

UNIVERSITY OF TORONTO
Faculty of Arts and Science
Midterm Exam – February 2007
CSC 309 H1 S
Instructor – Eyal de Lara
Duration – 1 hours

Examination Aids: One single-sided hand-written 8.5"x11" page containing notes

NAME _____

STUDENT NUMBER _____

PART I (Short Answers): _____ /40
PART II (Programming): _____ /60
Total: _____ /100

Part I of this examination is a series of short-answer questions.

Part II of this examination is a programming exercise.

Answer all questions in the spaces provided on this examination paper. There is no need to use more space than is provided. If you must, then use the back of the examination paper and so indicate in your answer.

General Advice:

- Skim through the entire exam before beginning your detailed work, to get a sense of where best to spend your time; if you get stuck on one question, go on to another and return to the difficult question later.
- Show your work, not just the final answer. Partial credit will be granted in cases that demonstrate correct reasoning, even if an error leads to an incorrect final result.
- For the programming exercise, if unsure of API details or language syntax details, try your best and include a comment indicating what you are attempting to achieve.

Good luck!

Part I – Short Answer Questions

1.- [20 points] Give two reasons why using the DOM structure (i.e., the application's user interface) for storing the logical state of a web application is a bad idea.

Makes the application hard to maintain and change.

DOM accesses are slower than accesses to data stored in native JavaScript objects.

2.- [10 points] Give two reasons for why good usability is more important for web applications than traditional desktop based applications.

Target user population is more broad, and may not be clearly defined.

Users don't have an economic investment in application and thus are more prone to switching to other sites.

3.- [10 points] What is the difference between HTTP, HTML and CSS?

HTTP is a transport protocol

HTML is a standard for structuring content.

CSS is a standard for formatting textual content.

Part II - Programming Question

4.- [60 marks] **Caution:** this is a complex programming question. Read all instructions in this and the following pages before answering.

Create a client-based card matching game. The game starts with an even number of cards facing down (Figure 1). The objective of the game is to find all the matching pairs. The game is played by turning over two cards at a time, first one (Figure 2) then the second one. If the two cards match (Figure 3), then they stay facing up. If they don't (Figure 4), they remain facing up for 2 seconds, and are then put back facing down. To flip a card, a user clicks on it with the mouse. The game ends when all matching cards have been found and all the cards are facing up (Figure 5).



Figure 1 Initial State



Figure 2 After Turning First Card



Figure 3 Cards Match



Figure 4 Cards do not Match

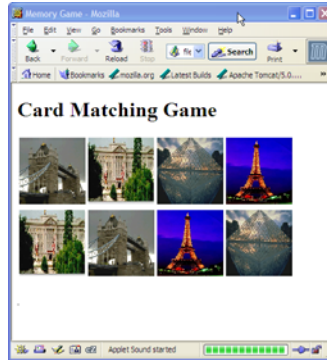


Figure 5 Game Over

In the following, we will refer to the rectangular area holding the cards as the *board*.

Your application should obtain information about the size of the board (i.e., number of rows and columns), the URL for images on the cards, and the specific placement of cards from a JavaScript file called **memory.js**. Bellow is a sample `memory.js` file. Note how the sample file has a different number of images than those shown on Figures 1 to 5. This is done to illustrate how your application should be able to handle boards of arbitrary size. You can assume that the board will always have a rectangular shape, i.e., all rows will have the same number of cards. Finally, for the back of the card use the image “back.gif”.

```
function card (id, url) {
    this.id = id;
    this.url = url;
}

var c1 = new card (1, "img1.gif");
var c2 = new card (2, "img2.gif ");
var c3 = new card (3, "img3.gif");

var row1 = [c1,c3,c2];
var row2 = [c3,c2,c1];

var game = [row1, row2];
```

} memory.js

To provide feedback to the user, your application should use a **provided** Java Applet called `Sound.class`, which exports the following interface:

```
class Sound {
    public void playMatch();
    public void playTryAgain();
}
```

When the two cards match, your application should call `playMatch()`. The Applet will then play the phrase: "Great! This is a match." When the cards don't match, your application should call `playTryAgain()`, which will utter the phrase: "Please try again." Note that these two Applet calls are non blocking.

