CompSci 105 S2 C
- ASSIGNMENT ONE -

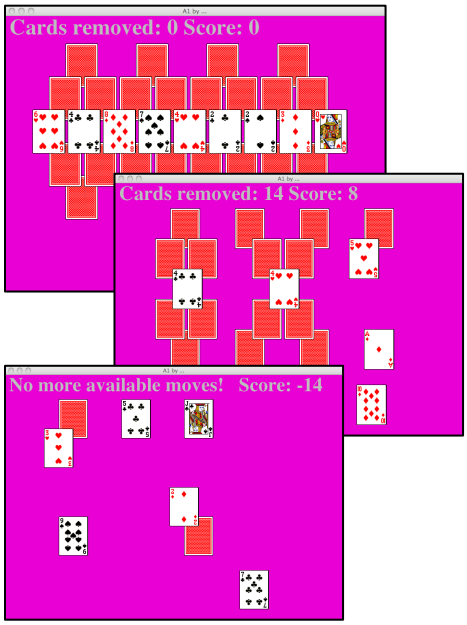
The work done on this assignment must be your own work. Think carefully about any problems you come across, and try to solve them yourself before you ask anyone else for help. Under no circumstances should you use code written by another person in your assignment solution.

Assignment One

Due 5.30pm , Thursday16th August, 2012
Worth 7% of your final mark

Aim of the assignment

- practice with 2D arrays and ArrayLists
- reading information from a text file



For this assignment you need to submit FIVE Java source files and ONE text file:

- `A1.java`
- `CardImageLoadUp.java`
- `A1Constants.java`
- `Card.java`
- `A1JPanel.java`
- `A1.txt` (see later in this document)

NOTE: The folder, `classic_cards`, contains a card image for each playing card. This folder needs to be in the same folder as your program files so that the images can be accessed by the code.

Part 1 The Card class (8 marks)

Complete the definition of the `Card` class so that the `CardTester` application executes correctly. Parts of the `Card` class are completed and should not be changed. The completed parts of the `Card` class are: `toString()` method, `drawCard()` method and the instance variables.

Things to note about the `Card` class:

The constructor

The two `boolean` instance variables are initialised to `false`. The card area should be initialised as a `new Rectangle(0, 0, 0, 0);` (the game (part 2) will crash if you leave the `area` instance variable as `null`.)

`public boolean checkPressPoint(Point pressPt)`

If the card has been removed this method returns `false`.
If the `Point` passed as a parameter is inside the card area, the method should return `true`.
In all other cases the method returns `false`.

`public String getCardStatusInformation()`

Returns a String containing the card value, card suit, x-position of the card, y-position of the card, the card has been removed `boolean` and, lastly, the card is face up `boolean`. All the information is separated by spaces. Three examples of Strings obtained from the `getCardStatusInformation()` method are shown below:

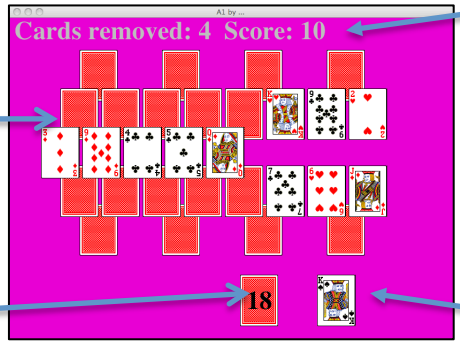
```
4 0 327 276 false false
4 3 404 276 false true
6 2 481 276 false false
```

Use the `CardTester` application to check your `Card` class definition. (Note that some parts of the class are not checked by the `CardTester` application.) Pressing the mouse on a card (or card area):

makes the card turn face down (if it is face up),
makes the card disappear (if it is face down),
makes the card appear (if it had previously disappeared).

As well, the textfield displays the `toString()` information and some other information about the card which was pressed.

Part 2 The Solitaire game



Score display

Table cards: initially there are 33 cards on the table (5 rows of up to 9 cards each row).

User pile: pile of cards removed from the table (or turned from the card stack) by the player. (Only the top of the user pile is visible.)

Card stack: the cards which are left over (18 cards to start with).

