

# Tuto Javascript V1

## Y. Stroppa

### 2020



# Sommaire

- Introduction
- Présentation du KIT de démarrage
- Présentation de l'application
- Conception des étapes
- Résultats
- Conclusion

# Introduction

- L'objectif de ce tuto est de vous montrer étape par étape la conception d'une application de type Web en partant de l'idée et pour arriver à un résultat opérationnel.
- On démarre avec un contexte déjà préparé qui est composé d'une application sous Node.js.
- L'organisation des fichiers est la suivante :

—

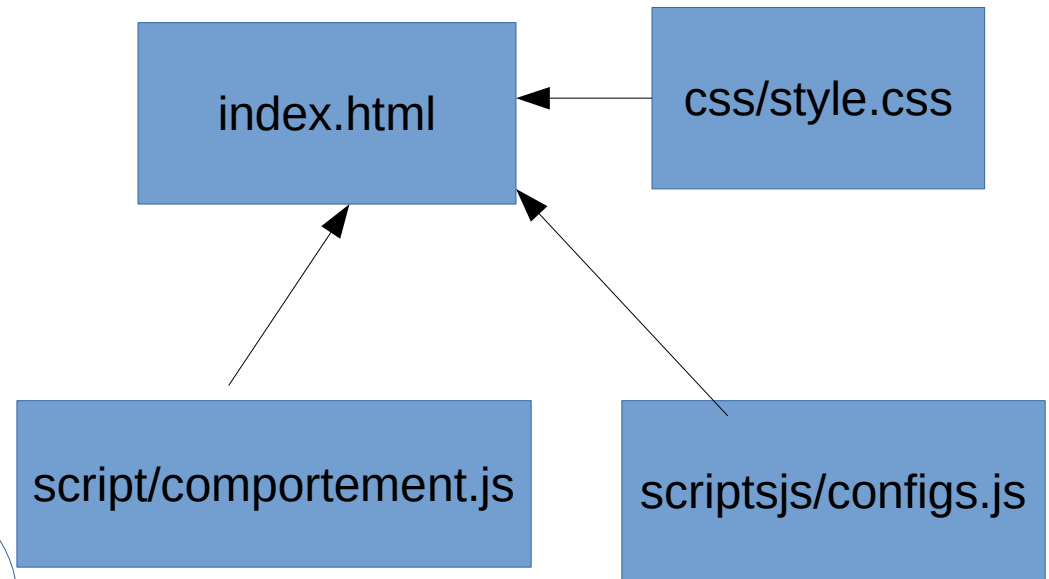
# Présentation du KIT

- Le Kit est composé d'un petit exemple déposé sur github à l'adresse :  
[https://github.com/ystroppa/PSB\\_M1\\_projet](https://github.com/ystroppa/PSB_M1_projet)
- Vous pouvez le télécharger à l'aide de la commande git clone [https://github.com/ystroppa/PSB\\_M1\\_projet](https://github.com/ystroppa/PSB_M1_projet)
- Ensuite une fois téléchargé dans le dossier nommé PSB\_M1\_projet, il vous suffit de vous positionner dans ce répertoire et d'exécuter les commandes :
  - `npm install`                      Installe à partir de npm les modules nécessaires à node.js
  - `npm start`                        Démarre node.js sur le port 8124

# Structure de notre application

- Arbosrecence

```
PSB_M1_projet/  
├── app.js  
├── bin  
│   └── www  
├── package.json  
├── public  
│   ├── css  
│   │   └── style.css  
│   ├── index.html  
│   └── scripts  
│       ├── comportement.js  
│       ├── config.js  
│       └── jquery-3.4.1.min.js  
├── routes  
│   └── index.js  
└── views  
    ├── error.jade  
    ├── index.jade  
    └── layout.jade
```



# Structure de notre application

- Du côté frontal :
  - index.html ==> présentation des éléments à l'utilisateur
  - css/styles.css ==> définition des styles utilisés par les différents éléments de notre DOM
  - js/configs.js ==> contient les déclarations des configuration de jeu
  - js/comportement.js ==> contient la déclaration des variables et des comportements de notre application. Coeur de l'application.

# Constitution de la partie CSS

- Description des styles utilisés par la page html.

# Constitution de la partie HTML

- On va tout de suite modifier le contenu de ce fichier pour le simplifier et ne garder que deux boutons RAZ, Verif et la liste déroulante :
- ```
// définition des métadonnées et des inclusions de styles
<div id="commande_top" class="commandes">
    <input type="button" id="bt3" value="RàZ" onClick="choixRaz();"/>
    <input type="button" id="bt2" value="Verif" onClick="verif();"/>
    <select name="config" id="config-select" onChange="chargement();">
    </select>
</div>
// inclusion des JavaScripts de notre appli
```



# Déroulement du chargement

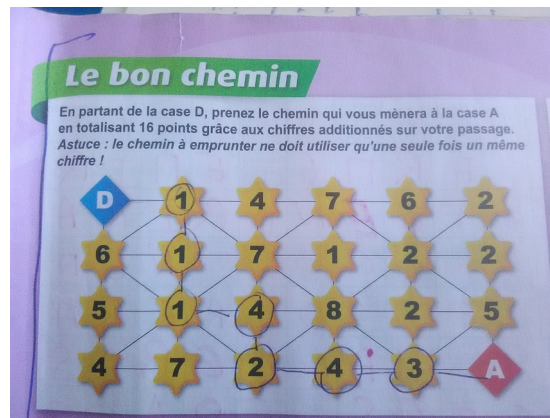
- Lorsque le navigateur charge le document html, regarde les inclusions définies dans l'entête (head) et les charge dans son contexte, ensuite charge les éléments définis dans le corps (body) et charge les inclusions js indiqués à la fin du fichier html.

# Constitution de la partie HTML

- Chargement des inclusions :
  - Charge le fichier de configuration
  - Charge le fichier comportement
- Description du fichier comportement.js
  - Le programme devra se dérouler de la manière suivante :
    - Charger la configuration à partir du fichier de configs.js
    - Afficher la première configuration
    - et initialiser l'affichage
  - Ensuite attendre les événements utilisateur

# L'application

- L'idée est de partir d'une représentation issue d'un journal de jeux et d'en faire une application en ligne utilisable à partir de n'importe quel média .... et en utilisant les techniques de PWA .