Your answer to this question should only use JavaScript, DOM, XHTML, and the provided Sound applet.

What you **don't** need to do:

- Detect that the game is over.
- Randomize the position of cards. Board information should come from the JavaScript file.

Extra space for question 4

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//w3c//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<head>
<title>Memory Game</title>
<script type="text/javascript" src="memory.js">
</script>

<script type="text/javascript">

function mycard (status, id, url) {
  this.status = status;
  this.id = id;
  this.url = url;
}

var board = new Array;

var card1 = null;
var card2 = null;

var img1 = null;
var img2 = null;

var wait = false;

var timerid = null;

function timerHandler() {
  window.clearInterval(timerid);
  img1.src = "block.bmp";
  img2.src = "block.bmp";
  card1.status = 0;
  card2.status = 0;
  card1 = null;
  card2 = null;
  wait = false;
}

function flip(img,i,j) {

  if (board[i][j].status == 1) // clicking on a card that is up
    return;

  if (wait != false) // waiting to flip cards face down
    return;

  img.src = board[i][j].url; // replace image on clicked card
```

```

        board[i][j].status = 1;    // flip card up on shadow structure

        if (card1 == null) {
            card1 = board[i][j];
        }
        img1 = img;
    }
    else {
        card2 = board[i][j];
        img2 = img;
    }
}

var sound = document.getElementById("sound");

    if (card1.id == card2.id) {
        sound.playMatch();
        card1 = null;
        card2 = null;
    }
    else {
        sound.playTryAgain();
        wait = true;
        timerid = window.setInterval("timerHandler()",2000);
    }
}

function createBoard() {
    table = document.getElementById("myTable");
    for (var i=0; i < game.length; i++) {
        var tr = document.createElement("tr");
        var row = new Array();
        for (var j=0; j < game[i].length; j++) {

            row [j] = new mycard(0,game[i][j].id, game[i][j].url);

            var newImg = document.createElement("img");
            newImg.setAttribute("src","block.bmp");
            newImg.setAttribute("width","100px");
            newImg.setAttribute("height","100px");
            newImg.setAttribute("onclick","flip(this," + i + "," + j + ")");
            var td = document.createElement("td");
            td.appendChild(newImg);
            tr.appendChild(td);
        }
        board[i]=row;
        table.appendChild(tr);
    }
}
</script>
</head>
<body onload="createBoard();">

```

```
<h1>Card Matching Game</h1>
<table id="myTable">
</table>
<applet id="sound" code="Sound.class" width="3" height="3" />
</body>
</html>
```

Marking scheme

1. Header (5)

- proper header information
- load script memory.js

2. Proper initialization (5)

- initialization vars are properly declared

3. Board generation (10)

- dynamically get board info from javascript file (4)
- set card variables appropriately (4)
- upon click, set to call function to flip card (2)

4. Flipping card (15)

- change card image upon click (3)
- using variable: wait for 2nd card to be flipped after first is flipped open (3)
- validate: no action when clicked on already open card; matched cards not flipped down (3)
- check for matching of cards: reset card variables (3)
- check for mismatched cards (3)

5. Sound feedback (5)

- load applet (2)
- play match/no-match sounds appropriately (3)

6. Handling timer (10)

- using setInterval/clearInterval: display mis-matched pair of cards for 2 seconds (5)
- reset card variables and wait time variable (to wait for 2nd card to be flipped) appropriately (5)

7. Misc (10)

- reasonably legible writing (indentation..) (2)
- html structure (js functions) (5)
- html body: init. board on loading html, table layout (3)