Università degli Studi di Torino

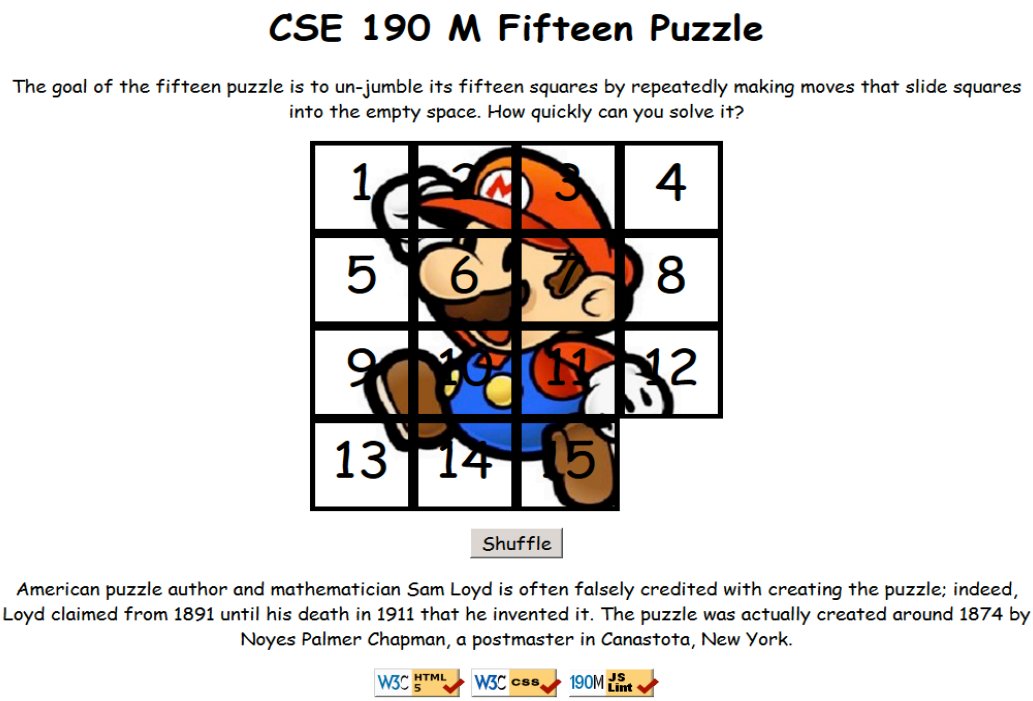
Tecnologie Web (6 cfu) - MFN0634

Consegna 05 - JS e DOM

Testo del compito

Lo scopo di questo esercizio è di prendere pratica con il Document Object Model (DOM) e gli eventi in JavaScript.

L'esercizio consiste nel riproporre il gioco del Puzzle a 15 pezzi ("Fifteen Puzzle"), un semplice gioco formato da una griglia di 4x4 elementi numerati, di cui uno mancante. L'obiettivo del gioco è spostare i pezzi sfruttando gli spazi liberi, fino a ricostruire la figura del puzzle.



Voi dovrete scrivere il codice CSS e JavaScript per la pagina fifteen.html che esegue il gioco "Fifteen Puzzle". Potrete anche scegliere l'immagine di background visualizzata sui tasselli del puzzle.

Utilizzate i seguenti files:

fifteen.js – contiene il codice JavaScript per la pagina web

fifteen.css — contiene lo stile CSS della tua pagina web

L'immagine a scelta deve chiamarsi:

background.jpg — dev'essere un'immagine abbastanza grande per un puzzle di dimenzioni *400x400px*

Non servirà creare altri file html, ma basterà utilizzare quello fornito dal presente esercizio che non va modificato. Scaricate il file .html e scrivete il codice JavaScript per lavorare con i file.

Scriverete il codice JavaScript in modo che faccia riferimento al DOM per cambiare la visualizzazione della pagina e cambiare lo stile degli elementi. Utilizzerete opportunamente il settaggio di classi, ID, e/o le proprietà di stile.

(Suggerimento: inizialmente collocate i pezzi nelle posizioni corrette 1,2,3,4,5,9, … e 13 e non toccatele più. Quello che rimane da risolvere è una scacchiera 3x3 che è più facile da gestire).

Specifiche della visualizzazione

Tutte le parti testuali della pagina sono scritte in corsivo con una dimensione di *14pt*. Tutto il contenuto è centrato, compreso l'heading in alto, i paragrafi, il puzzle, il bottone di shuffler e i pulsanti W3C in basso.