Playing the game: in this game of solitaire the player tries to remove all the cards from the table onto the user pile. A card can be removed from the table to the user pile if the card on the table has a value difference (higher or lower) of exactly 1 from the card on top of the user pile (e.g., three and a four, three and a two, ace and a two, ace and a king, etc.). Any card which is removed from the table becomes the top card of the user pile. If the player cannot remove any of the cards from the table, the player can turn a card from the card stack and put it face up onto the user pile. The aim of the game is to get as many points as possible.

Scoring the game: The player starts with 0 points. Taking a card from the card stack reduces the score by 5 points. Each card which is removed from the table gains points: the first card removed gains 1 point, the second card removed gains 2 points, the third card removed gains 3 points, etc. However each time a card is taken from the card stack (loss of 5 points) the point value gain for removing the next card from the table is set to 1 point again.

The files used for this part of the assignment are:

- `A1.java`
  This is the application class (note that the `A1JFrame` class is also defined inside this file).
- `A1Constants.java`
  This class contains the constants used in this game.
- `CardImageLoadUp.java (should not be changed)`
  This class contains static methods used for loading the card images, getting a single card image (used in the drawCard() method of the Card class to draw a single card) and for obtaining the width and the height of a single card.
- `Card.java - You have completed this class in part 1 of this assignment`
  This class stores the information about a single card and has methods for manipulating the card.
- `A1JPanel.java - You need to complete the code for this class`
  This is the JPanel class which handles MouseEvents, KeyEvents and generally manages the program.

VERY IMPORTANT

Check the marking schedule (and penalties on page 9) as you complete the assignment.

Please read this assignment document carefully to make sure that you complete all the requirements.

It is recommended that you develop your program in stages as follows:

Stage 1: The title bar of the JFrame (4 marks)

Your UPI should be displayed inside the title bar of the window. To do this, open the `A1.java` application file and insert your UPI after "A1 by":

```
JFrame gui = new A1JFrame("A1 by ...  ", ... );
```

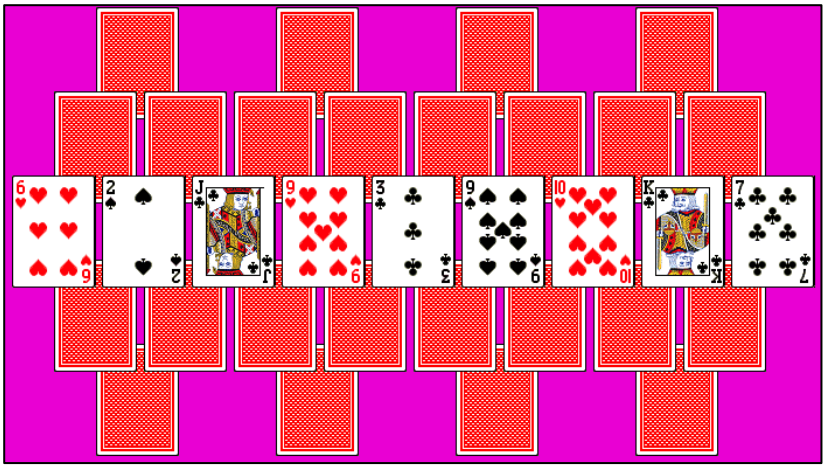
Stage 1 – The title bar The student's UPI is displayed in the title bar.		/4

Stage 2: Card[][] getRandomTableCards() (8 marks)

For this assignment you MUST use a 2D array of `Card` objects to store the five rows of table cards of the solitaire game. Complete the `getRandomTableCards()` method which creates and returns a two dimensional array of random `Card` objects. The 2D array has 5 rows and 9 columns (given by the constants `NUMBER_OF_ROWS`, `NUMBER_OF_COLS`) of cards but some of the cards in each row, except the central row, are `null`. The constant array:

```
final int[] CARDS_IN_EACH_ROW = {4, 8, 9, 8, 4};
```

stores how many non-`null` cards there are in each row of the 2D array, i.e., row 0 has 4 non-`null` cards followed by 5 `null` cards, row 1 has 8 non-`null` cards followed by 1 `null` card, etc. The cards in this game are set out as follows:



The parameter, `cardStack`, is an `ArrayList` of all 52 cards in the pack. The 2D array created and returned in the `getRandomTableCards()` method should contain 33 cards randomly removed from the `ArrayList` and stored in 5 rows and 9 columns. Once this method is completed the `cardStack` `ArrayList` contains the remaining 18 cards.

