UNIVERSITY OF SOUTHERN QUEENSLAND
CSC1401 - Foundation Programming, S2 2017
Assignment III Specification

# Memory Game

**Due Date: 6 Oct 2017**

**Weight: 20%**

**Type: Team-based(3 members)**

## Contents

## Goals and Topics

The assignment problem is straightforward. All necessary details have been supplied. The solution of the problem will use the programming concepts and strategies covered in Workshops 1 – 10 in the course. The subgoals are:

- Obtaining advanced understanding of values, variables and arrays;
- Understanding program input and output, functions and expressions;
- Understanding simple strategies like iteration, validation, sum, count, minimum, and maximum plans;
- Understanding of advanced strategies like swapping and shuffling, cycle position plan, sorting, tallying, searching, and recursion;
- Understanding of HTML objects, forms, and events;
- Translating simple design into programming code
- The mechanics of editing, interpreting, implementing, running and testing a program
- Commenting source code, especially JavaDoc on functions
- Becoming confident and comfortable with programming in small problems

## Background Information

Figure 1: The Memory Game

Memory game, also known as Concentration, Pexeso, Shinkei-suijaku in Japan, Pelmanism in UK, or simply Pairs, is a card game in which all of the cards are laid face down on a surface and two cards are flipped face up over each turn. The object of the game is to turn over pairs of matching cards. Concentration can be played with any number of players or as solitaire and is an especially good game for young children, though adults may find it challenging and stimulating as well. The scheme is often used in quiz shows and can be employed as an educational game.

### Rules and Strategy

Any deck of playing cards may be used, although there are special cards available, as shown in Fig. 1. The rules given here are for a standard deck of 52 cards, which are normally laid face down in 4 rows of 13 cards each. In turn each player chooses two cards and turns them face up. If they are of the same rank and colour[1] (e.g. 6 of hearts and 6 of diamonds, queen of clubs and queen of spades, or both jokers, if used) then that player wins

[1]Note that the computer game you are going to develop in this assignment follows slightly different rules. Refer to Section "Functionality of the Program" for details.

the pair and plays again. If they are not of the same rank and colour, they are turned face down again and play passes to the player on the left. The game ends when the last pair has been picked up. The winner is the person with the most pairs, and there may be a tie for first place.

Over the course of the game, it becomes known where certain cards are located, and so upon turning up one card players with good memory will be able to remember where they have already seen its pair. It is common for many players to think they know where pairs are and to turn over the one they are sure of first, then be stumped finding its mate. A better strategy is to turn over a less certain card first, so that if wrong, one knows not to bother turning a more certain card over.

An ideal strategy can be developed if we assume that players have perfect memory. For the One Flip variation below, this strategy is fairly simple. Before any turn in the game, there are $t$ cards still in play, and $n$ cards still in play but of known value. The current player should flip over an unknown card. If this card matches one of the known cards, the match is next chosen. Less obviously, if the card does not match any known card, one of the $n$ known cards should still be chosen to minimise the information provided to other players. The mathematics follow: *If a remaining unknown card is chosen randomly, there is a* $\frac{1}{t-1-n}$ *chance of getting a match, but also a* $\frac{n}{t-1-n}$ *chance of providing opponents with the information needed to make a match.* There are some exceptions to this rule that apply on the fringe cases, where $n = 0$ or 1 or towards the end of the game.

At VanillaGrape[2] you can play an online *Memory* game against the computer to help you understand the rules and strategies.

**Information Source**

- *Wikipedia, URL:http://en.wikipedia.org/wiki/Concentration_(game).*
- *Pelmanism, URL: http://www.pagat.com/misc/pelmanism.html.*

## Functionality of the Program

In this assignment, you are going to develop a computer based memory game that can be played solo against the clock. In the game, the player is given 60 seconds. Each time the player chooses two cards. The cards will stay if they are matching pairs of the same rank, ignoring colours. Otherwise, they will flip over back to face-down. The result is calculated by the number of matching pairs the user found in 60 seconds.

Figure 2 illustrates a gameplay. As you may see, the memory game has three control buttons:

- **Stop Game**: stops an ongoing game;
- **Flip Cards**: flips all cards over to face-down except from those found paired cards;
- **Show Cards**: shows all cards to face-up.

[2] http://vanillagrape.com/Memory/single_player/game.