# Description du fonctionnement de l'applcation

- Les règles sont très simples et consistent à définir un chemin qui relie le point de départ au point d'arrivée en pouvant se déplacer dans tous les sens. La somme des étoiles qui constituent le chemin doit être égale à la somme indiquée.
- Comment est constitué le jeu :
  - D'une grille de  $n$  lignes et de  $m$  colonnes.
  - De deux cases spécifiques qui indiquent le point de départ et le point d'arrivée.
  - Les autres cases portent une valeur ou éventuellement un opérateur.

# Organisation

- On doit charger les configurations des jeux en mémoire
- Charger la première
- Afficher le jeu
- Attendre que l'utilisation interagisse

# Première étape

- Comment définir une configuration de jeu
  - Pour cela nous allons utiliser un fichier nommé configs.js qui permettra définir nos configurations. Pour cela nous allons définir la variables configGame comme suit :
- `var configGame=[]` ; Un tableau de structure
- La configuration d'un jeu aura :
  - un nom
  - position de la grille à l'écran : `deb_x` et `deb_y`
  - un nombre de cases
  - un niveau de difficulté de 1 à 4
  - un tableau contenant les éléments de la grille
  - une indication : qui servira à rappeler les conditions et règles du jeu
  - la ou les solutions au jeu.
  - la somme à obtenir

# Première étape : config.js

- Exemple de configuration selon le schéma défini :

```
var configGame=["classique":{  
  "nb":24,"debx":60,"deby":60,  
  "difficulte":1,"somme":16,  
  "indications":"Sélectionner ..... de D et A.",  
  "game":[["D",1,4,7,6,2],[6,1,7,1,2,2],  
           [5,1,7,8,2,3],[4,7,2,4,3,"A"] ],  
  "solution":[["D",0,0,0,0,0],[6,0,0,0,0,0],  
               [0,1,0,0,0,0],[0,0,2,4,3,"A"] ],  
}];
```

# Deuxième étape :

- Lecture et chargement des configurations
  - On peut se reporter dans le fichier comportement.js et définir deux nouvelles fonctions : une qui servira à charger toutes les configurations dans le select de notre html (charge\_listeJeux()) ce qui permettra à l'utilisateur de sélectionner celle de son choix et une fonction chargement() qui chargera la configuration sélectionnée.
  - On exécutera au démarrage les deux fonctions dans l'ordre suivant :
    - charge\_listejeux() ;
    - chargement();



## Deuxième étape :

- `charge_listejeux()` ; Lecture de la variable `configGame` du fichier `configs.js` et chargement du select "config-select" de la page html à partir de la clé de la variable.

```
function charge_listejeux() {  
    var sele = document.getElementById('config-select');  
    for (var jeu in configGame){  
        var option = document.createElement("option");  
        option.text = jeu;  
        option.value = jeu;  
        sele.add(option);  
    }  
}
```

# Deuxième étape :

- `chargement()` ; Permettra à partir de la valeur sélectionnée par l'utilisateur de la configuration de jeu de charger une variable globale que l'on va nommer `configuration`.

```
var configuration ;
```

```
function chargement_() {  
    var sele = document.getElementById('config-select');  
    var config=sele.value;  
    configuration=configGame[config];  
    trace() ;  
    $("#descriptif > p").html(configuration.indications);  
}
```

# La fonction trace()

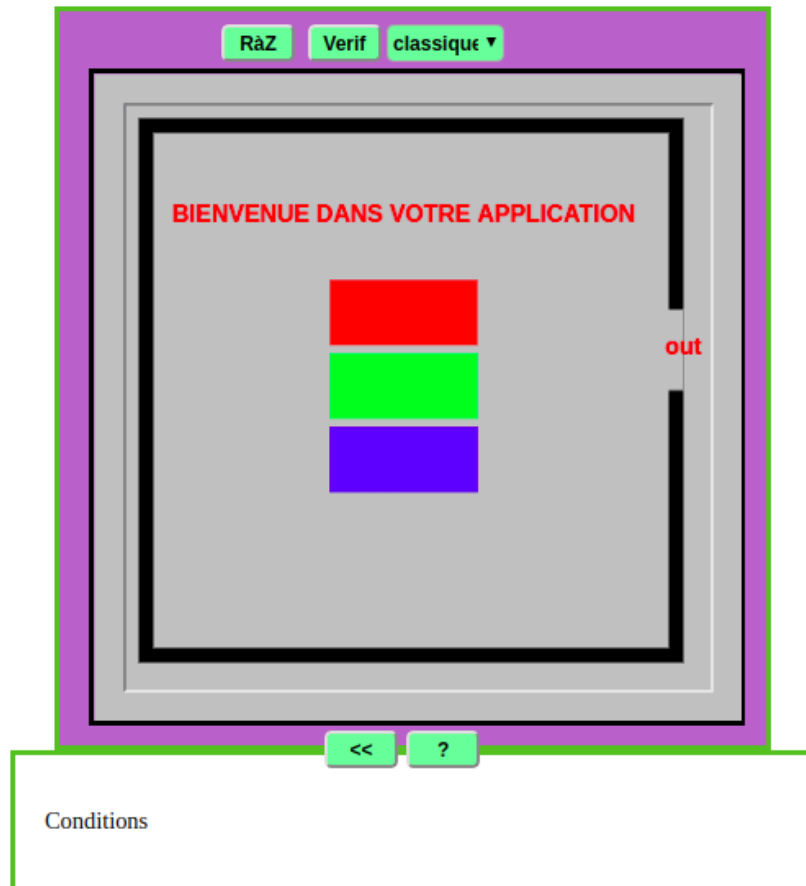
- La fonction trace() permet de dessiner le contexte d'affichage du jeu. Avec ses différents cadres et à la fin appelle affiche() qui va s'occuper à dessiner l'intérieur de 3 rectangles et d'un message dans la zone d'affichage.
- Si on veut maintenant déplacer ces éléments comment procéder ????

