The Assignment

We have to specify, design, document, implement, and test the classic 15-tile puzzle game in HTML, Cascading Style Sheets (CSS), and JavaScript (JS).

It will be impossible to write 15 pages about such a simple assignment, though. From start to finish, the process of designing, implementing, and testing the puzzle game took no more than 4 hours. Selecting, preparing, and polishing the picture took an extra hour. This report took another 4 hours to write.

A live version of everything is available from my own server at http://sdu.projectjj.com/NIS03/ (game is puzzle.html).

The Specification

Initial brainstorm of the assignment concluded that we need very little, if indeed any, CSS. For optimal cross-browser compability, CSS is best left out. I chose to develop and test with Mozilla Firefox 1.0PR, but the game has been tested and found working in Microsoft Internet Explorer 6.0 (with Windows XP Service Pack 2).

Another immediate thought was that we cannot save to the harddrive. JavaScript has no access to the file system whatsoever, and for good security reasons. But, JavaScript does have access to set, retrieve, and delete cookies, which is good enough. Using `String.split()` and `Array.join()` we can encode all the data about the players into a single cookie. That it was supposed to be for maximum 4 players seemed arbitrary, so I have disregarded that and not put a limit on the number of players. There can be as many players as can be encoded into the cookie, which is browser-dependent.

The assignment states that you 'select' two tiles to swap. This is a flawed solution since you can only swap with the empty tile. Instead, the player clicks the tile he/she wants swapped with the empty tile.

The Design

The game is split into 3 layers: Presentation (HTML), Application (JS), Storage (JS). Presentation is then strictly speaking further split into Data (HTML tags) and Styling (CSS). JavaScript is used for both the Application and Storage layer.

While it is technically possible to make very clear distinctions between these layers, it doesn't really make sense for such a simple game. It would slow down the overall execution and introduce possible incompatibilities.

The Implementation

The **Presentation** layer (puzzle.html) was made first. This file is the central controller for everything, and also sets the initial state of the game. The Application layer (js/game.js) will read the initial layout of the puzzle pieces and consider that layout the solution to the puzzle. It will then consider the 16th piece as the empty piece. This means that the pictures of the puzzle can be replaced, resized, and even loaded from a remote machine, without having to fiddle with JavaScript.

The **Application** implementation uses only very basic functionality. It uses the document.forms[id].elements[id] method of accessing dynamic elements, since this is the most cross-browser compatible method. The game board's current layout is stored in a one-dimensional array, which keeps references to the actual images.

When the page is first loaded, the InitGame() function is called. This function retrieves and unserializes the cookie with the players. If there are no known players, it will ask for the name of the first player.

Players are also stored in a one-dimensional array for ease of serialization/unserialization when storing/reading as a cookie. The format of this array is:

players[0] is the ID of the current player.
players[1] holds the name of player 0 (we count from 0).
players[2] holds the total games played by player 0.
players[3] holds the total moves made by player 0.
players[4] holds the name of player 1.
players[5] holds the total games played by player 1.
...so on, so forth.

Using players[0] to keep track of the current player ensures that even after a browser restart it will know who last played. The number of players is calculated on-the-fly with (players.length-1)/3. To get back to the data of the current player we access (players[0]*3)+1 through (players[0]*3)+3. We do not store the average moves per game since that can also be calculated on-the-fly.

When the Start New Game button is clicked the user is asked what name they want to play as, with the current name filled in as default. If you write in an existing name, your previous results are loaded. If you write a name that doesn't exist, a new player will be created with that name. The only way to see all existing players is to check the statistics. The game board will then be shuffled.

There is a seperate Shuffle button which will execute 25 random moves. The number 25 was chosen on instinct. It will shuffle in a way that a move made won't be reversed, as it would be pointless if the 25 random moves were spent on moving tile 15 to 16 and back. When using Start New Game, Shuffle does 200 moves, and the moves made do not count towards total moves in that game, but hitting the Shuffle button in an ongoing game will count those moves. This is a design choice, since sometimes a good shuffle can clean up the board.

I've implemented the shuffler to use the same move validator as when the user clicks a tile. This means that the user can follow the shuffling visually (at least on Firefox; IE seems to not redraw while in a loop). This makes the initial 200 moves rather slow on older machines, though, but nice to look at.