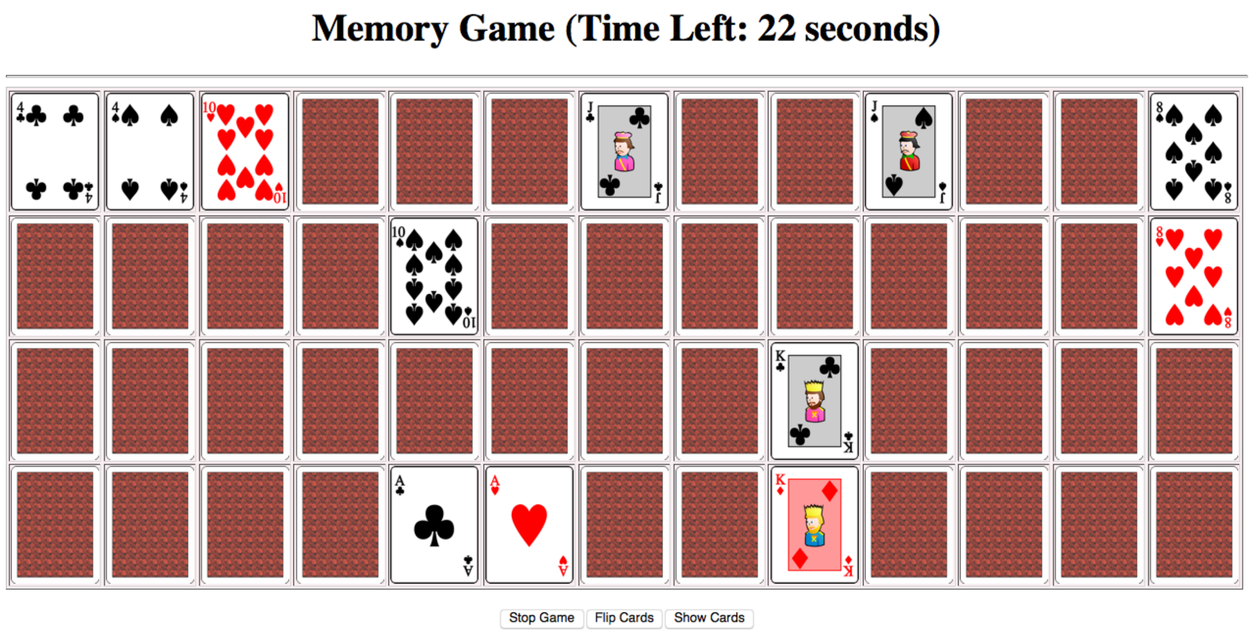
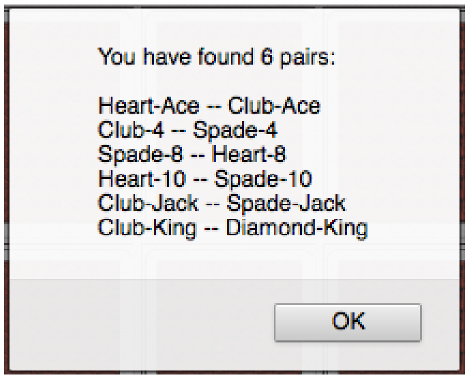


Figure 2: The Memory Game in this Assignment



You have found 6 pairs:

Heart-Ace -- Club-Ace
Club-4 -- Spade-4
Spade-8 -- Heart-8
Heart-10 -- Spade-10
Club-Jack -- Spade-Jack
Club-King -- Diamond-King

Figure 3: The Pop-up Alert Window Showing the Result of a Gameplay

In the title a timer is set, counting down in second from 60 to 0. The game will automatically stop when the time is up, and then an alert window will pop up presenting the gaming result, such as how many pairs the player has found and the list of cards in ascending order based on their ranks from *Ace*, *2*, *3*, ..., *10*, *Jack*, *Queen*, to *King*. Each card should be represented by the format of *suit* – *rank*, for example, "Diamond-10", "Spades-Jack". An illustration of the pop-up alert window showing the gaming result is in Fig. 3.

Download the template source file and card images on StudyDesk to start your work. Note that you need to put the program (the .html file) and card images in the same folder to make it working.

## Requirement Specification (Marking Criteria)

You should use Table 1 as the checklist for implementation and to guide your testing of the program. The total 40 marks will then be converted to 20, which is the grade carried by Assignment III.

You have two options in how to complete the assignment:

- Deign the program by yourself and implement the game following your own design. Note

Table 1: Requirement Specification (Marking Criteria)