Al centro della pagina ci sono 15 tasselli che rappresentano il gioco del puzzle, il quale occupa complessivamente *400x400* pixels (centrato orizzontalmente). Ogni pezzo del puzzle occupa complessivamente *100x100* pixels, e siccome ciascuno dei 4 bordi sono occupati da un bordo nero di *5px*, la parte interna di ciascun pezzo occupa *90x90* pixels.

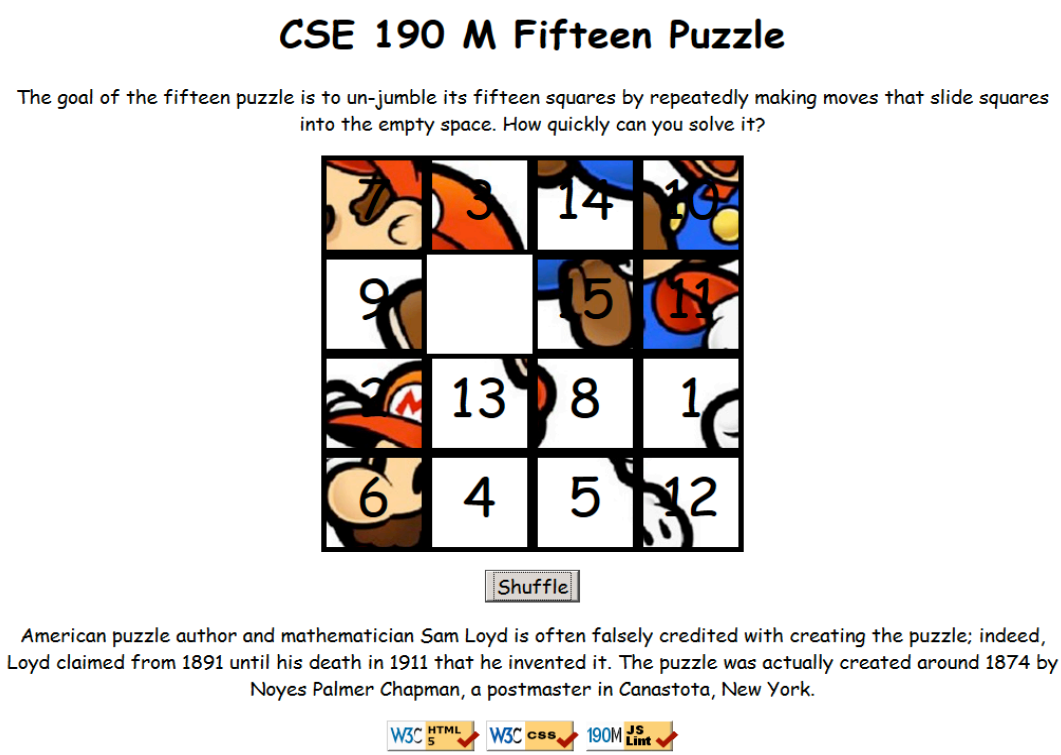
I pezzi sono numerati da 1 a 15, con un font di *40pt*. Al carimento della pagina, vengono disposti in modo corretto con il tassello mancante in basso a destra. Ogni pezzo visualizza una parte dell'immagine background.jpg, da collocare nella stessa cartella della pagina.

La parte di immagine visualizzata è legata al numero del pezzo: quello con il numero "1" visualizzerà la porzione di *100x100* in alto a sinistra. Il pezzo numero "2" mostrerà la parte adiacente dell'immagine di dimensione *100x100px* e così via.

(Nota bene: L'immagine di sfondo scelta appare sui pezzi settandola come *background-image* di ogni pezzo. Cambiando il valore di *background-position* di ciascun div, si può mostrare una parte diversa del background di ogni pezzo.

Un elemento che potrebbe generare confusione, a proposito di *background-position*, è il fatto che i valori di x/y spostano il background sull'elemento, ma non spostano l'elemento stesso. Gli offset sono poco intuitivi e necessitano di particolare attenzione. Per esempio, se si vuole un *div* di *100x100px* per mostrare l'angolo in alto a destra di un'immagine di *400x400px*, occorre settare la *background-position* con i valori *-300px 0px*.)

Al centro, sotto il puzzle, si trova un button con scritto **Shuffle** che può essere cliccato per rimescolare le tessere del puzzle. Lo screenshot è relativo a Firefox su Windows e potrebbe cambiare in altri ambienti.



Specifiche sul comportamento

Quando il pulsante viene premuto sopra un pezzo, questo si sposta nella casella libera. Se non ci sono caselle libere, il pezzo non si muove. Analogamente, se si preme una cella libera non succede nulla.

