

Facciamo un po' di scale: chitarra e html 5

Proseguiamo l'analisi dell'ambiente di sviluppo perfezionando lo strumento musicale che abbiamo realizzato nell'ultimo numero.



La volta scorsa abbiamo parlato dell'elemento `<canvas>` di `html5`, che permette di disegnare in `JavaScript`. Come abbiamo osservato, il toolkit grafico a disposizione è di tutto rispetto e si può confrontare con quello che offrono le librerie per la grafica bidimensionale. Ricordiamo che un eccellente strumento per esplorare, imparare e sperimentare con le possibilità che offre l'elemento `<canvas>` è il corso online di `w3schools`, all'indirizzo `w3schools.com/tags/ref_canvas.asp`.

In questa puntata, riprendiamo il nostro prototipo per riflettere criticamente sul risultato, applicare sani principi di design e offrire funzionalità reali. Vogliamo utilizzare la nostra tastiera per visualizzare lo schema delle scale più comuni, in diverse posizioni del manico.

Le difficoltà di realizzazione e i concetti che andranno riflessi dal codice dipendono dal dominio del problema, che è la struttura della musica e la costruzione dello strumento, quindi facciamo una piccola premessa per chiarire i concetti in gioco.

Parliamo di queste scale

Una scala musicale è una progressione di note, quasi sempre sette, da una nota scelta come partenza alla stessa nota un'ottava sopra. Tutti abbiamo familiarità con la successione delle note in musica, `do, re, mi, fa, sol, la, si, do`. Questa è una scala, precisamente la scala di `do` maggiore. Notiamo che nella scala compaiono sette note di nome diverso, questo è un principio generale.

Le note contigue, non sono tutte realmente contigue, come si vede guardando la tastiera di un pianoforte, che mostra dei tasti neri insieme ad alcuni tasti bianchi. Fra un `do` e un `re` c'è un tasto nero, così fra un `re` e un `mi`, mentre tra un `mi` e un `fa` o un `si` e un `do` non ci sono tasti neri. C'è una dissimmetria nella scala musicale, che ha origini storiche e fondamentali musicali e dà tremendamente fastidio a chi vorrebbe vedere una teoria matematicamente più aggraziata della musica. Non si può avere tutto.

Insomma, fra un `do` e un `re` ci sono due *semitoni*, mentre tra un `mi` e un `fa` ce n'è uno solo. Questa disegualianza

degli scalini della scala obbliga a aggiustamenti. Per esempio, una scala di `re` maggiore sarebbe `re, mi, fa#, sol, la, si, do#`, con il segno `#` a indicare che certe note devono essere alterate, suonando un tasto più verso la cassa, sulla chitarra, o premendo il tasto nero o bianco immediatamente a destra del tasto normale.

La particolarità di una chitarra

Consideriamo ora i vincoli musicali che impone la costruzione di una chitarra.

La prima cosa che osserviamo sul manico è una serie di barrette, chiamate tasti, su cui si appoggia la corda premuta dal dito. Ogni tasto corrisponde a un semitono.

Le corde di una chitarra sono costruite in modo diverso, con massa e tensione calibrate in modo da ottenere suoni crescenti, quindi per fare un salto ascendente di pochi semitoni si può premere uno dei tasti nelle vicinanze sulla stessa corda, mentre per un salto più ampio, conviene cambiare corda. L'accordatura normale comporta un salto di cinque semitoni fra una corda e la successiva, con i suoni più gravi nella corda più in alto e i più acuti in quella in basso.

Anche qui c'è una dissimmetria: fra la seconda e la terza corda, contando dalla più acuta, quella in basso, ci sono quattro invece di cinque semitoni. Per complicare ancora le cose, alcuni musicisti preferiscono un'accordatura diversa da quella tradizionale, spostando di nuovo tutte le note. Ci



sono ottime ragioni musicali per usare accordature alternative, alcune di queste hanno un nome e sono popolari. Con queste stranezze sembra piuttosto controintuitivo che la chitarra sia considerata uno strumento più semplice del pianoforte, infatti è solo uno strumento più economico e trasportabile, quindi da sempre più diffuso.