Notes for Stage 2

After each `Card` object has been created, the

```
setupIndividualCardPosition(Card card, int rowNumber,
                                                int colNumber) { ... }
```

method should be called. This method correctly sets the top, left, width and height of the `Card` object so the card is positioned in the correct place inside the `JPanel`.

Pressing the mouse on the card stack should remove cards from the card stack. You will not see any of the table cards in the `JPanel` area until stage 4 has been completed.

Stage 3: setUpVisibleRowOfCards() (4 marks)

You can see from the screenshot above that only the middle row contains cards which are all face up. Complete the `setUpVisibleRowOfCards()` method so that all the cards in the middle row of the 2D `Card` array, `cards`, are set to face up.

Stage 4: paintComponent () (6 marks)

Complete the `drawTableCards()` method (called from the `paintComponent()` method) so that all the table cards in the 2D `Card` array are drawn. The randomness of the cards will be tested later.

Notes for Stage 4

Make sure the `drawTableCards()` method only draws cards in the array which are not `null`.

You may choose to use the `drawRowOfCards()` helper method which draws a single row of cards.

Rows 0 and 4 should be drawn first, then rows 1 and 3 and lastly the middle row should be drawn.

Stages 2, 3 and 4 – The 2D array of cards are visible			
The getRandomTableCards() method returns a 2D array of Card objects.	**(4)**		
The cards are displayed correctly in rows and columns.	**(6)**		
The middle row is all face up cards.	**(4)**		
The cards are random. (Verified after stage 5)	**(4)**		
Penalty if the cards are not stored in a 2D array	**(-20)**		**/18**

Stage 5: starting a new game (6 marks)

The program should handle `KeyEvents`. When the user presses the key, 'n' or 'N' a new solitaire game should appear in the `JPanel`. Add the code which implements this.

Notes for Stage 5

The `reset()` method is already defined and sets up everything ready for a new game.

Stage 5 – Starting a new game			
A new game starts whenever the user presses either 'n' or 'N'.	**(6)**		
Verify that the cards on the table are random (stage 2).			**/6**

Stage 6: getRowColOfSelectedCard() (8 marks)

Complete the `getRowColOfSelectedCard()` method which has one parameter, the `Point` where the user pressed the mouse. If the mouse press is inside a card which is facing up, the method returns a `Point` representing the row and the column of the card in which the mouse was pressed, otherwise the method returns `null`.

Notes for Stage 6

The `Point` returned by this method represents the row column of the card where the user pressed the mouse, e.g., the point (0, 3) is returned when the user presses the mouse inside the card in row 0 column 3.

Make sure the method only tests cards in the 2D array which are not `null`.

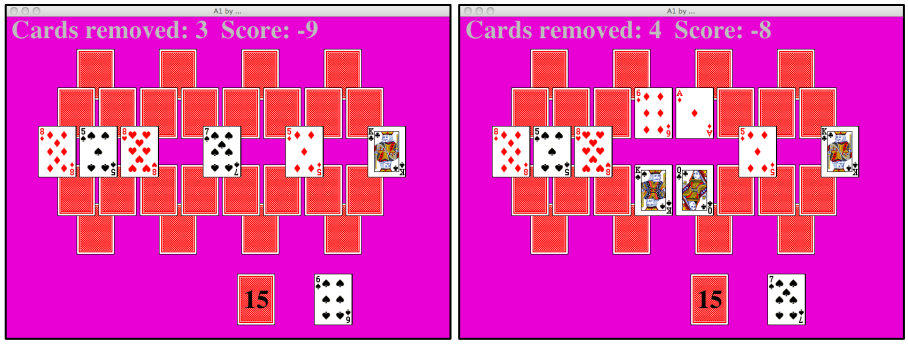
Once this stage has been completed, the face up cards should disappear when the user presses the mouse inside a card which has a value exactly 1 more/less than the user card. The neighbouring cards will only be revealed once stage 7 is completed.