| ID | **REQUIREMENTS** | MARK |
|---|---|---|
| | *Statement of Completeness* | |
| SC-1 | The statement is in appropriate length of 200-300 of own words covering issues on both programming and team working | 1 |
| SC-2 | The "State of assignment" reflects the true state of completeness | 1 |
| SC-3 | The "Problems encountered" discusses problems and dealing strategies | 1 |
| SC-4 | The "Reflection" discusses learned lessons and reasonable suggestions | 1 |
| | *Subtotal* | *4* |
| | *Functional Requirements* | |
| FR-1 | The game environment is set up the same as Figure 2 | |
| FR-1.1 | - Cards are displayed face-down in a 13 by 4 table | 2 |
| FR-1.2 | - The title and time are set on top; three buttons are set on the bottom | 1 |
| FR-2 | Clicking "Show Cards" button will show up all cards | |
| FR-2.1 | - Clicking "Show Cards" button will make all cards to turn face-up | 1 |
| FR-2.2 | - When clicking "Show Cards" cards are showing face-up one by one (animation effect) | 1 |
| FR-3 | Clicking "Flip Cards" button will turn all cards, except from those found paired, face-down | |
| FR-3.1 | - Clicking "Flip Cards" button will make face-up cards face-down | 1 |
| FR-3.2 | - When click "Flip Cards" the paired cards will stay face-up | 1 |
| FR-3.3 | - When click "Flip Cards" cards are turning face-down one-by-one (animation effect) | 1 |
| FR-4 | Click "Show Cards" or "Flip Cards" shouldn't affect an ongoing game, nor change the cards' order | 1 |
| FR-5 | The timer ticks down from 60 seconds when a game is loaded, and the game will be terminated when the time runs off | |
| FR-5.1 | - Timer ticks down second by second (animation effect) | 1 |
| FR-5.2 | - When 60 seconds runs off, the game stops | 1 |
| FR-5.3 | - No concurrent threads (timer) is allowed. The timer is cleared out when a game is terminated. | 1 |
| FR-6 | Starting a game | |
| FR-6.1 | - When a game starts the cards are in random order based on their ranks | 1 |
| FR-6.2 | - The randomising function is implemented by using the shuffling plan | 1 |
| FR-7 | Each time the player will click two cards. The card will stay face-up if paired, or flip face-down after a few milliseconds | |
| FR-7.1 | - After two cards selected, the cards stay face-up if their ranks are matching | 1 |
| FR-7.2 | - The paired (matching rank) cards will be recorded (stored) for presentation of gaming result when finished | 1 |
| FR-7.3 | - Paired cards cannot be selected again (clicking it shouldn't invoke any event leading to further tasks) | 1 |
| FR-7.4 | - After two cards selected, the cards turn back to face-down if unpaired (mismatching ranks) | 1 |
| FR-7.5 | - The mismatching cards will stay for a few milliseconds before turning face-down (animation effect) | 1 |
| FR-8 | When click "Stop Game" button an ongoing game is terminated | 1 |
| FR-9 | An alert window pops up, presenting the gaming result as Figure 3, when a game is terminated | |
| FR-9.1 | - The number of pairs displaying on the result alert window is correct | 1 |
| FR-9.2 | - The paired cards are presented in correct format of "suit-rank" and value (using Ace, Jack, Queen, and King), each pair in a separate line | 1 |
| FR-9.3 | - The paired cards are listed in ascending order from Ace to King, ignoring suits | 1 |
| FR-9.4 | - The sorting function is implemented by Bubble Sort algorithm | 1 |
| FR-10 | A new game with timer set to 60 seconds starts automatically when the gaming result window is killed | 1 |
| | *Subtotal* | *25* |
| | *Non-functional Requirements* | |
| NF-1 | The script is running free from any syntax errors, no code goes outside of script section except from what provided in template | 1 |
| NF-2 | No functions and strategies used in the script are beyond the course content | 2 |
| NF-3 | ALL functions should be formally commented in JavaDoc (appropriate in both style and content) | 3 |
| NF-4 | Code is grouped based on the common tasks attempting to. Each block of code is commented briefly | 1 |
| NF-5 | Code in the script is organised in order of variables, statements, and functions | 1 |
| NF-6 | All identifiers follow conventions in professional style | 1 |
| NF-7 | Code in the script is indented correctly for all loop, if-else statements, and functions | 1 |
| NF-8 | Code has been cleaned, all testing statements (e.g., print-lining) are removed. | 1 |
| | *Subtotal* | *11* |
| | **TOTAL** | **40** |

that the strategies employed in your design should be restricted to only those covered in the content of this course (Refer to Requirement ID $NF - 2$ on Table 1);

- Implement the program following the sample design in Appendix - I.

Clearly, the first option is more challenging and the second is more in the comfort zone. You can go with either one of them. However, in either option your work needs to satisfy, and will be marked against, the afore-described functionality and the detailed requirements specified in Table 1.

## Submission - Two Files

**A team makes only one copy of submission, in which all members are expected make equal contributions and share the same mark. Students need to register a team to the system first in order to make the submission portal available for making submission.**

For a complete submission you need to submit two files as specified below. You can submit them individually or compress them into a *.zip* (or *.rar*) file and submit it. The assignment submission system will accept only the files with the extensions specified in this section.

*1.* ***Statement of Completeness*** in a file saved in *.doc, .docx, .pdf* or *.txt* format in 200-300 of your own words describes the following issues. You should first specify the registered name of the team and all members' name and student ID on the top of the statement.

  - **The state of your assignment**, such as, any known functionality that has not been implemented, etc. (It is expected that most teams will implement all of the functionality of this assignment.)
  - **Problems encountered**, such as, any problems that you encountered during the assignment work and how you dealt with them. This may include technical problems in programming and people-soft problems in team working;
  - **Reflection**, such as, any lessons learnt in doing the assignment and handling teamworking and suggestions to future programming projects.

*2.* ***The program*** in a file saved with an *.html* extension contains the source code implemented following the functional and non-functional requirements.

Do NOT include the image files into the submission. The marker will download and put your program into the **local folder** with same images for testing and marking.

## Late Submission and Extension Request

Refer to *USQ Policy Library - Assessment Procedure* for information on the late submission policy and *USQ Policy Library - Assessment of Compassionate and Compelling Circumstances Procedure* for considerable special circumstances in extension request.

The Extension Request Form is available on StudyDesk. Should you need to request an extension please fill the form and email it to the Course Examiner with supportive documents (e.g., medical certificate or endorsement letter from supervisor in workplace) prior to the due date.

# Appendix I - A Sample Design of the Game

## Conceptual Design

Aiming at delivering the required functionality effectively, a conceptual design is developed and presented in Fig. 4. The design has two layers[3]:

- **Presentation Layer**: sets up the gaming environment and controls the presentation of cards. Five functions are included in the layer: $setGame()$, $showCard()$, $flipCard()$, $isPairedCard()$, and $sortCards()$;
- **Operation Layer**: handles the tasks in game playing, such as comparing the player-chosen cards to identify pairs. Five functions are included in the layer: $playGame()$, $stopGame()$, $resetTimer()$, $shuffleCards()$, and $selectCard()$.