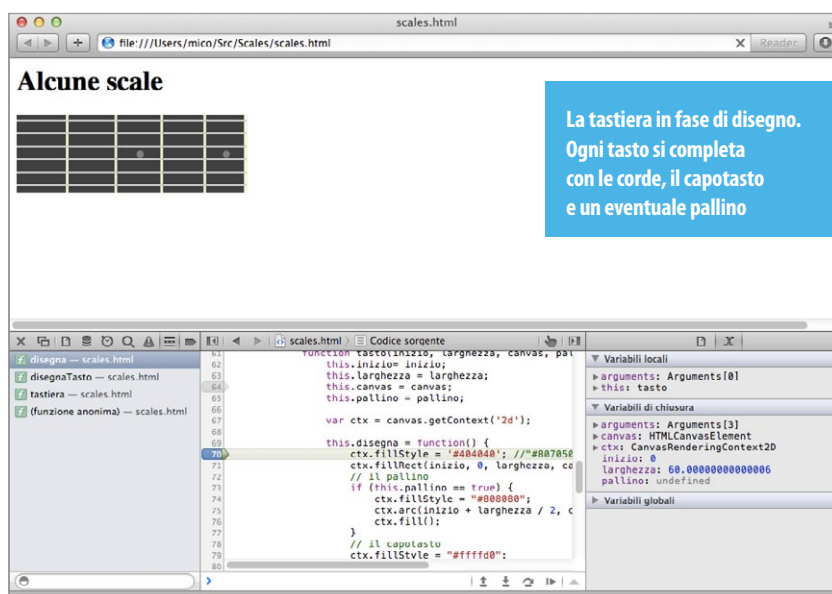
Uno sguardo al codice

Il codice che abbiamo creato nella precedente puntata era piuttosto spartano, anche se funzionale. Ecco il ciclo al centro del codice:

```
for (i = 1; i < numeroTasti; i++) {
  context.fillStyle =
    coloreTasto;
  context.fillRect(offset, 50,
    tasto[i], 100);
  if (tastiConPallino.indexOf(i)
    >= 0) {
    context.fillStyle =
      colorePallino;
    context.arc(offset +
      tasto[i] / 2, 100, 6, 0, ;
    context.fill();
  }
  offset += tasto[i] + 3;
}
```

Cominciamo con le critiche.

Ci sono una serie di variabili globali, per esempio il numero e il colore dei tasti, che starebbero meglio in un oggetto separato. C'è un test condizionale, che



gestisce determinati tasti, quelli che di solito hanno un pallino o un'altra decorazione. Questo test starebbe meglio in un metodo specifico, per portare la conoscenza specifica negli strati bassi del codice, mantenendo il più possibile semplici gli strati superiori. Se assumiamo di gestire solo browser con il supporto per le estensioni ECMAScript5, possiamo arrivare fino a qui, nella ricerca della semplicità

```
tastiera.tasti.forEach(tastiera.
  disegnaTasto);
```

Assumendo che esista un oggetto,

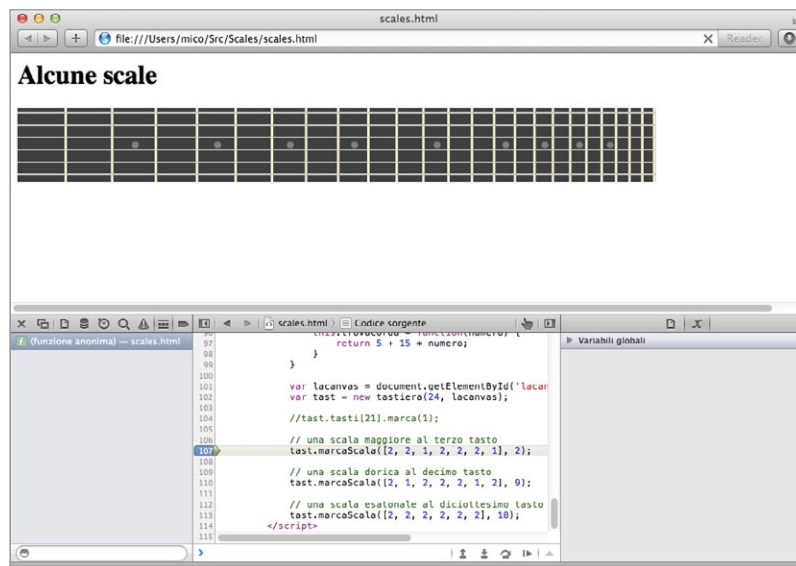
chiamato tastiera, che contenga un array di oggetti chiamato tasti e una funzione disegnaTasto. In questa funzione possiamo annegare la conoscenza di quali tasti siano da disegnare con un'evidenziazione. La funzione forEach viene invocata con un parametro, che è l'oggetto corrente nell'array. Grazie a forEach si può evitare l'uso di un ciclo con una variabile e non ci sono penalizzazioni in termini di prestazioni. Ecco un esempio molto semplice in cui un array viene stampato nella console di debug del browser:

```
var a = ["a", "b", "c"];
a.forEach(function(entry) {
  console.log(entry);
});
```