Quando il mouse passa sopra un pezzo che si può muovere, avendo una cella adiacente libera, i suoi bordi e il colore del testo devono diventare rossi. Quando il mouse esce dai bordi del pezzo, questo deve tornare normale. Per una cella vuota o su un pezzo che non si può muovere, non ci sarà alcun cambiamento. (Usare *:hover CSS pseudo-class*.)

Quando si preme il bottone Shuffle, le tessere sono mescolate casualmente. Si noti come alcune posizioni iniziali potrebbero non avere soluzioni. Vi suggeriamo quindi di generare uno stato iniziale che sia risolvibile, adottando un'opportuna strategia di mescolamento che muova a ritroso i pezzi scegliendo una posizione libera adiacente in modo casuale per un numero di volte ragionevole. Cercate di pensare ad un algoritmo efficiente!

Il gioco non prevede particolari eventi in caso di vittoria. Potete decidere di far visualizzare una finestra di pop-up (con un **alert** box) per congratularsi con l'utente o aggiungere altre parti opzionali. In alternativa (e molto meglio in termini di usabilità), invece di un alert box, potete visualizzare a mandare un messaggio di 'game over' e congratulazioni su un elemento del dom che viene mostrato solo dopo la vittoria e può sparire alla richiesta di 'new game' da parte dell'utente.

Strategia di sviluppo

Siccome il compito potrebbe non essere banale, suggeriamo di seguire questa strategia di sviluppo:

• In primo luogo, far apparire il puzzle nelle posizioni corrette senza nessuna immagine di background

• Aggiungere le parti corrette del background su ogni tessera

• Scrivere il codice che muove un pezzo, quando è cliccato, dalla sua posizione ad una cella qualsiasi purché vuota. Inizialmente, non preoccupatevi di controllare che ci sia una cella vuota adiacente.

• Scrivere il codice per decidere se una tessera si può muovere o no (se la cella adiacente è libera)

• Evidenziate la cella che si può muovere al passaggio del mouse. Tenete sempre traccia della cella libera.

- Scrivere l'algoritmo di mescolamento iniziale. Si suggerisce di scrivere dapprima il codice per fare una sola mossa casuale. Quindi, quando il bottone Shuffle viene premuto, eseguire questa mossa diverse volte, mediante un ciclo.

- Usate il debugger installato sul vostro browser tra gli strumenti dello sviluppatore. Eseguendo passo passo il codice js, dovreste essere in grado di trovare errori, anche quelli di tipo logico.

- Non usate finestre di pop-up per mostrare il risultato temporaneo delle variabili. Piuttosto usate console.log().

Spunti per l'implementazione:

• Utilizzate un posizionamento assoluto per settare le coordinate x e y di ogni pezzo del puzzle. L'area generale deve avere una posizione relativa affinché gli offset di ogni pezzo siano relativi rispetto all'area del puzzle.

• Convertite da stringa in valore numerico con **parseInt.** Questo funziona anche per stringhe con cominciano con un numero, ma che terminano con un contenuto non numerico. Per esempio, parseInt("123four") restituisce 123.

• Si deve includere un po' di **casualità** nell'algoritmo di mescolamento. Il numero casuale intero può essere generato da 0 a N oppure essere scelto casuamente tra N opzioni, con **parseInt(Math.random() * N)**

• Suggeriamo di non utilizzare un div esplicito per rappresentare le caselle vuote. Tenete traccia di dove sia, per riga e colonna e sia per x/y, ma non

creare un elemento apposta. Suggeriamo anche di non memorizzare i pezzi del puzzle in un array di due dimensioni, sarebbe difficile aggiornarlo quando si sceglie di muovere un pezzo.

• Si ricorda di usare funzioni per evitare di avere codice ridondante: sia per le operazioni comuni, come nel caso si voglia far muovere un pezzo, e sia per determinare se un pezzo si può muovere o meno.

• Vi servirà dover accedere a un punto del DOM per una cella in una particolare riga/colonna o posizione x/y. Suggeriamo di scrivere una funzione a cui passare come parametri una riga e una colonna, per restituire il DOM object della cella corrispondente. Potrà servire assegnare un **id** a ogni cella (*tile*), come ad esempio "tile_2_3" per la cella in seconda riga e terza colonna. In tale modo sarà poi facile accedere a tale casella tramite javascript. Quando viene mosso un pezzo, occorre quindi aggiornare il suo id con la nuova posizione.

Usate questi **id** in modo appropriato. Non spezzate la stringa "tile_2_3" per estrarre i valori 2 o 3 da essa. Invece, usate gli **id** in modo opposto: per accedere al DOM per la cella alla riga 2, colonna 3, costruite un **id** stringa di nome "tile_2_3".

