AZUL

ITIS MARIO DELPOZZO CUNEO

Collaboratori del progetto:

Martino Filippo

Simone Multari

Giuseppe Martini

Elia Peyrachia

Alessio Mazzei

William Lecca

Introduzione al progetto

Principali caratteristiche

Abbiamo individuato le principali caratteristiche che il progetto doveva presentare, basandosi sulle richieste presentate dal committente:

Essendo un gioco da tavolo, è inutile perdersi in interfacce troppo complicate e macchinose: la semplicità è la chiave di questo genere di passatempo. La combinazione di HTML CSS e Javascript è stata scelta come la migliore in questo senso, ma va sottolineato che viste le dinamiche del sistema di gioco, l'uso di script si è rivelato essenziale per la rielaborazione della GUI a seconda delle mosse fatte dai giocatori.

Per semplificare la comprensione della struttura e dell'organizzazione dei vari file html, javascript e CSS relative alle singole componenti, qui di seguito presentiamo un albero dei vari file e cartelle:

La struttura più complessa è quella del menù: per rendere più leggibile il programma ci siamo assicurati di racchiudere i file relativi alle singole voci in sezioni diverse.
I file "style.css" serviranno a configurare la predisposizione e l'aspetto delle pagine, "bottone.css" dal canto loro serviranno invece a determinare come i bottoni si presenteranno graficamente all'utente.
Ovviamente non possono mancare i file html: tra questi spunta index.html, utilizzato per il reindirizzamento ai link delle singole voci, in base ai click descritti nel javascript.

Possiamo invece osservare come espositore, piattaforma e tavolo, componenti principali dell'interfaccia grafica, siano composte semplicemente dal classico file html e dal relativo javascript.

Inoltre, tavolo contiene il file "classe.js" che descrive al suo interno tutte le strutture dati delle componenti che il tavolo stesso andrà ad ospitare.

Menu Principale

Ecco come si presenta il menù principale: semplice e pulito, esso presenta le classiche voci Gioca-Regolamento-Tutorial-Impostazioni-Esci.

Menu Principale

```
<link rel="stylesheet" type="text/css" href="style.css"> <!--Includo lo style.css , ovvero lo stile dell'interfaccia del menu-->
<link rel="stylesheet" type="text/css" href="bottone.css"><!--Includo bottone.css , ovvero lo stile dei bottoni del menu-->

<script type="text/javascript">

    function close_window() {
        if (confirm("Sei sicuro di voler uscire dal gioco?")) {
            window.close();
        }
    }


</script>
```

Prima parte di index.html dove vengono richiamati i fogli di stile e tramite javascript si configura la possibilità di chiudere il gioco grazie al pulsante.

Menu Principale

```
<body onload="">
        <div class="menu-container">
            <!--Tramite svg creo una zona rettangolare di dimensione finita entro si trovano i bottoni-->
            <svg viewBox="0 0 100 100"><rect width="100" height="100"></rect></svg>

            <!-- tramite le classi dello style.css vado a definire lo stile del menu, e con i link si potrà passare alla prossima schermata-->
            <div class="menu">
                <div class="menu-item-title">Menu</div>
                <div class="menu-item">
                    <a href="Modalita/Modalita.html">GIOCA</a>
                </div>
                <div class="menu-item">
                    <a target="_blank" href="Regolamento/Azul_Regolamento.pdf">REGOLAMENTO</a>
                </div>
                <div class="menu-item">
                    <a href="#">TUTORIAL</a>
                </div>
                <div class="menu-item">
                    <a href="Impostazioni/Impostazioni.html">IMPOSTAZIONI</a>
                </div>
                <div class="menu-item" >
                    <a href="#"onclick="close_window()">ESCI</a>
                </div>
            </div>
        </div>
</body>
```