Il codice è semplice: l'argomento di forEach è una funzione anonima che passa a console.log l'oggetto corrente. L'alternativa più compatibile è più verbosa

```
var index;
var a = ["a", "b", "c"];
for (index = 0; index < a.length;
  ++index) {
  console.log(a[index]);
}
```

Per una analisi più dettagliata della funzione forEach, consigliamo la guida annotata a ECMAScript che si trova su <http://es5.github.com>, da cui apprendiamo due cose: gli elementi dell'array sono spazzolati in ordine crescente di indice e la funzione è invocata solo per gli elementi che esistono, quindi un



Dopo l'esecuzione del costruttore, la tastiera completa

array sparso, in cui solo per certi indici esiste un oggetto, non pone problemi.

Avanti con i miglioramenti

Proseguendo con la nostra incapsulazione, la tastiera deve sapere quanti sono i tasti, di che colore sono, dove si trovano e quali sono quelli speciali. Pensiamo quindi a un oggetto tastiera, contenente un array di tasti e delle variabili di configurazione. Ogni tasto deve essere in grado di disegnare le corde, la barretta del capotasto e i pallini che segnano le note di una scala sul manico. La nostra prima ingegnerizzazione riesce a ridurre il codice di alto livello a un minimo che è espressivo e snello:

```
var lacanvas = document.
  getElementById('lacanvas');
var tast = new tastiera(24,
  lacanvas);

// una scala maggiore al terzo tasto
tast.marcaScala([2, 2, 1, 2, 2, 2,
  1], 2);

// una scala dorica al decimo tasto
tast.marcaScala([2, 1, 2, 2, 2, 1,
  2], 9);
```

Il risultato è quello mostrato nella figura, migliorabile esteticamente, ma funzionalmente corretto.

L'oggetto tastiera

L'istruzione

```
var tast = new tastiera(24,
  lacanvas);
```

crea un oggetto di tipo tastiera. Come abbiamo detto altre volte, JavaScript è un linguaggio a oggetti, ma non ha la struttura sintattica delle classi. Un oggetto è semplicemente un dizionario creato da una funzione, chiamata come costruttore. La funzione che costruisce un oggetto può usare la variabile *this* per inizializzare l'oggetto in costruzione. Ecco lo scheletro del costruttore della tastiera

```
function tastiera(numeroTasti,
  canvas) { }
```

Scegliamo di passare due parametri al costruttore, il primo è il numero di tasti,