When a player clicks a piece the ClickPiece() function is called. This function checks whether one of the surrounding tiles is the empty one, and if it's empty it will be swapped. Very basic checks, but they do the job. After each piece is moved, the state is checked against the stored initial layout. If the state matches the initial layout, the puzzle is declared completed and the player's statistics are added and saved.
Note: While Start New Game will allow you to create a new player, nothing about that player is stored until you finish a game with that player. This is to prevent overflowing the cookie with empty players.

The **Storage** layer is very simple. For reading and writing cookies I use a library (js/cookie.js) that I once found and cleaned up. I forgot where I got it from, though, but I take no credit for it. The application specific storage (js/storage.js) uses the cookie library. Player data is serialized to text with Array.join(”,”) and unserialized back to an array with String.split(”,”).

The **Statistics** page has its Presentation (stats.html) and Application (js/stats.js) layers much more intermixed due to the dynamic nature of the table it has to show. The ShowStats() function will first find the maximum average, and then output everyones statistics relative to that, in the order of the player IDs.

The **CSS** (css/default.css) for the game is nearly non-existent. The file is self-explanatory, and can be omitted entirely without affecting the game in the least.

The Testing Phase

For testing I put a debug textarea in puzzle.html. Various outputs follows...

Output: InitGame()

```
Initializing...
http://sdu.projectjj.com/NIS03/images/1.png
http://sdu.projectjj.com/NIS03/images/2.png
http://sdu.projectjj.com/NIS03/images/3.png
http://sdu.projectjj.com/NIS03/images/4.png
http://sdu.projectjj.com/NIS03/images/5.png
http://sdu.projectjj.com/NIS03/images/6.png
http://sdu.projectjj.com/NIS03/images/7.png
http://sdu.projectjj.com/NIS03/images/8.png
http://sdu.projectjj.com/NIS03/images/9.png
http://sdu.projectjj.com/NIS03/images/10.png
http://sdu.projectjj.com/NIS03/images/11.png
http://sdu.projectjj.com/NIS03/images/12.png
http://sdu.projectjj.com/NIS03/images/13.png
http://sdu.projectjj.com/NIS03/images/14.png
http://sdu.projectjj.com/NIS03/images/15.png
http://sdu.projectjj.com/NIS03/images/16.png
http://sdu.projectjj.com/NIS03/images/1.png,http://
sdu.projectjj.com/NIS03/images/2.png,http://sdu.pro
jectjj.com/NIS03/images/3.png,http://sdu.projectjj.
com/NIS03/images/4.png,http://sdu.projectjj.com/NIS
03/images/5.png,http://sdu.projectjj.com/NIS03/imag
es/6.png,http://sdu.projectjj.com/NIS03/images/7.pn
g,http://sdu.projectjj.com/NIS03/images/8.png,http:
//sdu.projectjj.com/NIS03/images/9.png,http://sdu.p
rojectjj.com/NIS03/images/10.png,http://sdu.project
jj.com/NIS03/images/11.png,http://sdu.projectjj.com
/NIS03/images/12.png,http://sdu.projectjj.com/NIS03
/images/13.png,http://sdu.projectjj.com/NIS03/image
s/14.png,http://sdu.projectjj.com/NIS03/images/15.p
ng,http://sdu.projectjj.com/NIS03/images/16.png
0,Tino Didriksen,0,0
Game Initialized
```

Output: Finished a game

```
Clicked a R edge piece: 16
Clicked a D edge piece: 16
Completed!
0,Tino Didriksen,1,266
```

Outputting all 266 moves would not show anything special. Of importance, shown here is the serialized cookie before and after the game: "0,Tino Didriksen,0,0" versus "0,Tino Didriksen,1,266". Multiple players would be serialized as e.g. "0,Tino Didriksen,1,266,Other Player,2,430". Again, the leading 0 simply means that player 0 is the last to have completed a game.

The Conclusion

The assignment was completed without any problems along the way, and the result is a clean and relatively cross-browser friendly implement of the classic 15-tile puzzle.

The Credits

- The photo of the Siberian Tiger (material/siberian_tiger.psd) is Copyright 2003 Søren Skarby, used with permission.
- The font Celestial (material/Celestial.ttf) is Copyright 2000 Neale Davidson, used as freeware font.
- The cookie library (js/cookie.js) is from an unknown author, cleaned up by myself.

Tino Didriksen