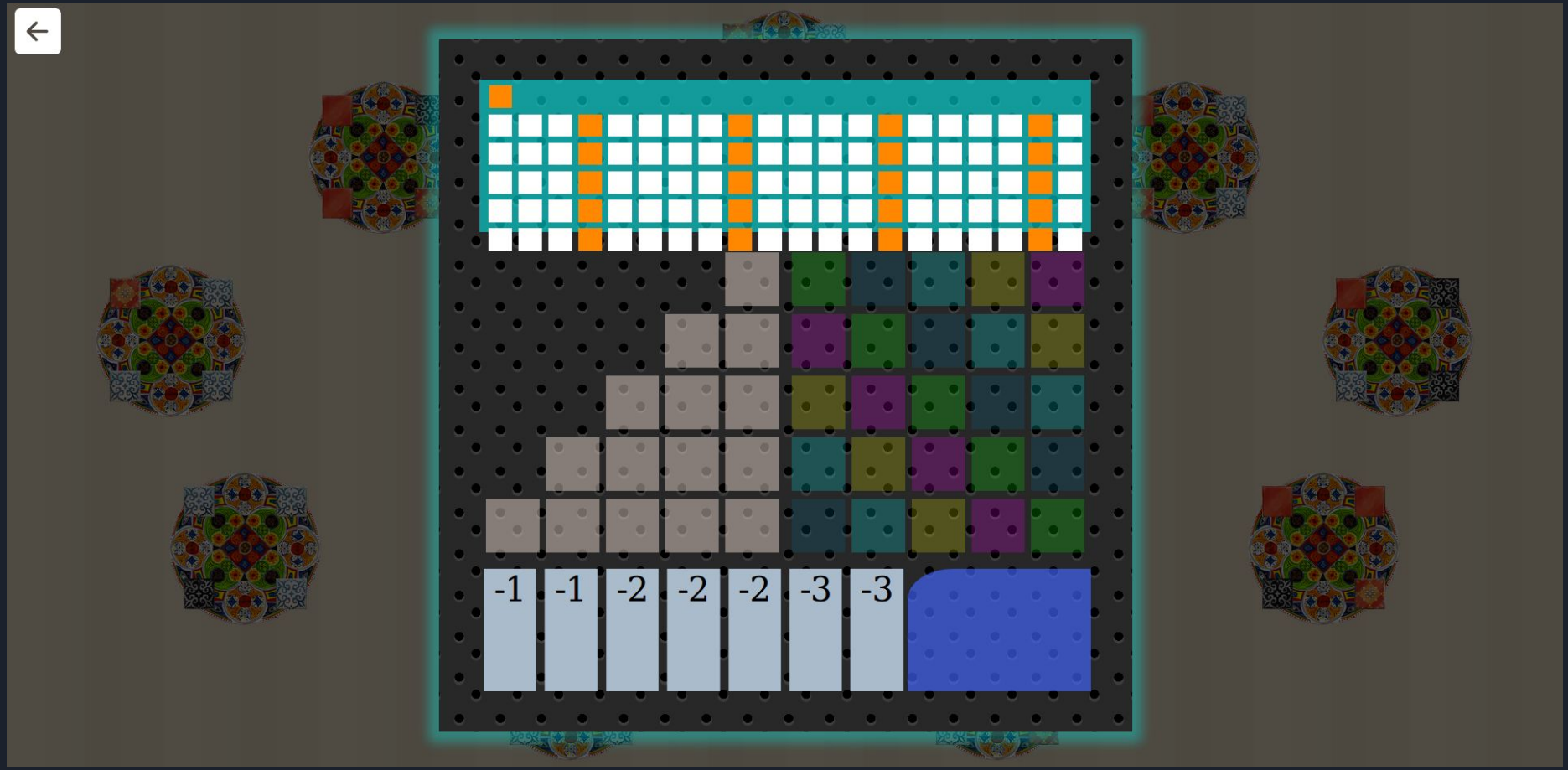
Seconda parte del file: qui si impostano i riferimenti ai link a seconda della scelta dei pulsanti:
Si potrà infatti accedere alla sezione delle modalità di gioco, al regolamento e al tutorial (Non ancora implementato ma presto disponibile nella prossima release, insieme alla modalità Person-Versus-Computer)

Introduzione all’area di gioco

Veniamo ora alla parte più importante: l’area di gioco e la modalità PVP(Player Versus Player):

Dal menù principale accedendo alla sezione “Gioca” sarà infatti possibile selezionare la sopra citata modalità, questo re-indirizzerà l’utente alla pagina html “tavolo.html”.