A function in one layer might use (call to) functions in the other layer.

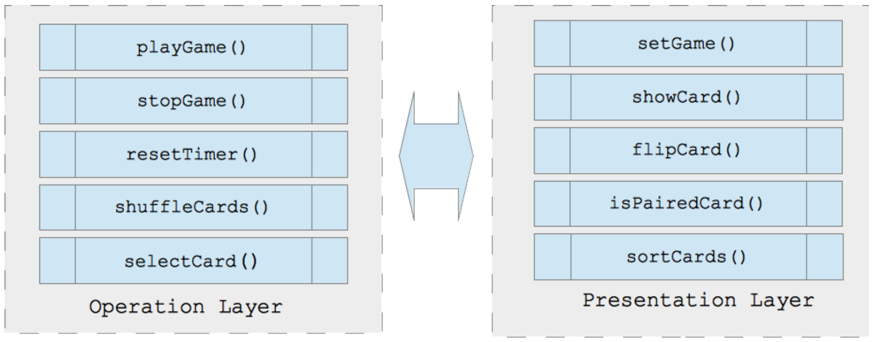


Figure 4: Conceptual Model

The program's architecture is designed on the basis of the conceptual design. Figure 5 presents the architecture and depicts the relationship between functions and the Interface, where solid arrows indicate the control flow in the game, and the open arrow suggests that it is $setGame()$ that prepares the Interface including the card set and three buttons ($stopGameButton$, $showCardsButton$, and $flipCardsButton$).

Array objects as are used to store data for game operation. Four arrays are created in the design of the program:

- $cardSet$: stores the unique IDs of all 52 cards. Conceptually, it is these unique IDs being shuffled or sorted when the card images appeared random (shuffled) or sorted. Initialise the elements (card IDs) using their index;

[3]Note that these layers are conceptual and drawn to help understand the design of the program. They are NOT components that you need to develop in the program.

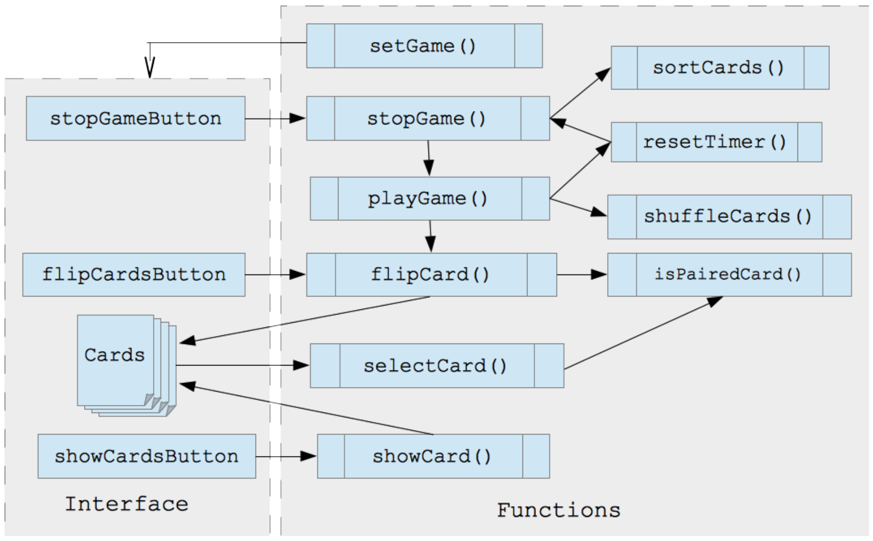


Figure 5: Architecture Design

- *preloadImages*: stores 52 *Image* objects, each with the *src* property referring to a card image file. The image files are named from 0 to 51, following the card ranks from Ace, 2, 3, . . . 10, Jack, Queen, King and then suites of Spades, Hearts, Diamonds, and Clubs;
- *guessCards*: stores the IDs of two cards that the player is currently choosing and pairing. Note that this array has only TWO elements;
- *pairedCards*: stores the IDs of cards that the player has paired correctly.

## Design Specification

### Starting Point

Call *setGame*() and then *playGame*() at the beginning of the program to load and start up the game.

### setGame()

This function sets up the environment for the game. It takes no argument and returns nothing.

1. Create a $13 \times 4$ table and insert the image of the back side card (*back.gif*) into all cells;
2. Assign a unique ID to each of the images, and set up an *onClick* event to call *selectCard*() with the ID as argument when the card image is clicked. It is suggested to use the image's filename (without extension) as the ID, e.g., 0 for Spade Ace (filename "0.png"), 1 for Spade 2 ("1.png"), etc.;

3. Create three buttons, namely "Stop Game", "Show Cards", and "Flip Cards", each with a unique ID;
4. Set up for each button an *onClick* event to call its corresponding function (as specified in Fig. 5) when the button is pressed;
5. Access to *gameArea* and *buttonArea* via their references (*document.getElementById*() with corresponding IDs) and set up the game environment via the *innerHTML* property.