IL CODICE DI ESEMPIO

```
<html>
  <body>
    <h1>Alcune scale</h1>
    <p>
      <canvas id="lacanvas" width="1000" height="90" style="border:0px solid
        #000000;" />
    </p>
    <script language="javascript">
      function disegnaTasto(entry) {
        entry.disegna();
      }

      function tastiera(numeroTasti, canvas) {
        this.numeroTasti = numeroTasti;
        this.scala = canvas.width;
        this.canvas = canvas;

        // un'approssimazione della costante
        // della progressione temperata
        this.ratio = 0.94;
        this.tasti = new Array();

        this.marcaScala = function(scala, inizio) {
          var pos = inizio;
          var corda = 5;
          var grado = 0;

          do {
            if (grado == 0) {
              this.tasti[pos].marca(corda, '#ff3030');
            } else {
              this.tasti[pos].marca(corda);
            }
            pos += scala[grado];
            grado++;
            if (grado >= scala.length) {
              grado = 0;
            }
            if (pos > inizio + 3) {
              pos -= (corda == 2 ? 4 : 5);
              corda--;
            }
          } while(corda >= 0);
        }

        // inizializza i tasti
        var da = this.scala;
        for (i = 0; i < numeroTasti; i++) {
          var larghezza = da * (1 - this.ratio);
          this.tasti[i] = new tasto(this.scala - da, larghezza, this.
            canvas);

          if ([3, 5, 7, 9, 12, 15, 17, 19, 21].indexOf(i + 1) >= 0) {
            this.tasti[i].pallino = true;
          } else {
            this.tasti[i].pallino = false;
          }
          da -= larghezza;
        }
      }
    </script>
  </body>
</html>
```

```

        this.tasti.forEach(disegnaTasto);
    }

    function tasto(inizio, larghezza, canvas, pallino) {
        this.inizio= inizio;
        this.larghezza = larghezza;
        this.canvas = canvas;
        this.pallino = pallino;

        var ctx = canvas.getContext('2d');

        this.disegna = function() {
            ctx.fillStyle = '#404040'; //"#807050";
            ctx.fillRect(inizio, 0, larghezza, canvas.height);
            // il pallino
            if (this.pallino == true) {
                ctx.fillStyle = "#808080";
                ctx.arc(inizio + larghezza / 2, canvas.height / 2, 4, 0, 2
* Math.PI);

                ctx.fill();
            }
            // il capotasto
            ctx.fillStyle = "#ffffd0";
            ctx.fillRect(inizio + larghezza - 3, 0, 3, canvas.height);
            // le corde
            ctx.fillStyle = "#d0d0d0";
            for (corda = 0; corda < 6; corda++) {
                ctx.fillRect(inizio, this.trovaCorda(corda), larghezza, 2);
            }

            this.marca = function(corda, colore) {
                ctx.beginPath();
                colore = (typeof colore == "undefined") ? "#a0a0ff" : colore;
                ctx.fillStyle = colore;
                ctx.arc(inizio + larghezza / 2, this.trovaCorda(corda) , 7, 0,
2 * Math.PI);

                ctx.fill();
            }

            this.trovaCorda = function(numero) {
                return 5 + 15 * numero;
            }
        }

        var lacanvas = document.getElementById('lacanvas');
        var tast = new tastiera(24, lacanvas);

        //tast.tasti[21].marca(1);

        // una scala maggiore al terzo tasto
        tast.marcaScala([2, 2, 1, 2, 2, 2, 1], 2);

        // una scala dorica al decimo tasto
        tast.marcaScala([2, 1, 2, 2, 2, 1, 2], 9);

        // una scala esatonale al diciottesimo tasto
        tast.marcaScala([2, 2, 2, 2, 2, 2], 18);
    }
</script>
</body>
</html>

```

in modo da poter creare tastiere con 21, 22 o 24 tasti, quelle comuni nella liuteria contemporanea. L'altro parametro che passiamo è la canvas su cui disegnare, un passaggio obbligato per due ragioni, la prima è che per disegnare occorre un riferimento al contesto grafico della canvas, e noi vogliamo che la tastiera si disegni da sé. La seconda ragione è che passando la canvas non abbiamo bisogno di indicare la larghezza e l'altezza dell'area su cui disegnare, quindi eliminiamo parametri in sovrappiù. Copiamo i parametri in variabili locali

```

        this.numeroTasti = numeroTasti;
        this.scala = canvas.width;
        this.canvas = canvas;

```

Consideriamo ora quel numero magico di cui abbiamo parlato la volta scorsa: la progressione delle note nella scala temperata è geometrica di ragione radice dodicesima di dodici. La lunghezza di ogni tasto si ottiene moltiplicando la lunghezza del precedente per la ragione della progressione geometrica, che arrotondiamo a 0,94.

```

// un'approssimazione della costante
della progressione temperata
this.ratio = 0.94;

```

Ora pensiamo a costruire un array di oggetti tasto inizializzando i parametri che lo definiscono, il punto di inizio e la larghezza. Passiamo il puntatore alla canvas anche al tasto, che se ne serve per disegnarsi.

```

this.tasti = new Array();
// inizializza i tasti
var da = this.scala;
for (i = 0; i < numeroTasti; i++) {
    var larghezza=da * (1-this.ratio);
    this.tasti[i] = new tasto(this.
        scala - da, larghezza,
        this.canvas);
    if ([3, 5, 7, 9, 12, 15, 17,
19, 21].indexOf(i))
    { this.tasti[i].pallino=true;
    } else {
        this.tasti[i].pallino =
        false;
    }
    da -= larghezza;
}