Stage 6 – Selecting a face-up card			
The user is able to select any one of the face up cards by pressing the mouse inside the card. If the difference in value of the selected card and the user card is 1 then the selected card should disappear and the user card is replaced by the selected card. Pressing the mouse inside any other card should have no effect.	**(8)**		**/8**

Stage 7: Turn neighbouring cards to face up (20 marks)

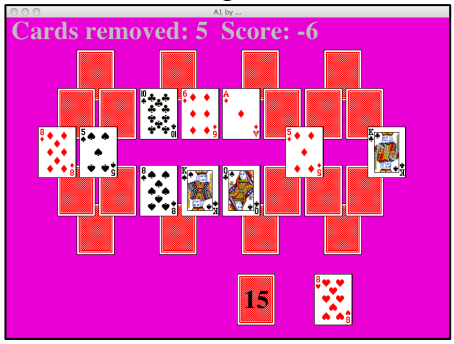
From the screenshot on the previous page, notice that
row 2 is the uppermost row,
the cards of row 2 cover the cards in rows 1 and 3, and,
the cards of rows 1 and 3 cover the cards in rows 0 and 4 respectively (the lowest levels).

Whenever the player removes a card from the table, any neighbouring cards, which are not covered by other cards (face up or face down), should be turned face up. For example, look at the following screen shots:



Above left: the player presses the mouse on the 7 of spades (row2 card 4). Because there is only a difference of one between the value of the user card (6 spades) and the value of the selected card (7 spades), the selected card is removed and it becomes the user card. As well, any neighbouring cards which are no longer covered by other cards (face up or face down) will be turned to face up. In the screenshot above right you can now see that the 6 of diamonds and the ace of diamonds in row 1, and the king of spades and queen of clubs in row 3 have been turned to face up.

The player now presses the 8 of hearts (row 2 card 2). Because there is only a difference of one between the value of the user card (7 spades) and the value of the selected card (8 hearts), the selected card is removed and it becomes the user card. As well, any neighbouring cards which are no longer covered by other cards (face up or face down) will be turned to face up. In the screenshot below you now see that the 10 of clubs in row 1 and the 8 of spades in row 3 have been turned to face up.



NOTE: There are other examples **on the last page** of this document.

Complete the `revealNeighbouringCards()` method. This method turns any cards which which are no longer covered to face up. Note that before the `revealNeighbouringCards()` method is executed the selected card has been removed from the 2D `cards` array, i.e., the position where the selected card was in the `cards` array is `null`.

Notes for Stage 7

When checking for neighbouring cards which should be turned face up, you need to check the row(s) immediately under the row of the card which has been removed. For example, when the player removes a card from row 1, only row 0 needs to be checked. When the player removes a card from row 3, only row 4 needs to be checked. When the player removes a card from row 2, both rows 1 and 3 need to be checked.

To find out which cards are the neighbouring cards, check for the intersection of the card areas. Once you have found a card with an area which intersects the removed card area, you still need to check that the card you have found has no other card covering it.

Make use of the helper methods so your code is not awkwardly nested, e.g.,

```
checkRowOfNeighbours( ... )
hasNoIntersectingNeighbourInRow( ... )
```

Stage 7 – Turning the neighbours to face up When a card is removed from the table, the correct neighbouring card/s (if any) are turned to face up. **(16)** The code is clear and easy to understand. **(4)**			**/20**

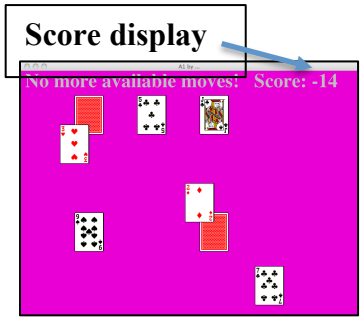
Stage 8: Testing if all the cards have been removed (6 marks)

When all the cards on the table have been removed the game should end. Complete the method, `noMoreTableCards()`. This method returns `true` if all the cards on the table have been removed, `false` otherwise.

Stage 8 – No more table cards The noMoreTableCards() method returns true if there are no more cards on the table, false otherwise.			**/6**

Stage 9: Scoring the game (10 marks)

Add code to the program so that the game is scored correctly. The score is displayed in the top right hand side of the `JPanel`.



Scoring the game: The player starts with 0 points. Taking a card from the card stack takes away 5 points. Each card which is removed from the table gains points: the first card gains 1 point, the second card gains 2 points, the third card gains 3 points, etc. However each time a card is taken from the card stack (loss of 5 points) the point value gain for removing the next card from the table is reset to 1 point again.