A pseudocode is provided to help you draw the table for card display:

```
1. open a table;
2. for(each row $i$ from 0 to 3)//4 rows
3.     start a new row;
4.     for(each cell $j$ from 0 to 12)//13 cards per row
5.             add an image object to the cell and set $id = j + i * 13$, $src = back.gif$", and
   $onclick = selectCard()$ with the argument set by the $id$
6.     end for loop
7.     close the row
8. end for loop
9. close the table
```

## showCard()

The function accesses to all cards and turns them face-up one by one according to the Gutenberg Rule with animation effect which is the same as the western habit of reading: left-to-right,top-to-bottom. The function takes an argument as the ID of a card image on the table and returns nothing.

1. Get the reference to the image by using *document.getElementById*() and the ID (argument);
2. Turn the card face-up by changing the value in the image's *src* property with the correspond ing value in *preloadImages*.
3. Increment the ID and call back to *showCard*() itself to access next card. Use *setTimeout*() to delay the process and make an animation effect.

(*Hint: Refer to the imagePreloading exercise for hints. Note that the imagePreloading exercise is working on the same image object in global so it doesn't need to pass an ID in argument. This function, however, deals with multiple image objects and relies on the ID passed in argument to determine which image object it works on when being called.*)

## flipCard()

The function accesses to all cards and turns them face-down one by one if the cards are not identified as the found, paired cards. This might happen when the user was cheating during a session by clicking "Show Cards" button to display all cards and then clicked "Flip Cards" button to turn them back to face-down. The function takes an argument as the ID of a card image on the table and returns nothing.

1. Get the reference to the image by using *document.getElementById*() and the ID (argument);
2. Check if the ID is an element in *pairedCards* by calling to the *isPairedCard*() function;
3. If not in *pairedCards*, replace the value in the image's *src* property by "*back.gif*".
4. Increment the ID and call back to *flipCard*() itself to flip next card. Use *setTimeout*() to delay the process and make an animation effect.

(*Hint: Relying on the same strategy as that in showCard*().)

## isPairedCard()

The function checks if a card is in the collection of found paired cards. It takes an argument (the ID of the card) and returns a Boolean value, *true*, if the argument (the ID) occurs in *pairedCards*, or *false*, otherwise.

(*Hint: using the searching plan for presence of target value in an array*).

## sortCards()

The function takes an array as the argument and uses the BubbleSort algorithm to sort the elements in the array. Note that as in the playing card context, the sorting algorithm should be based on the rank of cards from Ace, 2, 3, ... 10, Jack, Queen, and then King. The function returns nothing.

(*Hint: using cycle position plan to calculate the rank of a card*).

## playGame()

This function starts a new gaming session by firstly shuffling and flipping all cards face-down and then starting up the timer. The function takes no argument and returns nothing.

1. In a loop access to each of elements in *preloadImages* and *cardSet*:
   - Assign the element of *preloadImages* an instance of *Image* object that holds the reference to a card image;
   - Assign the element of *cardSet* a unique value for the ID of a card (e.g., using the stepper value of loop).
2. Instantiate a new instance for *pairedCards* and *guessCards*, respectively;
3. Shuffle all cards by calling *shuffleCards*() and pass *cardSet* as the argument;
4. Turn all cards face down by calling *flipCard*() and pass 0 as the initial argument;
5. Reset the timer by calling *resetTimer*() and pass 60 as the argument for a new game.

## resetTimer()

Starting with 60 seconds, this function makes the timer ticking down second by second. The function will terminate the current session if time runs out. The function takes a *Number* value, say, *sec*, as the argument and returns nothing.

1. Get the reference to *titleArea* by using *document.getElementById*();
2. Reprint to *titleArea* the title and the value in argument as the remaining time in seconds;
3. Decrement the value in *sec* to count down the timer;
4. Call back to itself (*resetTimer*()) with the updated *sec* if the time has not yet run out (*sec* is greater than zero). Use *setTimeout*() to delay the process and make an animation effect;
5. Call *stopGame*() to terminate the current game if the time has run out (*sec* becomes zero).

## stopGame()

When the user clicks the "Stop Game" button or the timer reaches 0 seconds, the *stopGame*() function terminates the current game session, displays on an alert window the gaming result (the number of found pairs and the cards listed in a sorted order), and then makes the game ready for next session.

1. Stop the timer by using *clearTimeout*().
2. Sort the cards in *pairedCards* by calling to *sortCards*(). (*Hint: You may need to create a new array to store the cardSet*[*ID*] *values where ID is an element in pairedCards*).
3. Generate and alert the "result" message using the sorted cards. (*Hint: Cycle Position Plan; using "\n" (newline indicator) to break a line in string*).
4. Make the game ready for next session by calling *playGame*().

## shuffleCards()

This function takes an array as the argument and randomises order of the elements in the array by using the shuffling plan. The function returns nothing.

## selectCard()

The function turns the card face-up when the user clicks a flipped, face-down card. In each of the second click (then two cards are shown), the function adds the cards to the *pairedCards* array if they are in a pair (with the same rank, ignoring suites) or flips the cards back to face-down again, otherwise. The function takes an argument as the ID of user clicked card (the card image on the table) and returns nothing.