```

Abbiamo riadattato il codice iniziale con diverse modifiche. Vogliamo conservare l'origine di ogni tasto e la calcoliamo nella prima riga all'esterno del ciclo,

memorizzandola nella variabile *da*, che passiamo al costruttore. Allo stesso modo calcoliamo la *larghezza* del tasto, uguale a *da* meno *da* moltiplicato per la ragione della progressione. La seconda riga all'interno del ciclo crea un oggetto tasto e lo conserva nell'array *tasti*. Infine, impostiamo la variabile booleana *pallino* per indicare se un tasto ha o no il marcatore, che di solito è un pallino. Aggiorniamo la scala della tastiera sottraendo la lunghezza del tasto prima di ripetere il ciclo. Nel costruttore disegniamo la tastiera in modo da lasciare la canvas pronta per successive decorazioni.

```
this.tasti.forEach(disegnaTasto);
```

La funzione *disegnaTasto* è invocata, come detto, per ogni elemento dell'array in ordine crescente e rimanda a una funzione *disegna*, definita per ogni tasto

```
function disegnaTasto(entry) {
  entry.disegna();
}
```

Passiamo ora al costruttore del tasto, copiatura delle variabili di costruzione e inizializzazioni

```
function tasto(inizio, larghezza,
  canvas, pallino) {
  this.inizio = inizio;
  this.larghezza = larghezza;
  this.canvas = canvas;
  this.pallino = pallino;
  var ctx = canvas.
    getContext('2d');
}
```

vediamo come strutturare il metodo



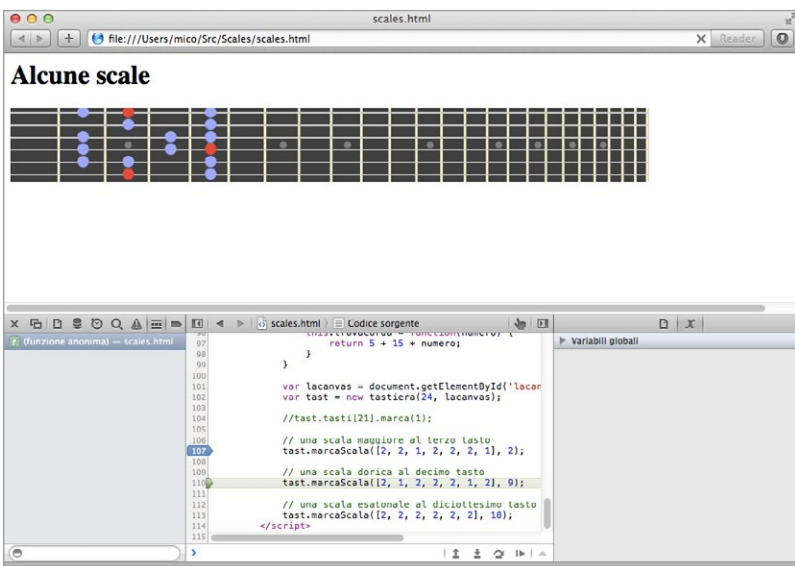
PER SAPERNE DI PIÙ

→ L'articolo di Wikipedia sulle scale ha bisogno di un po' di lavoro, ma punta a una lunga selezione di scale, incluse le meno comuni it.wikipedia.org/wiki/Scala_musicale

→ Una trattazione più approfondita e scolasticamente adeguata si trova all'indirizzo www.oradimusic.it/sitopub/seconda/scala/scala1.pdf

→ La guida annotata a JavaScript nella standardizzazione ECMAScript5 es5.github.com

→ Un'analisi in profondità dell'ereditarietà in JavaScript e un modello per simulare un'ereditarietà in stile Java www.crockford.com/javascript/inheritance.html



Disegniamo la prima delle scale che esercitano il nostro calcolatore musicale, una scala maggiore

disegna dell'oggetto *tasto*

```
this.disegna = function() {}
```

Per prima cosa disegniamo il tasto, in un colore scuro come il nero ebano impostato, o il marrone nel commento

```
ctx.fillStyle = '#404040';
//"#807050";
ctx.fillRect(inizio, 0, larghezza,
  canvas.height);
```

