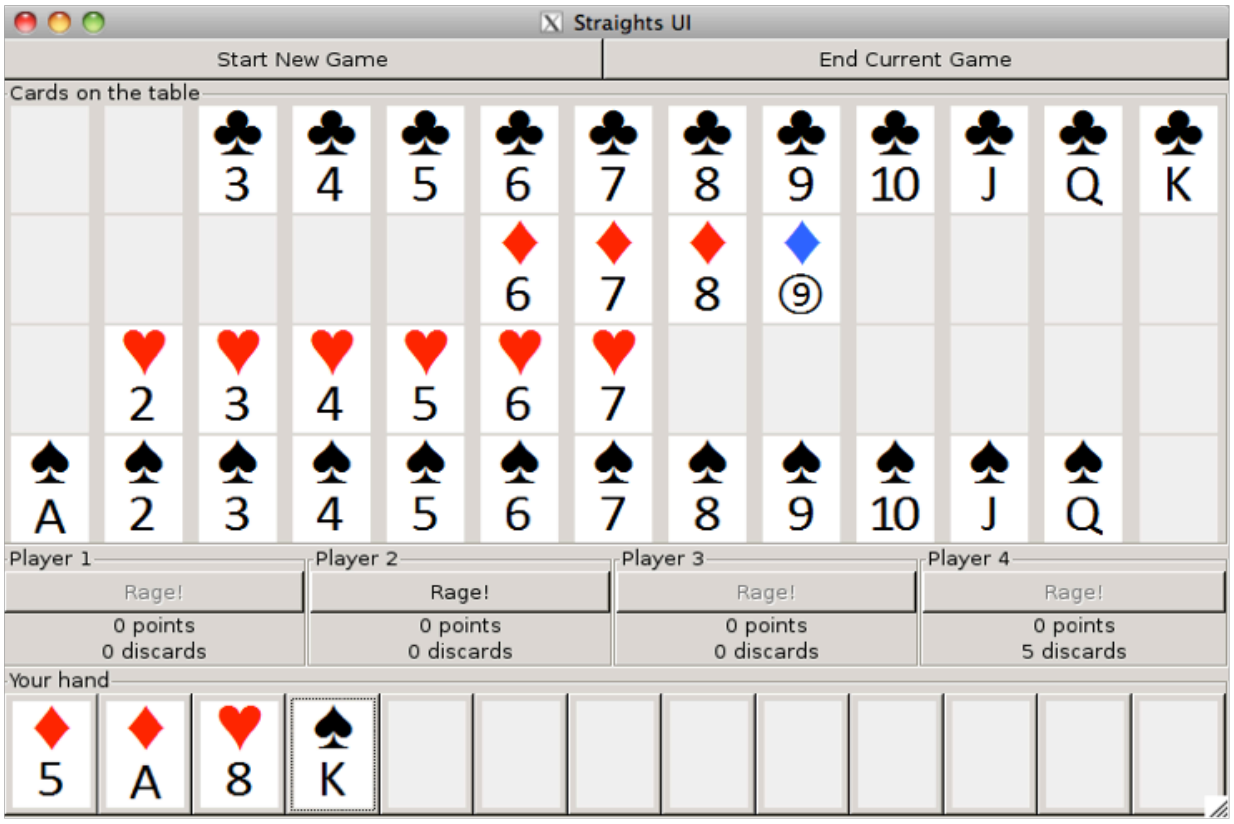
The game starts after the shuffle and the deal. The four players will take turns to play their cards. The player who has the 7 of spades goes first.

4. Gameplay – Human Player (cont.)

Whenever it is a human player's turn to play, update the user interface to display the following information (at a minimum):

- The cards on the table, arranged in four lists according to suit. Each list is an ordered sequence of all the ranks that have already been played in a particular suit.
- Each player's score
- Each player's number of discards
- The cards in the hand of the player whose turn it is

As an example of such a UI, below is a snapshot of David Lou's solution. Feel free to base your UI design on his or to create your own.



The program will then wait for the user to enter a command.

5. Gameplay – Commands

There are 4 valid commands that a human player can enter. These commands are made via a UI event (e.g., a mouse click on a button widget). Note that there is no `deck` command:

```
select <card>
start new game
quit
ragequit
```

5. Gameplay – Commands (cont)

a) select <card>

The human player selects a card in his or her hand. If the play is legal, then remove the card from the player's hand and add it to the appropriate straight on the table. If the play is not legal but there exists a legal play in the hand, then issue an error message (e.g., in a message dialogue). Otherwise, add the card to the player's discard pile.

b) start new game

At any time, the user can start a new game. If the user has reset the random-number-generator seed since the start of the current game, call the srand48 function to reinitialize the generator with the current seed value before shuffling the deck for the new game. Otherwise, shuffle the deck and start the new game.

c) quit

At any time, the user can terminate the program.

d) ragequit

Filled with anger, a human player decides to leave! Replace the current human player with a computer player, and resume the game!

6. Gameplay – Computer Player

The computer player that you will implement is very simple. It always makes the first legal play in its hand, and the played card is added to the table. If there are no legal plays, the first card in its hand will be discarded.