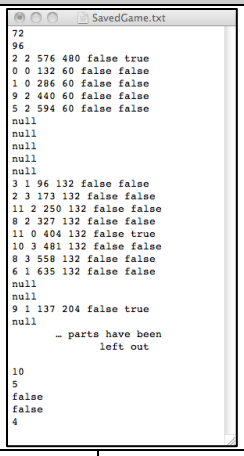
Notes for Stage 9

The instance variable, `userScore`, is used to store the current score and the instance variable, `pointsToAdd`, can be used to store the current number of points to add when the player next removes a card from the table.

Stage 9 – Scoring the game The current score is reduced by 5 points every time the player selects a new card from the card stack. **(4)** The correct number of points are added to the current score every time a card is removed from the table. **(6)**			**/10**

Stage 10: Writing a game to a file (2 marks)

The `JPanel` class contains the, `writeToFile()` method which can be used to save the state of the solitaire game to the text file, 'SavedGame.txt'.



The player should be able to save a game of Solitaire by pressing the 's' (lower or uppercase) key. Add the code which implements this to the `keyPressed()` method.

Look at the code which saves the game information to file. For stage 11 you will need to understand the order in which the game information is stored in the file.

Stage 10 – Storing a game to the `SavedGame.txt` file When the user presses the 's' key the game is stored in the `SavedGame.txt` file. **(2)**		**/2**

Stage 11: Loading a saved game from the SavedGame.txt file (8 marks)

A saved half finished solitaire game is stored in the text file, 'SavedGame.txt'. The player can load the last saved solitaire game by pressing the 'l' (lower or uppercase) key. In the `keyPressed()` method add the code which implements this.

Complete the `loadFromFile()` method which should correctly load the game from the 'SavedGame.txt' file into the `JPanel` instance variables. (Look carefully at the code which saves the game information to file (stage 10) because, for stage 11, you will need to understand the order in which the game information is stored in the text file.)

Handling Exceptions

If there is an exception when loading from the file an error message is displayed using a `System.out.println()` statement:

```
Error loading game from SavedGame.txt
```

Your code should always be able to load a game from the file unless the file is missing or corrupted. [If there is a problem with loading the information in the file (shouldn't happen), the solitaire game from the file may not always be useable if it is half-loaded. This means that the player will need to press the 'n' key for a new game whenever the information from the file is partially loaded.]

Notes

When loading the information about a card it is a good idea to use the `createACard()` helper method provided to create a `Card` object from the String read from the file:

```
private Card createACard(String info, int width, int height) { ... }
```

Stage 11 – Loading a game from the SavedGame.txt file When the user presses the 'l' key the game is loaded from the SavedGame.txt file. **(2)** The game is correctly loaded from the file. **(4)** The player is able to continue the game loaded from the file without problems. **(2)**		**/8**

Submission Instructions

You should submit all of the following files for this assignment through the web-based Assignment Dropbox.

REQUIRED FILES

For this assignment you need to submit FIVE Java source files and ONE text file:

- `A1.java`
- `CardImageLoadUp.java`
- `A1Constants.java`
- `Card.java`
- `A1JPanel.java`
- `A1.txt` – a text file containing your feedback on the assignment (see below).

MAKING MORE THAN ONE SUBMISSION

You can make more than one submission - every submission that you make *replaces* your previous submission. **Submit ALL your files in every submission**. Only your very latest submission will be marked.

NEVER SUBMIT SOMEONE ELSE'S WORK:

- If you submit an assignment you are claiming that you did the work. Do not submit work done by others.
- Do not *under any circumstances* copy anyone else's work – this will be penalized heavily.
- Do not *under any circumstances* give a copy of your work to someone else.
- We uses copy detection tools on the files you submit. If you copy from someone else, or allow someone else to copy from you, this copying will be detected and disciplinary action will be taken.

What you should include in the A1.txt file

You **must** include a text file named A1.txt in your submission. There will be a 5 mark penalty for not doing so. This text file must contain the following information:

Your full name
Your login name and ID number
How much time did the assignment take overall?
What areas of the assignment did you find easy?
What areas of the assignment did you find difficult?
Any other comments you would like to make.