1. If the card with ID (the argument) is not occurred in *pairedCards*, do Step 2-5:
2. Get the reference to the image using *document.getElementById*() and the ID (the argument);
3. Turn the card face-up by changing the value in the image's *src* property to the corresponding value from *preloadImages*.
4. Add the ID (the argument) into the *guessCards* array if this is the first card in selection, or, this is a different card from the first one if it is the second card in selection;
5. If the size of *guessCards* becomes two, check if the two cards have the same rank (*Hint: Cycle Position Plan for cardSet*[*ID*]);
   - If yes, add the cards (their IDs) into the *pairedCards* array;

- If not, flip the cards over to face-down after while, say, 300 milliseconds. (Get the reference to the images by using *document.getElementById*() and their IDs stored in *guessCards* and replace the value in the image's *src* property by *"back.gif"*. Using *setTimeout*() to make the animation effect);
- In either yes or not, make *guessCards* ready for next two cards in selection by re-instantiating *guessCards* (call to the constructor of *Array* again with *new* keyword).

## How Cards are Sorted and Shuffled

To help you understand how the cards are sorted and shuffled, here are some hints:

- Each element in *cardSet* corresponds to a cell (HTML image object) on the table in *gameArea*. The cells (HTML image objects) on HTML table are with IDs 0, 1, 2, 3, ..., 51, whereas *cardSet* also has 52 elements (indexed from 0, 1, 2, 3, ..., 51 as an array).
- The elements in *cardSet* refer to the image sources stored in *preloadImages*. While in initialisation, *cardSet* elements are assigned by values 0, 1, 2, 3, ..., 51, respectively, whereas card images are named (and stored in *preloadImage* array with index of) 0, 1, 2, 3, ..., 51 as well.
- Based on the first two points, after initialisation of *cardSet* the cards will be in sorted order (see Sub-figure (i) in the following Figure). Note that display of all cards in sorted order is not required in the assignment, but certainly you can force it show up by print-line using dummy testing statements.
- When the program shuffles the cards, it actually shuffles the values in *cardSet*. As a result, the corresponding image sources to the cell (HTML image object) are also shuffled (see Sub-figure (ii)), and the cards are then displayed in randomised order.

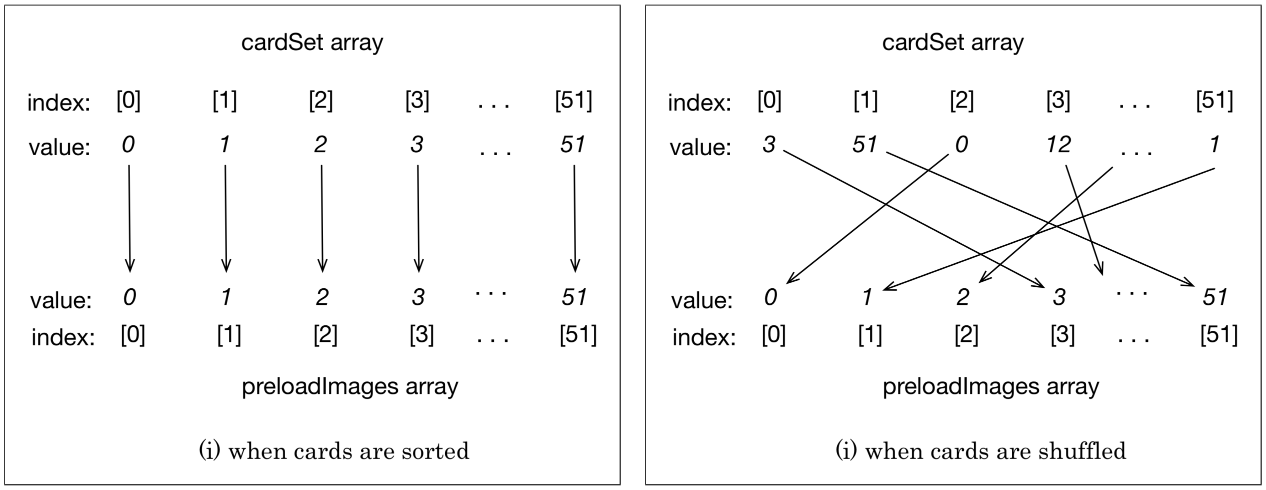

cardSet array
index: [0] [1] [2] [3] . . . [51]
value: 0 1 2 3 . . . 51
value: 0 1 2 3 . . . 51
index: [0] [1] [2] [3] . . . [51]
preloadImages array
(i) when cards are sorted
cardSet array
index: [0] [1] [2] [3] . . . [51]
value: 3 51 0 12 . . . 1
value: 0 1 2 3 . . . 51
index: [0] [1] [2] [3] . . . [51]
preloadImages array
(i) when cards are shuffled

Figure 6: Concept of Sorting and Shuffling Cards

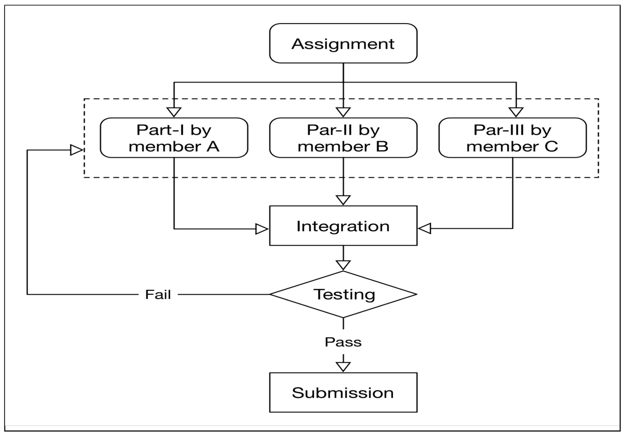
## Appendix II - Team-working Related Issues