Quello che troveremo davanti sarà una semplice schermata con le varie piastrelle predisposte e il tabellone per i punteggi e la scelta delle mosse.


-1
-1
-2
-2
-2
-3
-3

Tavolo:

Cliccare le piastrelle o la piattaforma centrale permetteranno l'ingrandimento di questi ultimi, in modo da poter gestire al meglio le proprie mosse e i propri turni, osservando il campo da gioco proprio e quello dei propri avversari.

Di seguito, il file HTML del tavolo da gioco:

```html
<!--    tavolo.html
        Questo documento HTML crea il tavolo da gioco con la piattaforma da gioco più N espositori (N cambia in base al numero di giocatori).
        Per funzionare, bisogna importare:
        - i file comuni (style.css e script.js);
        - i componenti del tavolo (piattaforma.css, piattaforma.js, espositore.css ed espositore.js);
        - i file del tavolo (tavolo.css e tavolo.js).
-->

<html>
    <head>
        <!-- "Impostazioni" generali del documento HTML -->
        <meta charset="utf-8" />            <!-- Codifica del documento -->
        <title>AZUL</title>                 <!-- Titolo della finestra -->

        <!-- Importa i fogli di stile (i file con estensione "css") -->
        <link rel="stylesheet" type="text/css" href="style/espositore.css">
        <link rel="stylesheet" type="text/css" href="style/piattaforma.css">
        <link rel="stylesheet" type="text/css" href="style/tavolo.css">

        <!-- Importa gli script javascript (i file con estensione "js") -->
        <script type="text/javascript" src="script/classe.js"></script>
        <script type="text/javascript" src="script/script.js"></script>
        <script type="text/javascript" src="script/espositore.js"></script>
        <script type="text/javascript" src="script/piattaforma.js"></script>
        <script type="text/javascript" src="script/tavolo.js"></script>
    </head>
    <body>
        <div class="tavolo"></div>      <!-- In automatico, tavolo.js aggiunge a questo <div> il DOM necessario per creare il tavolo da gioco -->
    </body>
</html>
```

Tavolo:

La parte javascript è composta dalle funzioni che si occupano di inizializzare in base al numero di giocatori il campo da gioco e permettono loro di interagire con esso.

Le principali sono:

- setLayoutEspositore(espositore, nEspositore, nEspositori)
- creaElementoTavolo(tavolo, nGiocatori)
- aggiungiEventiTavolo(tavolo)

Tavolo:

Qui la funzione che si occupa della creazione degli elementi del tavolo, essa assegna infatti alle variabili locali “piattaforma” e “nEspositori” strutture dati basate sul numero di giocatori partecipanti, creando quindi la relativa GUI.

```
/* Funzione - Creazione e posizionamento degli elementi sul tavolo (piattaforma ed espositori) */
function creaElementoTavolo(tavolo, nGiocatori) {

    /* Variabile - Creazione dell'elemento con classe "piattaforma":
     *             --> è il figlio dell'elemento con la classe "tavolo"
     *             --> è un <div> (ELEMENTO_HTML)
     *             --> le assegno la classe "piattaforma" (CLASSE_PIATTAFORMA)
     *             --> modifico l'elemento appena creato dalla funzione con la funzione "setPiattaforma"
     */
    var piattaforma = creazioneComponente(tavolo, ELEMENTO_HTML, Classe.TAVOLO.PIATTAFORMA.nome, null, setPiattaforma);

    /* Variabile - Salvataggio del numero degli espositori da creare nel tavolo (dipende dal numero dei giocatori) */
    var nEspositori = (nGiocatori * 2) + 1;

    /* Ciclo - Creazione dell'elemento e del layout per gli N espositori */
    /* !TODO */
    for (var nEspositore = 0; nEspositore < nEspositori; nEspositore++) {
        var espositore = creazioneComponente(tavolo, ELEMENTO_HTML, Classe.TAVOLO.ESPOSITORE.nome, 'espositore-' + nEspositore, setEspositore);
        setLayoutEspositore(espositore, nEspositore, nEspositori);
    }
}
```

Piattaforma:

PUNTEGGIO

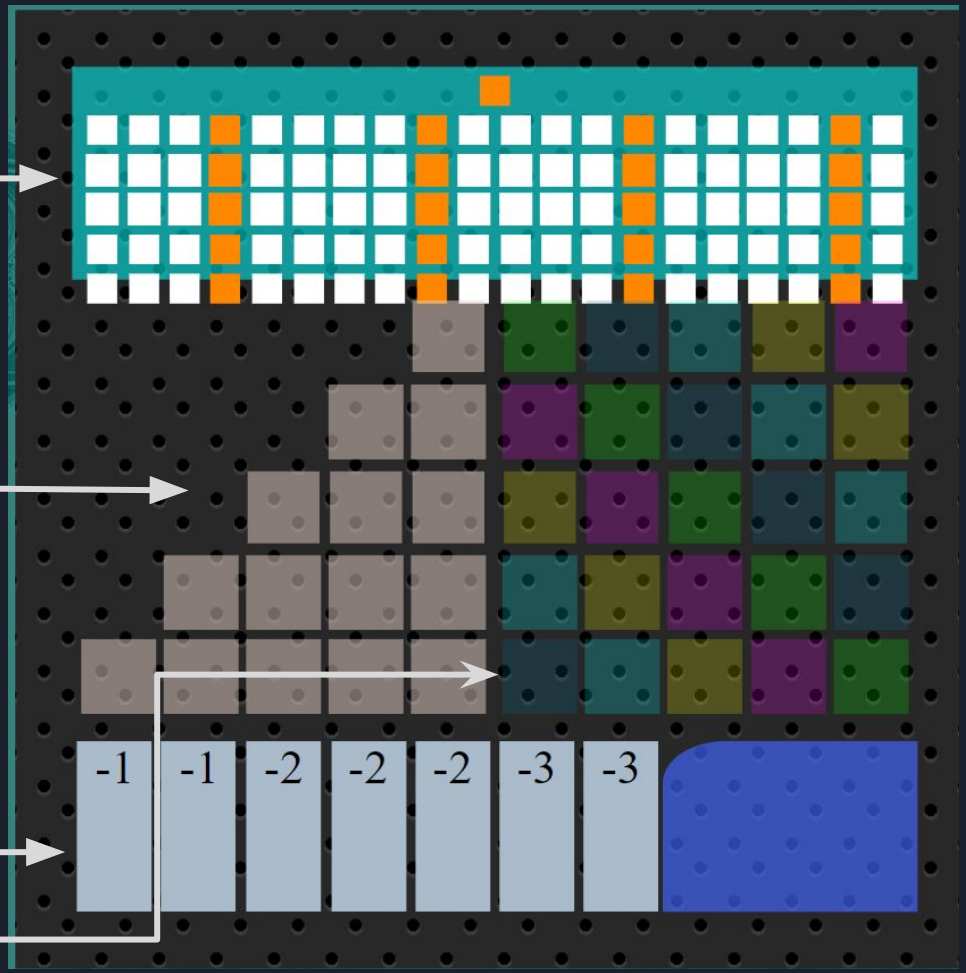
Composta da queste principali funzioni:

- setPiattaforma()
- setPunteggio()
- setRighe()
- setParete()
- setBasamento()
- aggiungiEventiPiattaforma()

RIGHE

BASAMENTO

PARETE



Piattaforma:

I File HTML della piattaforma centrale e degli espositori sono molto simili come struttura: entrambe infatti fanno riferimento ad un proprio javascript e ad un proprio foglio di stile.

