

07 novembre 2023

Sofiene Habib Belkhiria

ETML

Rue de Sébeillon

P-Bull Snake

Snake JavaScript

# VARIABLE

## Syntaxe

* Pas de type
* Mettre « Let » et pas « var » car var est globale au code (une variable « let » agis comme en C#)

Exception

* Dans une classe si l’instanciant d’une variable ne se trouve ni dans le constructeur ni dans une méthode alors la variable ne prend ni « let » ni « var »

## Objet

* Il est possible de créer un objet directement avec une variable en faisant « let nomObjet = {varible1 :0 , variable2 : blabla } » dans cette exemple nous créons une objet « nomObjet » avec deux variable

Utilisation

* Comme un objet de classe « nomObjet.variable1 »

# IMPORT/EXPORT/FONCTION

## Syntaxe Import

* Pour importer un module ou un fichier voici la syntaxe « import ‘cheminfichier’ ; »
* S’il ne faut importer qu’une classe parmi un fichier qui en comporte plusieurs il est possible d’utiliser cette syntaxe « import {nomClasse} from ‘cheminFichier’ »

## Syntaxe Export

* Pour exporter une classe il vous suffit de taper « export » suivi de l’instanciation de classe comme ceci : 

## Fonction

* Les fonctions JavaScript agissent comme des méthodes en C# elle se déclare comme ceci



* A noté que les paramètre et return agissent comme en C#

# Tableau

## Syntaxe

* La déclaration d’un tableau se fait comme une variable normale cependant pour indiquer que l’on parle d’un tableau il est primordial de le mettre après le signe égale comme ceci : 

## Ajout d’élément

* Afin d’ajouter un élément dans un tableau il est possible d’utiliser la méthode « push () » comme ceci : Dans notre cas nous ajoutant un objet ayant les propriété « X et Y » à la fin du tableau cependant la méthode «push() » peut aussi ajouter des simple valeur.
* Il est aussi possible d’utiliser la méthode « unshift() » si vous voulez ajouter un élément au début du tableau

## Suppression d’élément

* Si vous voulez supprimer un élément précis du tableau il est possible d’utiliser la methode « splice() » comme ceci : celle-ci fait en sorte de supprimer un nombre définis d’élément en partant d’un indice dans notre exemple deux éléments seront supprimer à partir de l’indice 2, les élément 3 et 4 seront donc supprimer.
* Si vous voulez supprimer uniquement le premier élément d’un tableau il vous faut utiliser la méthode « pop () » qui supprime le dernier élément du tableau. Comme ceci :



* Dans le cas de la suppression du premier élément du tableau il est conseillé d’utiliser la méthode « shift () » comme ceci :



# Switch Case

## Utilité

* Le switch case pourrais s’apparenter à une suite de if, cependant ceux-ci ne change que dans un cas bien précis

## Syntaxe

* La syntaxe du switch case se compose de cette manière :
* switch (event.key) {
* // change la direction du serpent vers le haut si la flèche du haut est touché
* case 'ArrowUp':
* if (direction != 'Down') {
* direction = 'Up'
* }
* break;
* // change la direction du serpent vers le bas si la flèche du bas est touché
* case 'ArrowDown':
* if (direction != 'Up') {
* direction = 'Down';
* }
* break;
* // change la direction du serpent vers la gauche si la flèche de gauche est touché
* case 'ArrowLeft':
* if (direction != 'Right') {
* direction = 'Left';
* }
* break;
* // change la direction du serpent vers la droite si la flèche de droite est touché
* case 'ArrowRight':
* if (direction != 'Left') {
* direction = 'Right';
* }
* break;
* }
* Dans notre cas (event.key) correspond à la variable dont nous allons vérifier l’état, ce qui veut dire que dans la valeur de la variable le code s’exécutant sera différent.

# Ecouteur d’événement

## Utilité

* Les écouteur d’événement ou EventListener permet d’écouter des événements se produisant dans un élément DOM (Document Object Model)

## Syntaxe

 window.addEventListener('keydown', Touche);

* Cette eventListener indique que nous voulant écouter les événements ayant lieu dans la fenêtre du navigateur et qui sont de type « keydown » puis elle indique que nous utilisant la méthode Touche.
* Cela veut donc dire que à chaque fois que l’utilisateur appui sur une touche la méthode « touche » s’exécutera.

# Rapport Snake

## Introduction

* Cette partie de l’aide-mémoire a pour but de passer en revue la manière dont le jeu du Snake en JS a été produit
* A noter que ce jeu à servis d’introduction au Java Script, cela veut dire que toutes les notions vues plus tôt seront donc abordée dans cette également mais seront donc expliquer du point de vue du jeu.