# Le déplacement des objets

- Dans cet exemple on utilise la notion de canvas qui permet de dessiner et de tracer à l'aide d'instructions HTML des représentations graphiques ou de charger directement des images. Mais est-ce que le fait de tracer à l'aide de `fillRect`, `drawLine` ... génère des éléments dans la DOM que l'on pourrait extraire pour changer les propriétés.
- Pour vérifier il suffit de passer en mode console au niveau du navigateur et de regarder dans l'onglet éléments si il existe des éléments spécifiques pour les rectangles ????


# Résultat

## Squelette du jeu 2020



```
Elements Console Sources Network >>
<!doctype html>
<html>
  <head>...</head>
  <body>
    <center>...</center>
    <div id="conteneur" class="center">
      <div id="contenu">
        <div id="content">
          <div id="commande_top" class="commandes">...</div>
          <div id="canv">
            <canvas id="canvas" width="440px" height="440px" style="border:
            3px solid rgb(0, 0, 0);" == $0
          </div>
          <div id="commande_bottom" class="commandes">...</div>
        </div>
      </div>
    </div>
    <script src="./scripts/config.js"></script>
    <script src="./scripts/comportement.js"></script>
    <div id="descriptif" class="center_texte">...</div>
    <div id="piedpage">
    </div>
  </body>
</html>
```

# Solution

- Comme on ne peut pas récupérer de descripteur sur les zones graphiques il nous reste la solution de tout redessiner à chaque fois. Exemple si on veut déplacer le rectangle bleu de 10 positions vers la droite.
- Il suffit d'implémenter une boucle on verra le tracé cumulatif des 10 rectangles qui se superposent en partie les uns à côté des autres.... 
- Maintenant si on veut donner l'impression que le rectangle se déplace, il faut à l'aide de la fonction `setInterval` appeler toutes les secondes la fonction `trace` et décaler la position du tracé du rectangle bleu.

# Déplacement avec setInterval

```
var positX=1;
```

En début de fichier dans les déclarations

```
function affiche(){
```

```
.....
```

```
  hh.fillRect(160+positX*10,240,100,44);
```

```
  positX+=1;
```

```
  if (positX>10) {
```

```
    clearInterval(myTimer);
```

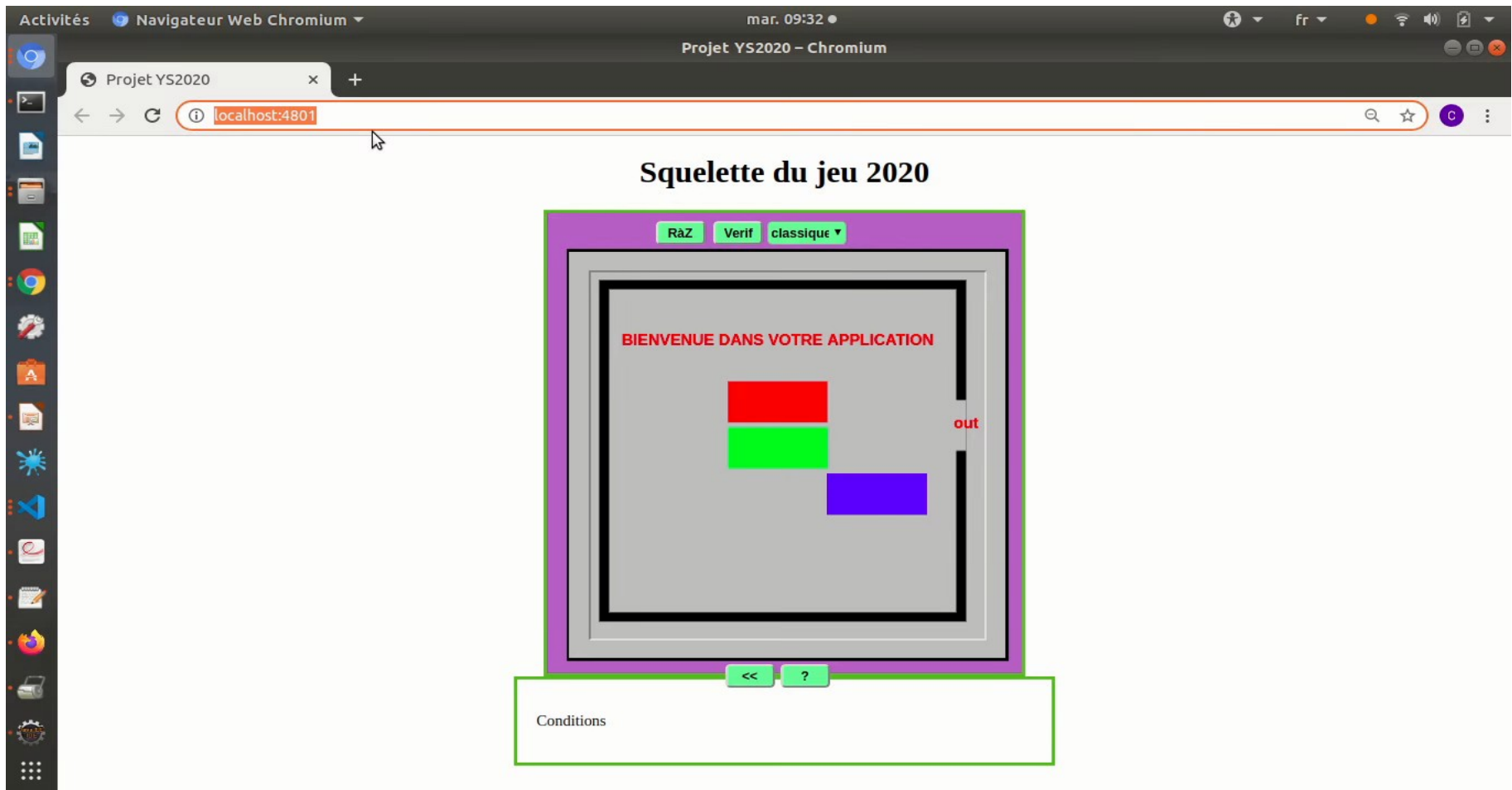
```
  }
```

```
}
```

On paramètre la position et  
on arrête le timer au bout de  
10 itérations

```
var myTimer = setInterval(trace, 1000);
```