Qui di seguito un esempio di funzione prima citate che permette la creazione della GUI del punteggio

```
/* Funzione - Creazione GUI per la sezione "punteggio" (parte superiore) */
function creaPunteggio(punteggio) {
    var idCella = 0;

    var elemRiga = null;
    var elemCella = null;

    elemRiga = creazioneComponente(punteggio, ELEMENTO_HTML, Classe.PIATTAFORMA.PUNTEGGIO.RIGA.nome, null, null);
    elemCella = creazioneComponente(elemRiga, ELEMENTO_IMMAGINE, Classe.PIATTAFORMA.PUNTEGGIO.CELLA.nome, null, null);
    elemCella.id = "punteggio-" + idCella++;
    elemCella.src = IMMAGINE_SFONDO_PUNTEGGIO_ETICHETTA;
    elemCella.innerHTML = idCella;

    for (var nRiga = 0; nRiga < N_RIGHE; nRiga++) {
        var elemRiga = creazioneComponente(punteggio, ELEMENTO_HTML, Classe.PIATTAFORMA.PUNTEGGIO.RIGA.nome, null, null);

        for (var nCella = 0; nCella < N_COLONNE_PUNTEGGIO; nCella++) {
            var elemCella = creazioneComponente(elemRiga, ELEMENTO_IMMAGINE, Classe.PIATTAFORMA.PUNTEGGIO.CELLA.nome, 'punteggio-' + idCella++, null);

            if ((idCella % 5) == 0) {
                elemCella.src = IMMAGINE_SFONDO_PUNTEGGIO_ETICHETTA;
                elemCella.innerHTML = idCella;
            } else {
                elemCella.src = IMMAGINE_SFONDO_PUNTEGGIO;
            }
        }
    }
}
```

Espositore:

Elemento sul quale verranno generate le tessere che il giocatore potrà decidere di pescare turno per turno..

La generazione degli espositori è gestita dallo script tavolo.js, che dopo aver calcolato il numero di espositori da inserire li genera mediante la funzione creazioneComponente()

```
/* Variabile - Salvataggio del numero degli espositori da creare nel tavolo (dipende dal numero dei giocatori) */
var nEspositori = (nGiocatori * 2) + 1;

/* Ciclo - Creazione dell'elemento e del layout per gli N espositori */
/* !TODO */
for (var nEspositore = 0; nEspositore < nEspositori; nEspositore++) {
    var espositore = creazioneComponente(tavolo, ELEMENTO_HTML, Classe.TAVOLO.ESPOSITORE.nome, 'espositore-' + nEspositore, setEspositore);
    setLayoutEspositore(espositore, nEspositore, nEspositori);
}
```

Espositore:

L’elemento sul quale abbiamo costruito l’espositore è un <DIV>

Lo scopo del codice dell’espositore è quello di creare dei “figli”, le piastrelle, posizionarle nel modo più simmetrico possibile all’interno della propria struttura e di impostare correttamente gli eventi a cui saranno soggette, primo su tutti il click dell’utente che andrà a sceglierle

Piastrelle:

Le piastrelle possono essere di 5 diversi tipi, riconosciute e gestite a livello di script da numeri da 0 a 4, a cui successivamente viene aggiunto un id; allo stesso modo dell'espositore, le piastrelle sono un elemento <DIV>, create anch'esse con la funzione creazioneComponente() Per la generazione ci si appoggia sul file borsa.js, che, mediante la funzione prendiPiastrelle() restituisce un array con i numeri identificativi già generati randomicamente: così facendo il conteggio delle piastrelle ed il numero corretto da generare viene gestito da uno specifico file

```
/* Funzione - Estrae casualmente dalla borsa N_PIASTRELLE_PER_ESPOSITORE piastrelle e le restituisce */
function prendiPiastrelle() {
    var piastrellePrese = null;

    /*  Condizione - Verifico che il contenitore delle piastrelle NON sia vuoto
     *  - ALLORA        --> Estraggo 4 piastrelle dal contenitore
     *  - ALTRIMENTI    --> Segnalo all'utente che sono finite le tessere dentro la borsa
     */
    if (piastrelle.length != 0) {
        piastrellePrese = [];

        /* Ciclo - Estraggo casualmente dalla borsa N_PIASTRELLE_PER_ESPOSITORE piastrelle */
        for (var nPiastrelle = 0; nPiastrelle < N_PIASTRELLE_PER_ESPOSITORE; nPiastrelle++) {
            var random = Math.random() * piastrelle.length;
            piastrellePrese.push(piastrelle.splice(random, 1));
        }

    } else
        errore('Tessere esaurite', -2);

    /* Return - Restituisco le 4 piastrelle estratte casualmente dalla borsa */
    return piastrellePrese;
}
```

Piastrelle:

Come già accennato in precedenza, alle piastrelle viene aggiunto il controllo degli eventi, che ne permettono (successivamente al click dell'utente) il passaggio dall'elemento "padre" espositore, all'elemento "padre" tavolo, rendendo quindi possibile l'azione di gioco