[Solo se sapete usare JQuery] Potete usare la funzione **$** per trovare il pezzo nelle coordinate corrette x/y.

Implementazione e valutazione:

Vi ricordiamo che consegnare gli esercizi di laboratorio serve solo ed unicamente a fornirvi un servizio di correzione, per cercare il più possibile di farvi capire cosa sbagliate ed in modo che possiate migliorare. Non assegneremo delle penalità a chi non li consegna, dato che il voto finale sarà dato in base alla correttezza e alla completezza del progetto e della relazione che ne descrive le specifiche (maggiori dettagli verso la fine del corso). Non controlleremo in questa fase se copiate oppure no: se copiate danneggiate solo voi stessi perché non approfitterete del servizio fornito. Un compito perfetto ci fa piacere perché ci fa perdere meno tempo, un compito molto incompleto e pieno di errori eviteremo di correggerlo per evitare di perdere troppo tempo. In ogni caso, anche in questi casi estremi, non ci faremo influenzare in fase di valutazione finale.

Il codice CSS dev'essere validato con il W3C CSS validator.

Il file .js deve essere il piu' possibile privo di errori (usate gli strumenti dello sviluppatore del vostro browser anche per il debugging).

Usate poche variabili globali ed evitate codice ridondante, impiegando anche opportuni parametri e la restituzione di valori in modo appropriato. Le operazioni più frequenti devono diventare funzioni, per evitare che il codice diventi troppo complicato e lungo. Potete utilizzare la keyword nel vostro event handlers.

Sono consentite alcune variabili globali, ma non troppe: si devono usare soprattutto variabili locali. Se una variabile è usata spesso, dichiaratela come "costante" definendola in lettere maiuscole (es: IN_UPPER_CASE) e usandola nel codice. Non salvate come variabili globali gli oggetti DOM, come nel caso dei

valori restituiti da **$** o dalla funzione **document.getElementById.** Ad esempio, una buona soluzione di questo codice avrà soltanto 4 variabii: la riga e colonna della cella vuota (settate inizialmente entrambe a 3), il numero di righe e colonne nel puzzle (4), e la larghezza/altezza di ogni pezzo (100). Le ultime due sono variabili costanti.

Commentate in modo adeguato il codice JavaScript. All'inizio nell'header descrivere il programma e inoltre descrivete le sezioni più complicate del vostro codice.

Formattate il codice mediante l'uso di spazi adeguati e dell'indentazione. Usate nomi delle variabili che siano intuitive ed evitate linee di codice troppo lunghe. Un buon file .js, in questo caso, potrebbe avere circa 100 linee mentre il CSS ne porebbe avere circa 35.

Separate la parte di **content** (HTML), dalla **presentation** (CSS), e dal **behavior** (JS). Il codice JS dovrebbe usare stili e classi del CSS, piuttosto che settare ogni proprietà in JS. Per esempio, al posto di settare lo style o un DOM, usate una **className** e definite lo stile di tale classe nel file CSS. Al contrario, le proprietà degli stili relativi alle posizioni x/y dei pezzi e i relativi sfondi non si riescono a collocare nel file CSS, quindi potete metterli nel codice JS.

Usate in modo corretto JavaScript in modo da non avere altro codice, né funzioni onclick o altro, incluso nel codice HTML: adottate uno stile JS "discreto" (unobtrusive).

Infine, caricate tutti i file su una cartella **lab05** sotto il vostro progetto tweb su gitlab2. Non dimenticate di caricare anche i file fifteen.js, fifteen.cs, fifteen.html e background.jpg anche se non li avete modificati. Potrete poi controllare il vostro sito semplicemente usando la URL:
http://localhost:8080/tweb/lab05/

(ovviamente la porta ed il path possono cambiare da una configurazione all'altra; inoltre qui non ncessarie sono parti server e potreste anche usare il browser da solo).

Ricordate infine che il compito dovrà essere consegnato esclusivamente su gitlab2 e che su moodle dovete indicare solo la URL del vostro repo, come ad esempio:

https://gitlab2.educ.di.unito.it/ruffo/tweb.git

Se lo avete già fatto la volta scorsa non c'è bisogno di rifarlo: noi faremo semplicemente un aggiornamento del progetto che avremo nel frattempo già clonato. Ricordate che le **scadenze** vengono date per consentirvi di seguire al meglio la correzione generale che verrà pubblicata in seguito. Noi faremo solo un controllo, alla scadenza, se sono state rispettate o meno. In ogni caso, questo non pregiudica la possibilità per le studentesse e gli studenti di ottenere il massimo dei voti all'esame. Inoltre ricordate di dare l'autorizzazione all'accesso ai Proff. Botta e Ruffo.