On déclenche le timer





# Etape 2

- On revient à notre problème initial, il faut dessiner les éléments de notre jeu dans le cadre et gérer les interactions des utilisateurs et effectuer les changements visuels de nos objets graphiques. Le problème est que l'on n'a pas la possibilité d'extraire des descripteurs pour permettre en JavaScript de changer les propriétés des objets et de détecter sur quel objet vient de cliquer l'utilisateur. Sachant que ce que l'on connaît c'est la position exacte (x,y) sur laquelle vient de cliquer avec sa souris l'utilisateur.
  - Donc comment retrouver l'élément qui est à cette position ?

# Gestion de l'événement click

- Pour gérer l'événementiel nous allons définir un module d'écoute sur l'objet fenêtre ( la zone complète du Canvas).

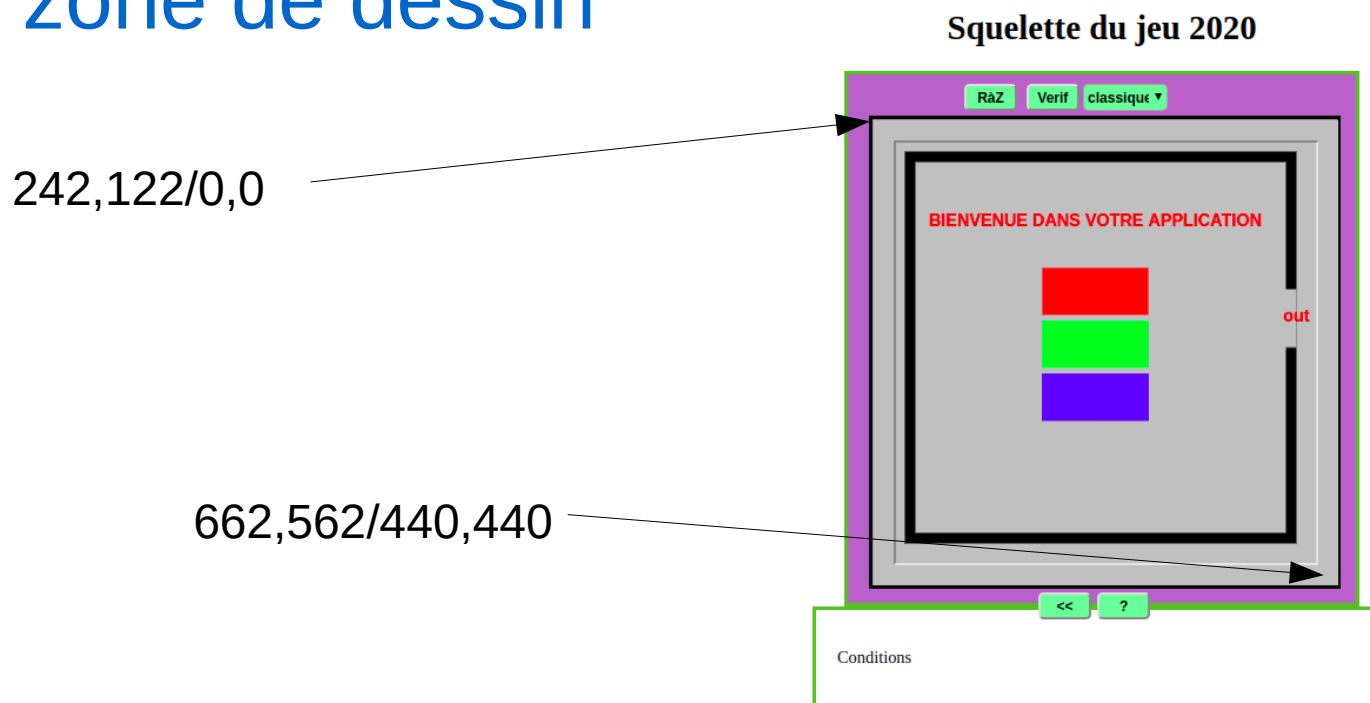
```
fenetre.addEventListener('click',function(evt) {  
    var x = evt.pageX;  
    var y = evt.pageY;  
    console.log(x,y);  
    var node = evt.target;  
    while (node) {  
        x -= node.offsetLeft - node.scrollLeft;  
        y -= node.offsetTop - node.scrollTop;  
        node = node.offsetParent;  
        console.log(x,y);  
    }  
});
```

← Position dans la page

← Position dans la zone de dessin

# Tests de la fonction

- Testez le fonctionnement de cette fonction :
- et en déduire les coordonnées intérieures de la zone de dessin



# Préparation du tracé

- Maintenant il nous faut construire notre grille d'étoiles dans le cadre d'affichage de notre application web. Pour ce faire, on va définir une fonction de tracé des étoiles. Pour cela on va rechercher sur Internet si il existe déjà ce type de tracé :
  - <https://www.tutorialspoint.com/How-to-draw-a-star-by-using-canvas-HTML5>