Riassumendo: Codice Passo-Passo

Vogliamo adesso dare un’idea di come funzionino tutte le componenti descritte, a livello di script e di codice:

Urge una premessa, sarà un riassunto molto semplificato per evitare di inserire tutto il codice del progetto e per evidenziare i punti cardine del sistema messo (*quasi*) in piedi

Passo 1 → Tavolo.html \ Tavolo.js

Nonostante l'applicazione abbia come index il menù, la reale generazione del campo da gioco con i componenti avviene dal file Tavolo.html, appoggiandosi direttamente al file Tavolo.js.

Nel file HTML ci siamo limitati ad incorporare tutti i file di script e css che ci servivano per la creazione dell'ambiente di gioco

File css

File js

```
<!-- Importa i fogli di stile (i file con estensione "css") -->
<link rel="stylesheet" type="text/css" href="style/espositore.css">
<link rel="stylesheet" type="text/css" href="style/piattaforma.css">
<link rel="stylesheet" type="text/css" href="style/tavolo.css">

<!-- Importa gli script javascript (i file con estensione "js") -->
<script type="text/javascript" src="script/classe.js"></script>
<script type="text/javascript" src="script/script.js"></script>
<script type="text/javascript" src="script/borsa.js"></script>
<script type="text/javascript" src="script/espositore.js"></script>
<script type="text/javascript" src="script/piattaforma.js"></script>
<script type="text/javascript" src="script/tavolo.js"></script>
```

Passo 1 → tavolo.html \ tavolo.js

Andiamo dunque ad analizzare lo script tavolo.js:
Questo file si esegue automaticamente appena la sua controparte HTML viene aperta grazie a questa funzione:

```
window.addEventListener(EVENTO_LOAD, eventoRicercaElementiTavolo);
```

La funzione eventoRicercaElementiTavolo( ) cerca dal documento DOM tutti gli elementi corrispondenti alla classe tavolo: serve per gestire più tavoli.
Successivamente si richiama la funzione `creaElementoTavolo(tavolo, nGiocatori)`, a cui, per ogni tavolo individuato, si creano i seguenti componenti:

```
var piattaforma = creazioneComponente(tavolo, ELEMENTO_HTML, Classe.TAVOLO.PIATTAFORMA.nome,
                                      null, setPiattaforma)
```

Passo 1 → tavolo.html \ tavolo.js

```
setLayoutEspositore(espositore, nEspositore, nEspositori);
```

Infine, per quanto riguarda la gestione grafica, si richiama questa funzione per disporre ordinatamente gli espositori intorno alla piattaforma.

```
aggiungiEventiTavolo(tavolo);
```

Successivamente la funzione principale, richiama anche la aggiungiEventi che imposta il comportamento del campo da gioco generale, come lo zoom dell'elemento, la comparsa del pulsante "indietro" se si attiva lo zoom e la sua "scomparsa" dal DOM nel caso lo si prema.
Vengono aggiunte inoltre tutte le impostazioni di formattazione, come la luce soffusa dietro l'elemento zoommato.

Passo 2 → Creazione della piattaforma (GUI)

Quando il tavolo.js crea l'elemento HTML con la classe 'piattaforma' vengono richiamate alcune funzioni di piattaforma.js. Quelle che creano la GUI sono:

- ➔ `function setPiattaforma(piattaforma)`
  Passato come parametro l'elemento HTML con la classe 'piattaforma',
  questa funzione crea le varie sezioni della piattaforma;
- ➔ `function setPunteggio(punteggio)`
  Viene richiamata da setPiattaforma( ); crea la sezione 'punteggio' e la aggiunge alla piattaforma;
- ➔ `function setRighe(righe)`
  Viene richiamata da setPiattaforma( ); crea la sezione 'righe' e la aggiunge alla piattaforma;
- ➔ `function setParete(parete)`
  Viene richiamata da setPiattaforma( ); crea la sezione 'parete' e la aggiunge alla piattaforma;
- ➔ `function setBasamento(basamento)`
  Viene richiamata da setPiattaforma( ); crea la sezione 'basamento' e la aggiunge alla piattaforma;

Passo 2 → Creazione della piattaforma (evento)