### Team Registration

Express to the "*Assignment 3 Forum*" or "Social Forum" your interest in either recruiting people to form a team or joining a team with vacant positions. Once you have a team formed, please fill in the "*Team Registration Form*", which is available on StudyDesk, and email it to the Course Examiner. You will receive a confirmation message in replied email. Also as a sign of registration completion, you should find your team name on the "*Registered Teams*" table under Assignment3 on StudyDesk, as well private forum and chatroom opened for your team discussions.

### Team-working Styles

The following figures illustrate two styles in team-based programming working. You may choose either one of them or rely on any other styles as long as it suits your team.



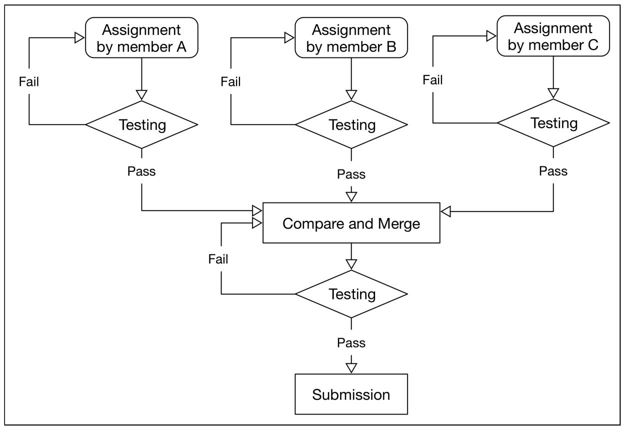


Figure 7: Team-working Styles

### Team Conflict Resolution

It is observed in many projects that a team may experience conflict or even crisis during teamwork, because members are individuals with different backgrounds and personalities. While trying to resolve a conflict in your team, please follow the 2-Stages strategy:

**Stage I - Internal Resolution** Try to resolve the conflict within the team. Figure 8 illustrates the steps to conflict resolution – a variation of the model introduced by Joseph Phillips in 2010 [4] – that you should take while trying to resolve a conflict.

[4]Joseph Phillips (2010). IT Project Management: On Track from Start to Finish, Third Edition, Chapter 7 - and Chapter 8 - Managing Teams. McGraw-Hill Osborne Media.

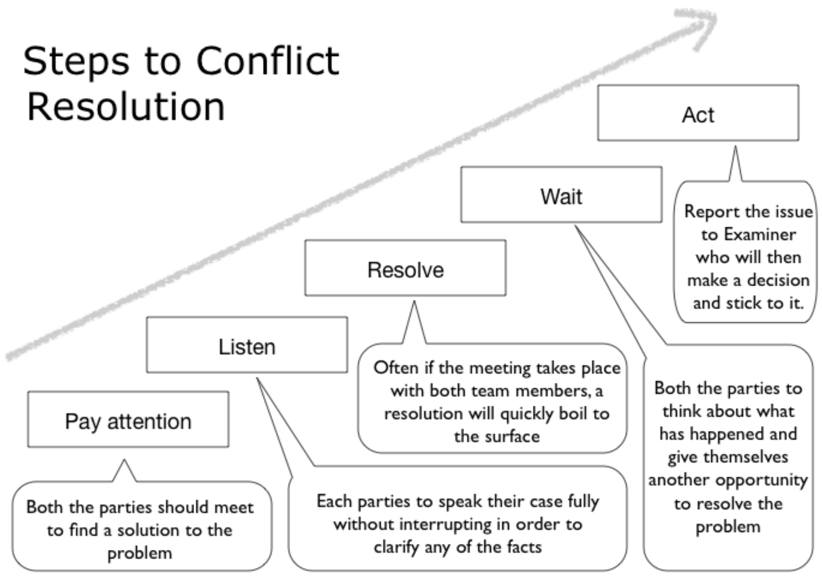
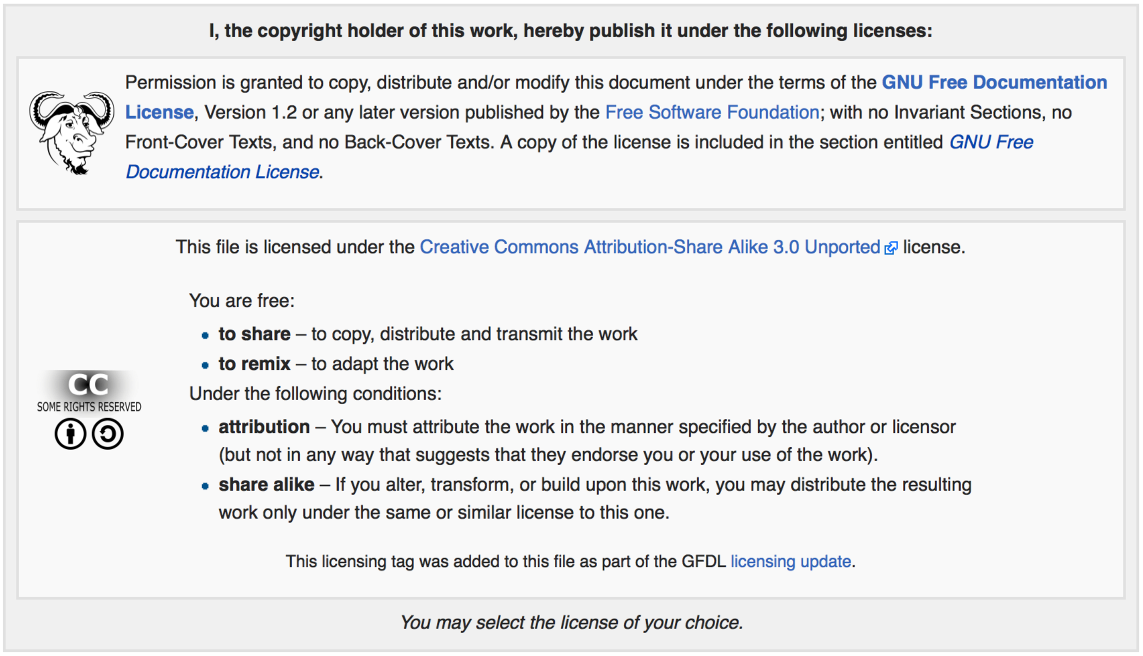