# Intégration de la fonction

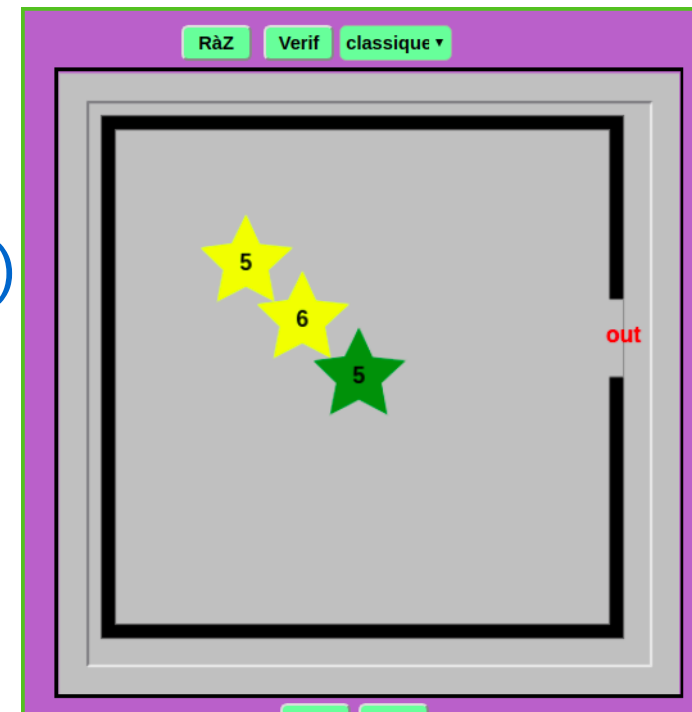
- On va recopier les instructions dans une fonction que l'on va nommer `dessine-star` et qui va recevoir les paramètres `x,y,alpha,contenu` et `couleur`. On prépare les paramètres pour adapter la position de l'étoile dans le cadre (`x,y`) afficher son contenu (`contenu`) et définir la couleur d'affichage.

```
function dessine_star(x,y,alpha,contenu=5,couleur="yellow") {  
    hh.fillStyle = couleur;  
    hh.beginPath();  
    hh.moveTo(x+108*alpha, y);  
    hh.lineTo(x+141*alpha, y+70*alpha);  
    hh.lineTo(x+218*alpha, y+78.3*alpha);  
    hh.lineTo(x+162*alpha, y+131*alpha);  
    hh.lineTo(x+175*alpha, y+205*alpha);  
    hh.lineTo(x+108*alpha, y+170*alpha);  
    hh.lineTo(x+41.2*alpha, y+205*alpha);  
    hh.lineTo(x+55*alpha, y+131*alpha);  
    hh.lineTo(x+1*alpha, y+78*alpha);  
    hh.lineTo(x+75*alpha, y+68*alpha);  
    hh.lineTo(x+108*alpha, y);  
    hh.closePath();  
    hh.fill();  
    hh.fillStyle="black";  
    hh.fillText(contenu,x+108*alpha,y+131*alpha);  
}
```

# Tests de l'affichage

- Pour tester la fonction que l'on vient de créer :
    - modifiez affichage et rajoutez les instructions suivantes :
- `dessine_star(100,100,0.3) ;`
  - `dessine_star(140,140,0.3,5) ;`
  - `dessine_star(140,140,0.3,5,"green")`

```
function affiche(){  
  // exemple d'affichage d'un texte dans la zone canvas  
  dessine_star(100,100,0.3) ;  
  dessine_star(140,140,0.3,6) ;  
  dessine_star(180,180,0.3,5,"green") ;  
}
```



On pourra supprimer tous les éléments spécifiques au Timer et à l'affichage des rectangles ....

# Tracé du contexte de jeu

- Pour tracer le cadre de notre jeu, il suffit de faire deux fonctions. Une fonction `charge_grille()` qui va nous permettre à partir de la configuration de charger un tableau de nos étoiles et une fonction `trace_grille()` qui va balayer le tableau et déclencher les affichages des étoiles dans le cadre.
- On va tout d'abord pour des raisons de simplicité créer une classe `Cellule` qui va regrouper les attributs et méthodes permettant l'exploitation de ces objets.

# Classe Cellule

```
class Cellule {  
    constructor(x,y,contenu) {  
        this.x=x;  
        this.y=y;  
        this.contenu=contenu;  
        if (contenu=="A") this.couleurOrig="red";  
        else if (contenu=="D") this.couleurOrig="green";  
        else this.couleurOrig="yellow";  
        this.statut=false;  
        this.couleur=this.couleurOrig;  
    }  
    trace(){  
        dessine_star(this.x,this.y,0.2,this.contenu, this.couleur);  
    }  
}
```



# Les fonctions

```
var liste_cellules=[] ;
```

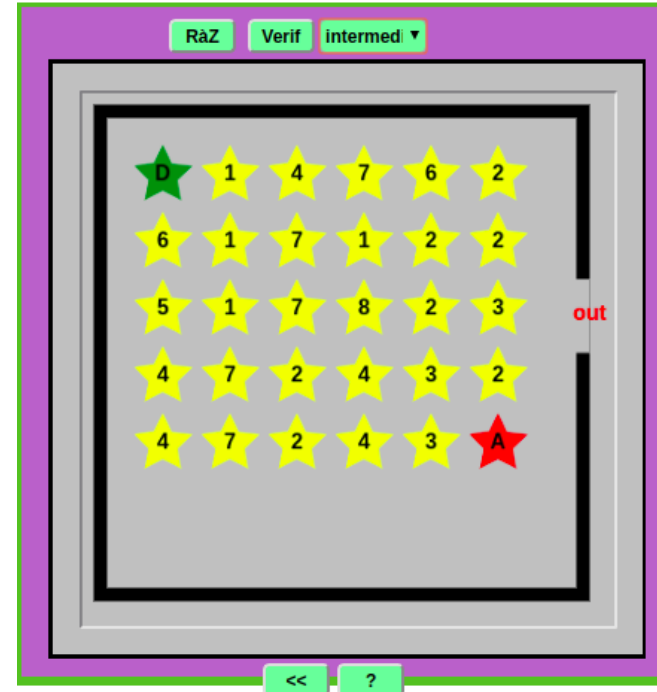
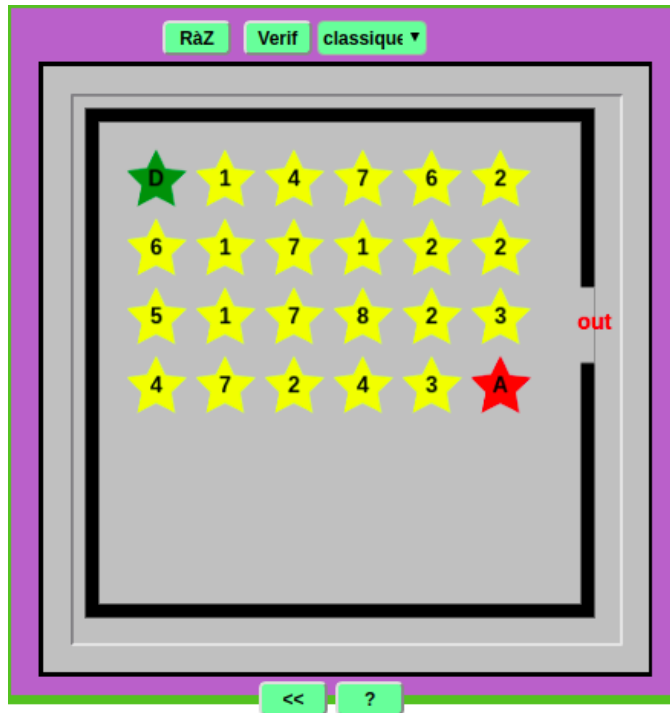
```
function charge_grille() {  
  // on balaye la grille et on affiche les differents elements  
  var position_x=configuration["deb_x"];  
  var position_y=configuration["deb_y"];  
  configuration.game.forEach(e=> {  
    position_x=configuration["deb_x"];  
    e.forEach(ee=> {  
      var cell= new Cellule(position_x, position_y,ee);  
      liste_cellules.push(cell);  
      position_x+=50;  
    });  
    position_y+=50;  
  });  
}
```

```
function trace_grille() {  
  liste_cellules.forEach(e=> {  
    e.trace();  
  });  
}
```