Si noti che usiamo liberamente *ctx*, che è una variabile definita nella chiusura di ogni oggetto *tasto*. Ricordiamo che i primi due parametri di *fillRect* sono il punto di origine del rettangolo, con lo zero in alto a sinistra nella canvas. Gli ultimi due parametri sono larghezza e altezza del rettangolo. Il colore di riempimento è definito nel *fillStyle* del contesto grafico. Disegniamo, adesso, il pallino, se il tasto è stato marcato come uno dei tasti da segnare

```
// il pallino
if (this.pallino == true) {
  ctx.fillStyle = "#808080";
  ctx.arc(inizio + larghezza / 2,
    canvas.height/2, 4,0,2 * Math.PI);
  ctx.fill();
}
```

Ricordiamo che i primi due parametri di *arc* sono il punto di origine dell'arco, segue il raggio (4 in questo caso) e infine inizio e fine dell'arco. Da zero a due pi greco, imposta un cerchio completo. Il colore impostato nel contesto è un grigio medio, 80 infatti è la metà della capacità di un byte non segnato in esadecimale. Valori uguali per rosso, verde e blu danno un grigio, più o meno scuro. Questi codici numerici si potrebbero spostare in variabili di configurazione

o rendere più espliciti, ma questo è un miglioramento successivo. Disegniamo ora il capotasto

```
// il capotasto
ctx.fillStyle = "#fffd0";
ctx.fillRect(inizio + larghezza -
  3, 0, 3, canvas.height);
```

Rosso e verde al massimo, blu un po' indietro per un giallo brillante. Il capotasto è disegnato come una barretta rettangolare larga 3 pixel, che inizia tre pixel prima dell'inizio del nostro tasto più la larghezza dello stesso, cioè si accosta al margine destro. Il tasto si estende per tutta l'altezza dell'area di disegno, così come il tasto.

Infine disegniamo le corde, sempre dei rettangoli alti 2 pixel, con la stessa origine e la stessa larghezza del tasto, ma posizionati ad altezze diverse.

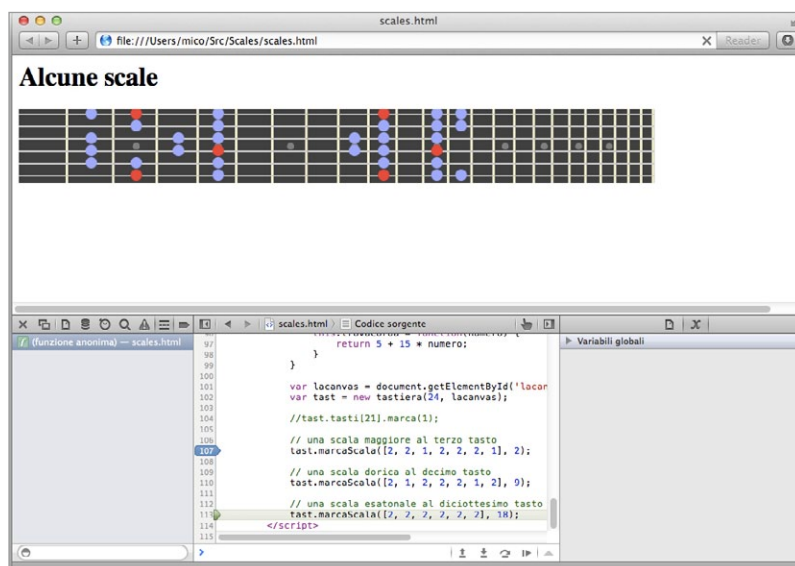
Una funzione di utilità

Usiamo una funzione *trovaCorda* per calcolare l'ascissa di una corda, dato il numero.

```
// le corde
ctx.fillStyle = "#d0d0d0";
for (corda = 0; corda < 6; corda++)
{ ctx.fillRect(inizio, this.
  trovaCorda(corda), larghezza, 2);
}
```

La funzione *trovaCorda* calcola il punto di inizio di ogni corda sommando un margine di 5 pixel e 15 pixel per ogni corda. Questo vale perché la nostra canvas è alta novanta pixel.

```
this.trovaCorda = function(numero)
{
  return 5 + 15 * numero;
}
```



Ed ecco la seconda, una scala minore dorica in posizione più alta nel manico.

In un raffinamento successivo, 5 dovrebbe essere metà del margine che si vuole lasciare ai lati delle corde, mentre quindici è l'altezza della canvas, meno il margine, diviso per i cinque spazi fra le sei corde, o un calcolo più complesso se vogliamo disegnare tastiere con un numero variabile di corde.