Quando il tavolo.js crea l'elemento HTML con la classe 'piattaforma' vengono richiamate alcune funzioni di 'piattaforma.js'. Quelle che gestiscono gli eventi sono:

- ➔ `function aggiungiEventiPiattaforma(piattaforma)`
  Passato come parametro l'elemento HTML con la classe 'piattaforma',
  questa funzione aggiunge i vari eventi al parametro;
- ➔ `function eventoRighe(righe, parete, infoCella)`
  Viene richiamata da aggiungiEventiPiattaforma( ); aggiunge l'evento per inserire le piastrelle all'interno della sezione 'righe';

Passo 2 → Creazione della piattaforma (evento legato alla sezione ‘righe’)

- ➔ `function eventoRighe(righe, parete, infoCella)`
  - `function controlloPiastrella(infoCella, piastrella, righe, parete)`
    Controlla che l’inserimento della piastrella sia corretto
  - `function aggiungiPiastrella(righe, nRiga, piastrella)`
    Aggiunge una piastrella nella sezione ‘righe’
  - `function isRigaPiena(nRiga)`
    Controlla che la riga sia piena (nella sezione ‘righe’)
  - `function svuotaRiga(righe, nRiga)`
    Svuota la riga (nella sezione ‘righe’)
  - `function inserisciPiastrella(nRiga, piastrella, parete)`
    Inserisce una piastrella (nella sezione ‘parete’)

Passo 3 → Creazione dell’espositore

- `function eventoRicercaComponentiEspositori()` → Cerca possibili espositori all’interno del documento, successivamente crea le piastrelle;

- `function setEspositore(espositore)` → Mediante le seguenti funzioni, imposta il formato delle piastrelle ed il loro comportamento:
  - ```
    var piastrella = creazioneComponente(espositore, ELEMENTO_IMMAGINE,
    Classe.ESPOSITORE.PIASTRELLA.nome, 'piastrella-' + nPiastrella,
    setPiastrella)
    ```
  - `setPiastrella(piastrella)`
    ```

Passo 4 → Utilizzo del file classe.js

Dentro classe.js è presente una struttura logica per sapere i nomi delle classi CSS utili.
Questa struttura ‘ad albero’ può partire dal ramo:

- ➔ tavolo (aumenta il confort di utilizzo nel file tavolo.js);
- ➔ piattaforma (aumenta il confort di utilizzo nel file piattaforma.js);
- ➔ espositore (aumenta il confort di utilizzo nel file espositore.js);
- ➔ errori (per richiamare gli errori ed ignorare la struttura che lì precede).

Per farsi restituire la classe con un output di tipo TESTUALE basta richiamare la proprietà ‘nome’ (presente in tutti i ‘percorsi dell’albero’).
Es. Classe.TAVOLO.nome → restituisce la stringa ‘tavolo’ corrispondente alla classe ‘tavolo’

Passo 5 → Utilizzo del file borsa.js dentro espositore.js

Appena viene caricato il file borsa.js, viene attivato un listener che aspetta il caricamento della finestra per invocare la `funzione eventoCaricaBorsa()` che carica dentro ad un array 100 piastrelle di 5 tipologie differenti (20 piastrelle per tipo).

Per estrarre le piastrelle dalla borsa basta richiamare la `funzione prendiPiastrelle()` che restituisce un array contenente 4 piastrelle.

Passo 6 → script.js

```
function creazioneComponente(corpo, tipoElemento, classe, id, funzione = null)
```

Infine urge la descrizione della funzione forse più utilizzata del codice, quella che crea tutti gli elementi di gioco:
Questa funzione restituisce l'elemento che si desidera creare, e necessita di un:

- "corpo" → del padre, a cui verrà impostato come figlio.
- "tipoElemento" → elemento HTML (TAG) su cui verrà costruito il documento
- classe → la classe css per la formattazione dell'elemento
- id → l'identificatore che verrà assegnato all'elemento (preferibilmente univoco)
- funzione → La funzione (come setEspositore) che verrà eseguita su quel tipo di elemento
    - Si passa senza parametri, perché le funzioni di set lavorano su un elemento che è già implicito nella chiamata alla creazioneComponente( )