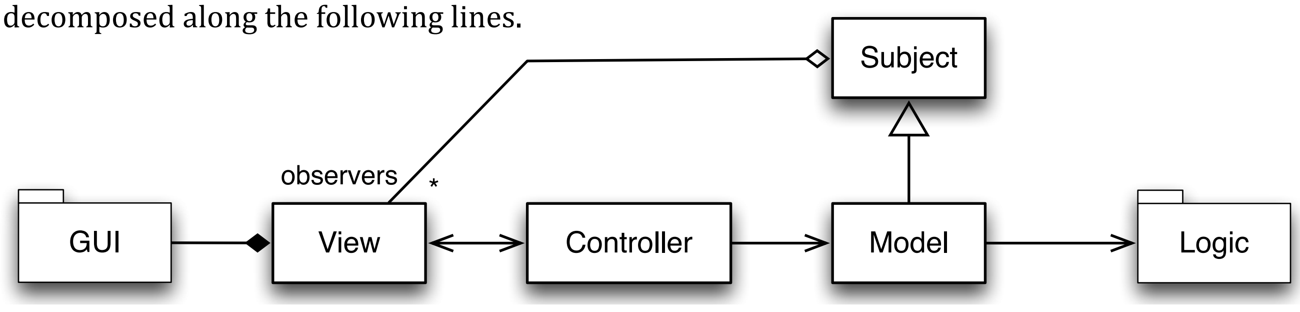
7. Scoring

When all the cards have been played, the round ends. At this point, output (e.g., in a message dialogue) each player's list of discards and update the players' scores. If one player has accumulated 80 points or more, the game ends. The player with the fewest points wins. Output a victory message for the winning player. If multiple players tie for first place, output a victory message for each winner.

If no one exceeded the score limit, then reshuffle the deck and begin another round.

8. Design Requirements

Your solution must use the Model-View-Controller design pattern to separate your program logic (i.e., the code that implements the Straights game) from the GUI; it must also use the Façade design pattern to create a simple interface for your game logic. Thus, we are expecting your program to be decomposed along the following lines.



8. Design Requirements (cont.)

Once the program is initialized, the flow of control alternates between the GUI and the Model portions of the program:

- The GUI waits for some user input (e.g., select a menu option, press a button).
- The GUI event is forwarded to the Controller, which "translates" the GUI event into appropriate calls to operations in the Model part of the program.
- A change to the state of the Model is announced to Observers (the View) of the Model
- The View updates the GUI to reflect the changed Model state
- The View prompts the human player for his or her next input, and the sequence repeats.

We recommend that you separate your logic code from your GUI code by placing them in separate subdirectories. If you do this, you will have to modify your Makefile so that it looks for files in the appropriate subdirectories. Also, in your GUI code, the path to your image files must be relative to the location of the executable.

9. Incremental Development

We **strongly** recommend that you incrementally develop the GUI and connect it with the logic from your previous assignment, so that you are sure to have something to demo if you are unable to complete the project:

1. Create a gtk window and close it. (See gtkmm-examples/01initialHelloWorld)

2. Create a window that will let you display card images horizontally. This implies a Gtk::HBox, possibly inside a Gtk::Frame, will be added to the Gtk::Window. See gtkmm-examples/01initialHelloWorld and gtkmm-examples/04pixmaps/ex2 for some ideas on how to approach the problem.

3. Make a menu option (or a button or a toolbar button) to start a new game. You will find the examples in gtkmm-examples/03menusAndToolbars/ex[12] useful.

4. Add the Observer pattern (part of MVC), so that the logic can notify the GUI when the game state changes (e.g., new card is played, new card is discarded), and have the display update appropriately.

5. Add the Façade design pattern, to create a single interface to all of the classes that you developed in the first deliverable. If you have a game class, you may wish to use this, or simply create a new class. At this point, your Straights program should be able to deal out the cards to the players and show the hand of the human player whose turn is first.

6. Add the ability for the human player to select a card to play, and have the GUI transfer (via a Controller) the appropriate face and suit information to the logic. This requires registering a handler with your widget. Note that Gtk::Image by itself doesn't receive events, so you will need to embed it in something that does, such as a Gtk::Button or a Gtk::EventBox.

7. Modify the program logic to notify the View that a particular player has played a card or that a round has ended.

BONUS

You can earn bonus marks for adding extra functionality or features to your program. Example bonus features include adding a textual record of the players' plays, providing visual cues to which cards are legal plays, saving and resuming partially played games, allowing a human player to undo a sequence of at least 4 plays, drawing images on top of other images, adding music, or other extensions of comparable difficulty.

Each bonus feature is worth up to an additional 5% on the second deliverable. You can earn a maximum of 10% in bonus marks (i.e., up to 12.5 marks on top of the project's 125 marks).

You will be required to demonstrate your enhancements as part of your demo of your second deliverable.

In addition, you must document what you have done in the file bonus.txt. This will be a description of each bonus feature you have implemented. Include any necessary seed values and input sequences that are needed to demonstrate the extra features.