Modifiez chargement()  
Au dessus de trace() rajoutez l'appel  
à charge\_grille().

Modifiez trace()  
et rajoutez à la fin trace\_grille()  
On pourra enlever l'appel à  
affichage()

# Résultats



# Amélioration du rendu

- On peut s'apercevoir que on n'a pas différencié les éléments D et A des autres .
- Donc pour les différencier, on va définir une nouvelle fonction que l'on va nommer dessine-losange avec les mêmes paramètres que la fonction dessine-etoile.
- Voici le contenu de cette fonction qui permet de dessiner l'objet souhaité.

# function dessine\_loange()

```
function dessine_loange(x,y,alpha,contenu=5,couleur="green") {  
    hh.fillStyle = couleur;  
    hh.beginPath();  
    hh.moveTo(x+108*alpha, y);  
    hh.lineTo(x+218*alpha, y+100*alpha);  
    hh.lineTo(x+108*alpha, y+205*alpha);  
    hh.lineTo(x, y+100*alpha);  
    hh.lineTo(x+108*alpha, y);  
    hh.closePath();  
    hh.fill();  
    hh.fillStyle="black";  
    hh.fillText(contenu,x+108*alpha,y+131*alpha);  
}
```

Une fois la fonction chargée dans le contexte navigateur vous pouvez la tester immédiatement à l'aide de la console

```
dessine_loange(100,100,0.3) ;
```

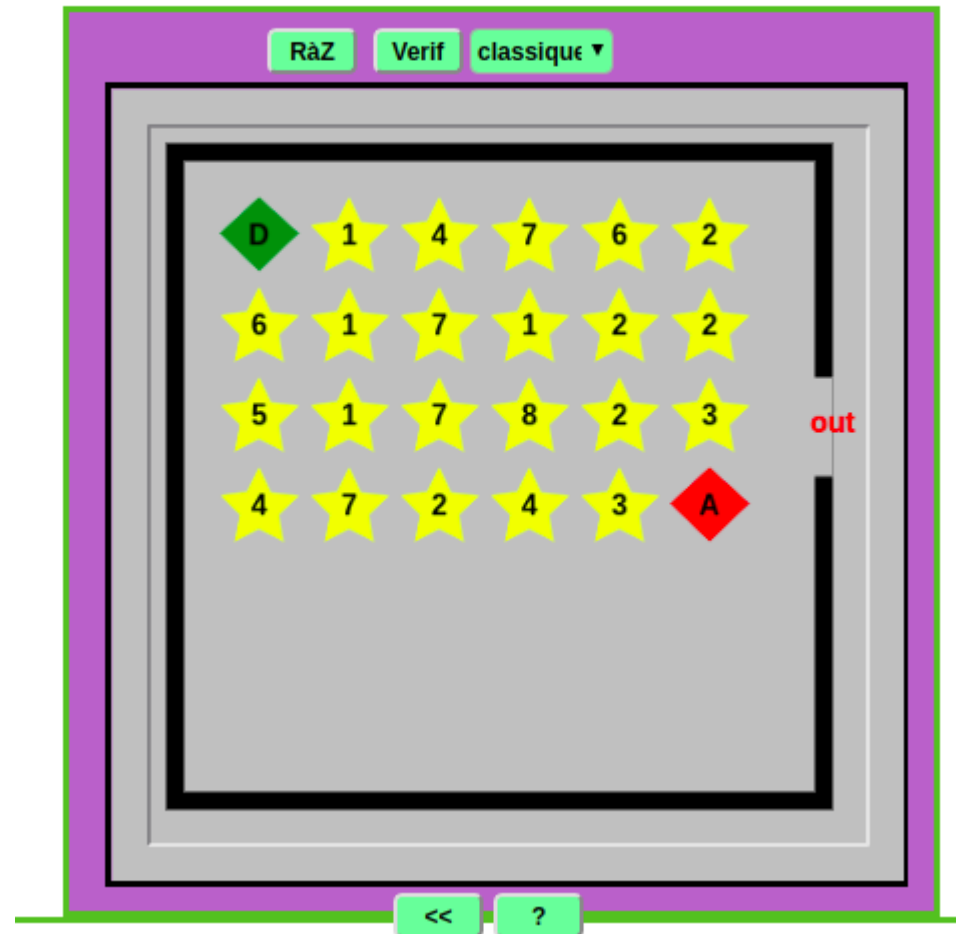
# Intégration de la fonction

- Pour intégrer cette fonction dans le reste du programme il vous suffit de modifier la méthode trace de la classe cellule pour prendre en compte en fonction du contenu le tracé adapté.

```
trace(){  
    if (this.contenu>0)  
        dessine_star(this.x,this.y,0.2,this.contenu, this.couleur);  
    else  
        dessine_losange(this.x,this.y,0.2,this.contenu,this.couleur);  
}
```

# Résultat

## Squelette du jeu 2020



# Adaptation et gestion des événements

- Il nous reste à définir la gestion du click de l'utilisateur pour mettre en surbrillance les éléments sélectionnés et vérifier la somme des étoiles et valider si c'est gagné.
- Comment faire ?????