## Classe Serpent.js

Paramètre

* La classe Serpent prend en compte 4 paramètres
* La Position x du serpent.
* La position y du serpent.
* La taille de départ du serpent.
* L’état du serpent (mort ou vivant)
* Au final le constructeur de la classe ressemble à ceci :
* constructor(x, y, taillePartie,dead) {
* this.serpentX =x;
* this.serpentY =y;
* this.nombrePartie = taillePartie;
* this.dead = dead;
* }

Méthode : drawSnake

drawSnake(width /\*largeur de la tête\*/, height /\*hauteur de la tête\*/, serpentParts /\*nombre de partie du serpent\*/, snakeTete /\*tête du serpents\*/) {

    let canvas = document.querySelector('canvas');

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

    // couleur de la tête

    ctx.fillStyle = 'blue';

    // dessine la tête

    ctx.fillRect(snakeTete.x, snakeTete.y, width, height)

    // dessine le couleur

    for (let i = 0; i < serpentParts.length; i++) {

        // couleur du corps du serpent

        ctx.fillStyle = 'green';

        // dessine le corps du serpents

        ctx.fillRect(serpentParts[i].x, serpentParts[i].y, width, height);

    }

  }

* Le but de cette méthode est de dessiner le serpent, elle prend en paramètre la taille (largeur/Hauteur) de chaque partie du serpent, la tête du serpent.
* La méthode permet de dessiner la tête du serpent en bleu grâce à ce code :
* ctx.fillStyle = 'blue';
* // dessine la tête
* ctx.fillRect(snakeTete.x, snakeTete.y, width, height)
* Le reste de la méthode permet dessiner le reste du corps du serpent en vert grâce à cette boucle :
* for (let i = 0; i < serpentParts.length; i++) {
* // couleur du corps du serpent
* ctx.fillStyle = 'green';
* // dessine le corps du serpents
* ctx.fillRect(serpentParts[i].x, serpentParts[i].y, width, height);
* }
* A noté que « serpentParts » est un tableau contenant le corps du serpent.

Méthode : move

* La méthode move a pour but de faire bouger le serpent, voici son récapitulatif.
* move(serpentParts, tete,direction){
* // valeur par la quelle le serpent se déplacera
* const toMove = 40;
* // tête du serpent
* let teteSerpent ={x:tete.x, y:tete.y};
* // partie 0 du corps du serpent attribuer à la tête du serpent
* serpentParts[0] = teteSerpent;
* // déplacement du corps en fonction de la position 0(tête)
* for(let i = serpentParts.length -1 ; i> 0; i--){
* //mouvement
* serpentParts[i].x = serpentParts[i-1].x;
* serpentParts[i].y = serpentParts[i-1].y;
* }
* // attribution de la position x et y de la tête a la partie 0 du corps
* serpentParts[0] = teteSerpent.x;
* serpentParts[0] = teteSerpent.y;
* // changement de direction du serpent
* switch (direction) {
* case "Up":
* tete.y -=toMove;
* break;
* case "Down":
* tete.y +=toMove;
* break;
* case "Left":
* tete.x -=toMove;
* break;
* case "Right":
* tete.x +=toMove;
* break;
* }
* }

La méthode peut se décomposer en deux partie, la première est accès sur la boucle for :

for(let i = serpentParts.length -1 ; i> 0; i--){

      //mouvement

      serpentParts[i].x = serpentParts[i-1].x;

      serpentParts[i].y = serpentParts[i-1].y;

    }

Cette boucle fait en sorte que chaque partie du corps suivent les coordonnées de celle se trouvant devant elle se qui permet au serpent de se déplacer de manière homogène.

La deuxième partie consiste en un « switch case » déterminant vers quelle direction le serpent va se diriger.

Méthode : serpentDead

* Cette méthode a pour but de tuer le serpent s’il touche le bord du terrain ou s’il se touche lui-même.
* Afin de vérifier si le serpent touche le bord nous utilisons une condition if comme celle-ci :
* // mort du serpent si il touche le bord
* if ((snakeTete.x < 0) || (snakeTete.x >= 800) || (snakeTete.y < 0) || (snakeTete.y >= 800)) {
* this.dead = true;
* }
* Cette condition compare la position x et y de la tête avec la position des bords et tue le serpent si elles correspondent
* Afin de vérifier si le serpent se touche lui-même nous utilisant une condition dans une boucle comme ceci :
* // mort du serpent si il se touche lui même
* for(let i = 0; i < serpent.length -1; i++){
* if(snakeTete.x == serpent[i].x && snakeTete.y == serpent[i].y){
* this.dead = true;
* }
* }
* Cette boucle prend la position de chaque partie du corps du serpent et les comparent aux coordonnées de la tête puis tue le serpent si elles correspondent.

## Classe Apple.js

Paramètre

* La classe Apple.js prend en compte 2 paramètres
* La position x de la pomme.
* La position y de la pomme.
* Le constructor de la pomme est donc comme ceci :
* constructor(x,y){
* this.appleX = x;
* this.appleY = y;
* }

Méthode : drawApple

* La méthode drawApple a pour but de dessiner une pomme rouge sous forme de carré de la même taille que la tête du serpent.
* Elle se présente comme ceci :
* drawApple(x,y,w,h) {
* let canvas = document.querySelector('canvas');
* let ctx = canvas.getContext('2d');
* // couleur de la pomme
* ctx.fillStyle = 'red';
* // position de la pomme
* ctx.fillRect(x, y, w, h)
* }

Méthode : randomApple

* La méthode randomApple sert à attribuer une position aléatoire à la pomme, elle se présente comme ceci :
* // détermination aléatoire de la position de la pomme
* randomApple(width) {
* this.appleX = Math.floor(Math.random() \* 10) \* width;
* this.appleY = Math.floor(Math.random() \* 10) \* width;
* }

## Main.js