Steps to Conflict Resolution
Act
Report the issue to Examiner who will then make a decision and stick to it.
Wait
Both the parties to think about what has happened and give themselves another opportunity to resolve the problem
Resolve
Often if the meeting takes place with both team members, a resolution will quickly boil to the surface
Listen
Each parties to speak their case fully without interrupting in order to clarify any of the facts
Pay attention
Both the parties should meet to find a solution to the problem

Figure 8: Steps to Conflict Resoution

**Stage II - Report to Course Examiner** You should upgrade the action to Stage II only after all means of effort in Stage I have been taken and failed. While it is necessary, please fill in the "*Crisis Report*", which is available on StudyDesk, and send it to the Course Examiner with supportive evidence such as meeting minutes and communication records. **Note that a report without supportive evidence will be declined because it may contain bias and puts somebody else in an unfair, disadvantaged position**. The Course Examiner will then be engaged and based on investigation, make a final decision.

## Acknowledgement

The card images in this assignment are collected from Wikipedia[5], and distributed and re-used within the CSC1401 class under the following license:



**I, the copyright holder of this work, hereby publish it under the following licenses:**

Permission is granted to copy, distribute and/or modify this document under the terms of the **GNU Free Documentation License**, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled *GNU Free Documentation License*.

This file is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported license.

You are free:

- **to share** – to copy, distribute and transmit the work
- **to remix** – to adapt the work

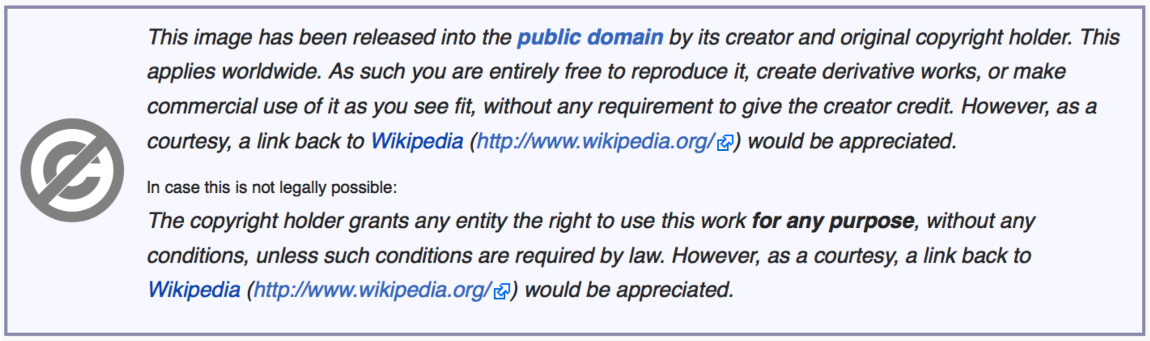
Under the following conditions:

- **attribution** – You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- **share alike** – If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

This licensing tag was added to this file as part of the GFDL licensing update.

*You may select the license of your choice.*

The image in Fig. 1 is collected from Wikipedia[6] and used under the following license:



*This image has been released into the **public domain** by its creator and original copyright holder. This applies worldwide. As such you are entirely free to reproduce it, create derivative works, or make commercial use of it as you see fit, without any requirement to give the creator credit. However, as a courtesy, a link back to Wikipedia (http://www.wikipedia.org/) would be appreciated.*

In case this is not legally possible:

*The copyright holder grants any entity the right to use this work **for any purpose**, without any conditions, unless such conditions are required by law. However, as a courtesy, a link back to Wikipedia (http://www.wikipedia.org/) would be appreciated.*

[5]http://en.wikipedia.org/wiki/Playing_card
[6]http://en.wikipedia.org/wiki/Concentration_%28game%29