La funzione che calcola il punto medio di una corda è utile anche per dare a un tasto la possibilità di disegnare un marcatore, un pallino colorato, sopra le corde. Useremo questi pallini per disegnare le scale.

Ecco il codice della funzione che marca un punto su una corda in uno specifico tasto

```
this.marca = function(corda,
colore) {
  ctx.beginPath();
  colore = (typeof colore ===
"undefined") ? "#a0a0ff" : colore;
  ctx.fillStyle = colore;
  ctx.arc(inizio + larghezza / 2,
this.trovaCorda(corda) , 7, 0, 2 *
Math.PI);
  ctx.fill();
}
```

Osserviamo l'uso di `trovaCorda` nella `arc` che disegna il pallino.

Il `fillStyle` del contesto grafico è impostato con il secondo parametro della funzione, ma se questo mancasse, impostiamo un blu brillante con l'assegnazione condizionale della seconda riga del corpo della funzione. Quindi, se

```
typeof colore === "undefined"
```

il colore di default sarà `a0a0ff`. Questo meccanismo per creare parametri opzionali è quello generalmente accettato dalla comunità JavaScript. Il linguaggio

di per sé non ha un costrutto sintattico per inizializzare automaticamente parametri mancanti.

L'ultimo tocco

Passiamo finalmente al disegno di una scala, creando un metodo `marcaScala` che metteremo ovviamente nella tastiera e che userà il metodo `marca` dei diversi tasti che la compongono.

Usiamo un'approssimazione cruda, per iniziare. Definiamo il metodo in modo che disegni una scala con due parametri: un array di intervalli fra gli elementi della scala, espressi in numero di semitoni, e la posizione in cui iniziare a disegnare le posizioni partendo dalla sesta corda, quella più in basso.

```
this.marcaScala = function(scala,
inizio) { var pos = inizio;
          var corda = 5;
          var grado = 0; }
```

Usiamo un'approssimazione cruda, per iniziare. Definiamo il metodo in modo che disegni una scala con due parametri: un array di intervalli fra gli elementi della scala, espressi in numero di semitoni, e la posizione in cui iniziare a disegnare le posizioni partendo dalla sesta corda, quella più in basso.

Cominciamo disegnando il pallino nel punto indicato, selezionando il colore di default, o un rosso nel caso del primo grado della scala.

```
if (grado == 0) {
  this.tasti[pos].marca(corda,
  '#ff3030');
} else {
  this.tasti[pos].marca(corda);
}
```

proseguiamo incrementando la

posizione di scrittura del numero di semitoni specificato nella scala

```
pos += scala[grado];
grado++;
```

Adesso dobbiamo forzare due circolarità, la prima sui gradi della scala:

```
if (grado >= scala.length) {
  grado = 0;
}
```

Questo ci permette, in parole povere, dopo un sì di tornare a un do.

La seconda circolarità sta nell'esplorazione delle corde. Usiamo un metodo semplice, ma molto efficace: se ci spostiamo più di tre tasti dalla posizione iniziale, troviamo la nota sulla corda successiva, sottraendo cinque dalla posizione per tenere conto dell'accordatura più alta. Passando dalla terza alla seconda corda, sottraiamo solo quattro per tenere conto della dissimmetria dell'accordatura normale.

```
if (pos > inizio + 3) {
  pos -= (corda == 2 ? 4 : 5);
  corda--; }
```

L'approccio che seguiamo è piuttosto brutale, perché non possiamo tenere conto di accordature non convenzionali, ma anche questo è un miglioramento che possiamo tralasciare per ora.

Si noti il valore condizionale di cui si decreta la posizione al passaggio di corda, che vale 4 se stiamo lasciando la terza corda, cinque negli altri casi.

Il codice che esplora le corde è racchiuso in un ciclo `do .. while`, che termina quando abbiamo completato la corda 0, la prima, quella più in alto nel disegno.

```
do { while(corda >= 0);
```

Mettiamo alla prova il codice, con il disegno di tre scale diverse, due di sette note e una scala artificiale di sei note.

```
// Scala maggiore al terzo tasto
tast.marcaScala([2, 2, 1, 2, 2, 2,
1], 2);
// Scala dorica al decimo tasto
tast.marcaScala([2, 1, 2, 2, 2, 1,
2], 9);
// Scala esatonale al 18esimo tasto
tast.marcaScala([2, 2, 2, 2, 2, 2,
18]);
```

Il risultato si può vedere nella figura.