# Adaptation

- On va se positionner dans la fonction `addEventListener` et une fois que l'on a bien les coordonnées sélectionnées `x,y` il faut pouvoir vérifier à qui appartient ces coordonnées.
- Pour réaliser ce traitement, on va balayer l'ensemble des objets graphiques à partir de notre variable `liste_cellules` et pour chaque cellule vérifier cette condition. Et une fois que c'est fait on appelle la fonction `trace_grille()`.
- Donc il nous faut définir pour la classe `Cellule` une nouvelle méthode `verif()` qui reçoit des coordonnées `x,y` et si ces coordonnées sont internes à la cellule elle change de couleur.

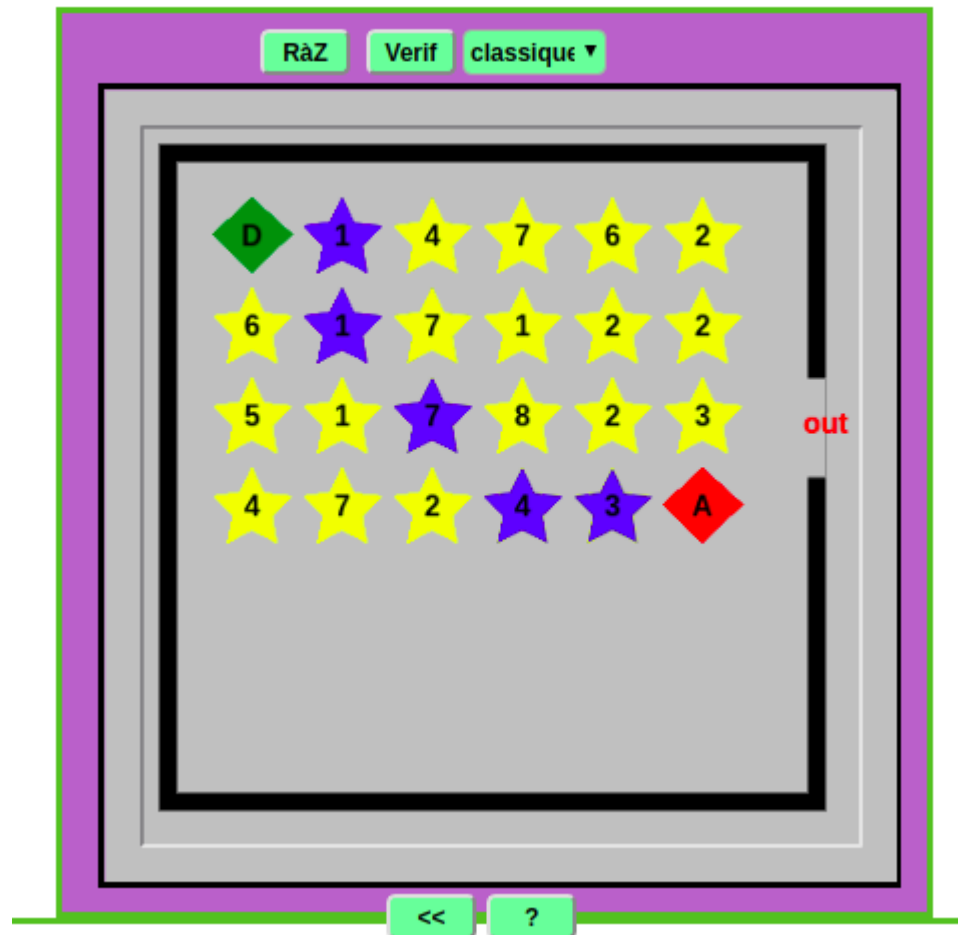


# Méthode verif() classe Cellule

```
verif(x,y) {  
  if ((x>=this.x && x <=this.x+50) &&(y>=this.y && y <=this.y+50)) {  
    if (!this.statut) {  
      this.couleur="blue";  
      this.statut=true;  
      console.log("j ai trouve");  
    } else {  
      this.couleur=this.couleurOrig;  
      this.statut=false;  
    }  
  }  
}
```

# Résultat

## Squelette du jeu 2020



# Finalisation du projet

- Il nous reste quelques petite étapes à terminer :
  - Permettre de réinitialiser le jeu (RàZ)
  - La vérification (bouton verif)
  - Indiquez le succès de l'utilisateur (Gagné)
  - Définir une séquence de jeux possibles
- Habillage de PWA

# Fonction RaZ

- Le bouton est actuellement connecté sur la fonction choixRaz()
- Dans cette fonction il nous suffit de réinitialiser le tableau des objets et de redessiner ce qui nous donne les instructions suivantes :

```
function choixRaz() {  
    liste_cellules=[];  
    charge_grille();  
    trace();  
}
```

# Fonction verif()

- Objectif est de vérifier les séquences que l'utilisateur a joué et d'indiquer celles qui ne font pas parties de la solution.
- Dans la configuration du jeu nous avons fourni la solution qu'il nous suffit de contrôler et de comparer avec les étoiles dont le statut est à true.

# Ce qui nous donne

```
function verif(){
  var i=0;
  configuration.solution.forEach(e=> {
    e.forEach(ee=> {
      if (ee==0 && liste_cellules[i].get_statut())
        liste_cellules[i].erreur();
      i++;
    });
  });
  trace();
}
```

Dans la classe Cellule nouvelle méthode à insérer

```
erreur() {
  this.couleur="red";
}
```

# La fonction gagnée

- Comment vérifier que l'utilisateur à bien gagné la partie

# Fonction Somme

- Afficher la somme cumulée des sélections que vient de faire l'utilisateur et si la somme est égale à la valeur recherchée et qu'en plus il n'y a pas d'erreur c'est donc le bon chemin.