Marking Schedule

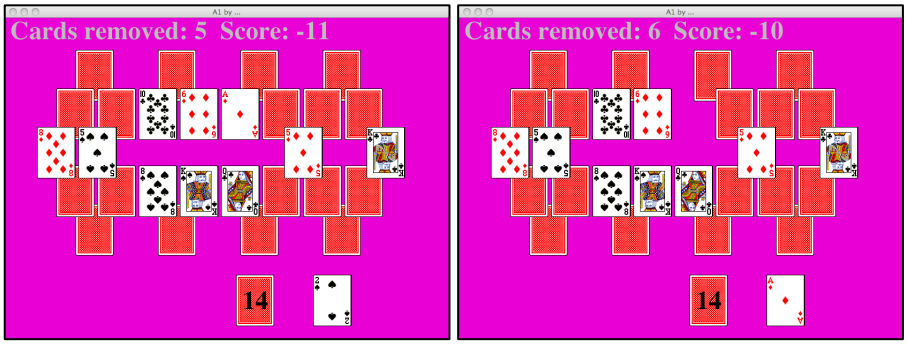
Style Comments at the top of the Card class and the JPanel class with name, date and a comment. **(2)** Good identifier names. **(2)** Correct indentation. **(3)** Uses helper methods. **(3)**		**/10**
The Card class		**/8**
Student's UPI is in the title bar of the window.		**/4**
Stage 2 – getRandomCardsArray() method **Stage 3** – Set up visible middle row **Stage 4** – Drawing the cards **Stage 5** – Starting a new game **Stage 6** – Get row, col of selected card **Stage 7** – Turning neighbouring cards to face up **Stage 8** – All table cards have been removed **Stage 9** – Scores the game correctly **Stage 10** – Player presses 's' key to store current game **Stage 11** – Loads the game from the text file correctly		**/8** **/4** **/6** **/6** **/8** **/20** **/10** **/6** **/2** **/8**
Grand Total		**/100**

Penalties

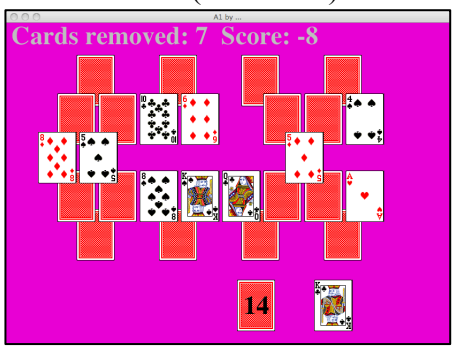
Did not use a 2D Card array - penalty of 20 marks
Program declares instance variables when local variables should be used - penalty of 10 marks
A1.txt information was not submitted - penalty of 5 marks

Stage 7: Reveal neighbouring cards – other examples

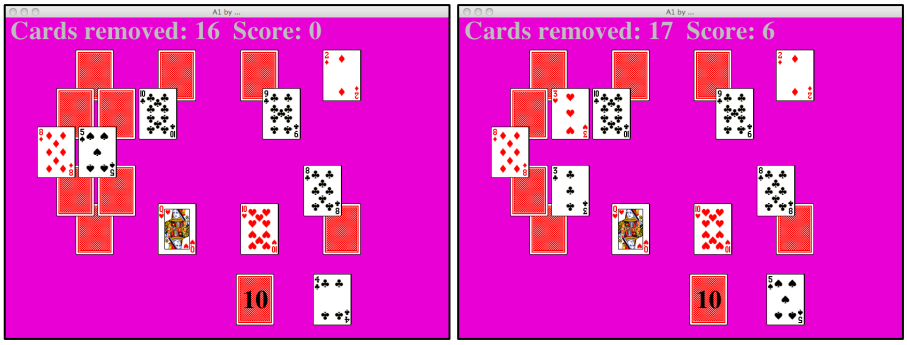
Example 1:



Above left: the user presses on the ace of diamonds (row1 card 4). Because there is only a difference of one between the value of the user card (2 spades) and the value of the selected card (ace of diamonds), the selected card is removed and it becomes the user card. In this case none of the cards in row 0 are completely uncovered so no extra cards are revealed (see above right).
The user now presses the king of clubs (row 2 card 8). Because there is only a difference of one between the value of the user card (ace) and the value of the selected card (king), the selected card is removed and it becomes the user card. In this case the last card of row 1 and the last card of row 3 are completely uncovered so they are both revealed (see below).



Example 2:



Above left: the user presses on the 5 of spades (row2, col 1). Because there is only a difference of one between the value of the user card (4 spades) and the value of the selected card (5 spades), the selected card is removed and it becomes the user card. In this case the second card of row 1 and the second card of row 3 are completely uncovered so they are both revealed (see above right).