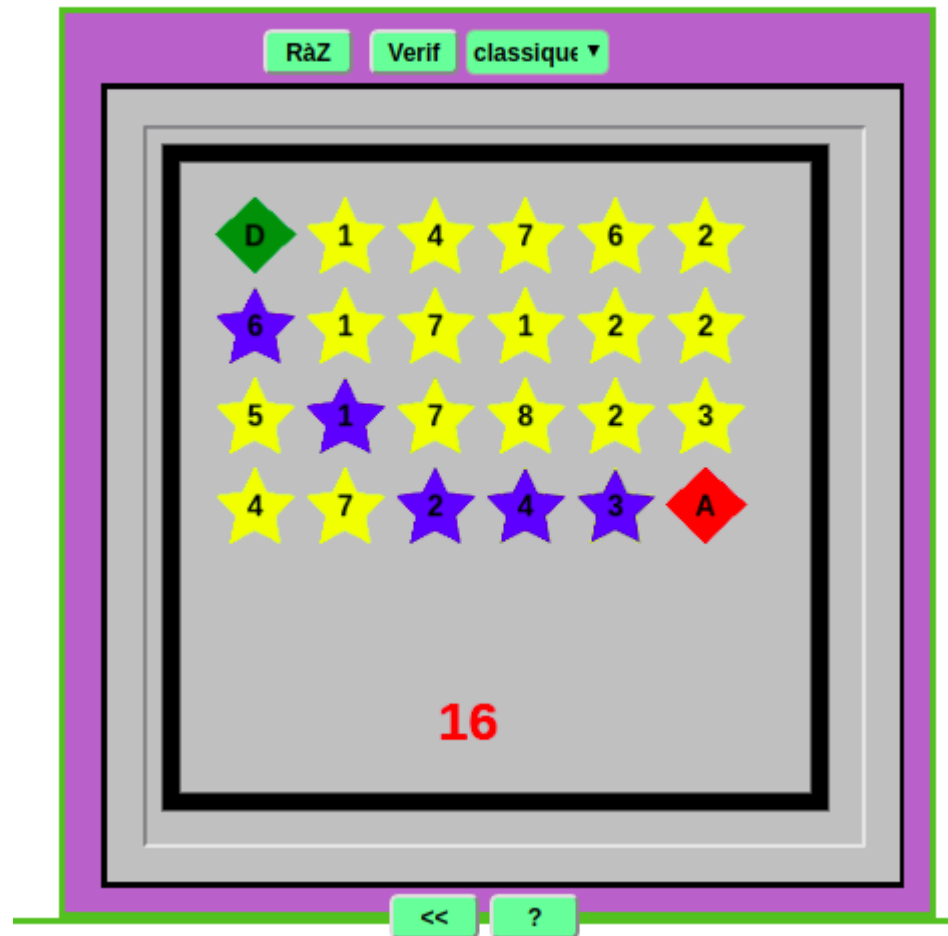
```
function somme() {  
    Vsomme=0;  
    liste_cellules.forEach(e=> {  
        if (e.get_statut()) {  
            Vsomme+=e.contenu;  
        }  
    });  
}
```

Ajouter l'appel de cette fonction dans le module d'écoute et afficher le résultat dans `trace_grille()`



# Résultat

## Squelette du jeu 2020



# Installation et mise en place de PWA

- On s'inspirant du site [pwa-rocks](#) et de l'application 2048 on va pouvoir définir le fichier manifest.json de l'application et les fichiers de configuration de notre service worker pour un usage simple du cache.

# Fichier manifest.json

```
{
  "short_name": "BC",
  "name": "BonChemin",
  "icons": [
    {
      "src": "/meta/touch/icon-128x128.png",
      "sizes": "128x128",
      "type": "image/png"
    },
    {
      "src": "/meta/touch/icon-192x192.png",
      "sizes": "192x192",
      "type": "image/png"
    },
    {
      "src": "/meta/touch/icon-256x256.png",
      "sizes": "256x256",
      "type": "image/png"
    },
    {
      "src": "/meta/touch/icon-384x384.png",
      "sizes": "384x384",
      "type": "image/png"
    },
    {
      "src": "/meta/touch/icon-512x512.png",
      "sizes": "512x512",
      "type": "image/png"
    }
  ],
  "start_url": "/",
  "display": "standalone",
  "background_color": "#ECC402",
  "orientation": "portrait"
}
```

A rajouter dans le fichier index.html

Dans l'entête : head

<link rel="manifest" href="manifest.json">

# Le fichier de chargement de la configuration du worker

```
if ('serviceWorker' in navigator) {  
  navigator.serviceWorker.register('sw.js').then(function(registration) {  
    console.log('ServiceWorker registration successful with scope: ',  
registration.scope);  
  }).catch(function(err) {  
    // registration failed :(  
    console.log('ServiceWorker registration failed: ', err);  
  });  
}
```

# Le fichier sw.js

```
'use strict'
const CACHE_NAME = 'cache-bonchemin';
// The files we want to cache
const resourceList = [
  '/',
  'index.html',
  '/css/style.css',
  '/scripts/comportement.js',
  '/scripts/config.js',
  '/meta/touch/icon-128x128.png',
  '/meta/touch/icon-192x192.png',
  '/meta/touch/icon-256x256.png',
  '/meta/touch/icon-384x384.png',
  '/meta/touch/icon-512x512.png'
];

self.addEventListener('install', event => {
  event.waitUntil(caches.open(CACHE_NAME).then(cache => {
    return cache.addAll(resourceList);
  }));
});

function addToCache(cacheName, resourceList) {
  caches.open(cacheName).then(cache => {
    return cache.addAll(resourceList);
  });
}

self.addEventListener('fetch', event => {
  event.respondWith(caches.match(event.request).then(response => {
    return response || fetch(event.request);
  }));
});
```

On pourrait rajouter l'événement activate pour la gestion des mises à jour éventuelles sur le site