Idée de conception Game And Watch Donkey Kong

Introduction

A notre début de S4 PEL, nous avons choisi ce projet tutoré de Game and Watch dans le but de pouvoir découvrir un nouveau langage (qui n’a pu être abordé que brièvement).

Notre principale motivation fut donc la découverte d’un nouveau langage (Javascript), associé avec de nouveaux concept (sprites, canvas), le tout en équipe. C’est durant ce projet que nous avons découvert le logiciel GIT, permettant ainsi un partage du code facile en évitant les pertes d’informations.

Phase de tutoriels:

Au début du S4, notre cursus ne nous avait pas donné l’occasion de faire du JavaScript, c’est pour cela que nous avons dû nous initier à ce langage sur quelques tutoriels mêlant affichage, événements et interactions.

Pong (HTML5 canvas + Javascript)

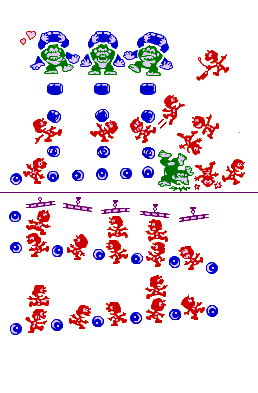
**Le canvas**  
Nous avons d’abord commencé par découvrir ce qu’était un canvas.   
C’est un cadre d’une dimension donnée qui va nous permettre de dessiner des images à l’intérieur.  
Dans ce premier tutoriel, nous n’avons dessiné que de simples formes (rectangle blancs, texte)  
  
**Séparer les types d’entités**  
Nous avons également appris une règle sur les canvas :  
On va séparer les différents types d’entités sur différents canvas.  
Dans notre tutoriel : On a séparé le terrain (background), les raquettes+balles (joueurs), et le score (jeu)  
Cela permet une meilleure fluidité, évite de rappeler inutilement l’affichage d’une partie du code, et rend le code plus lisible.

Sprite Pièce (Javascript)

**Update/Render**  
Ce second tutoriel nous a introduit au concept de sprite.  
Ce qu’on retiendra de ce tutoriel et la séparation entre mise à jour logique et graphique.  
Pour chaque sprite, on aura alors une fonction **Update** qui fera la mise à jour logique (modifie la valeur des variables en fonction de la situation) et une fonction **Render** qui se chargera d’afficher sur les canvas les différents sprites. La seconde fonction est beaucoup plus simple que la première car ne fait pas de changements de variables.  
  
**Feuille de sprite**  
On a donc été introduits au concept de feuille de sprites, où on va insérer chaque image de l’animation souhaité dans celle-ci. On affiche qu’une partie de l’image à chaque render et on modifie la partie a chaque update.

Conception du jeu:

Les différentes classes

Diviser les classes par entités, chaque entité dispose de sa feuille de sprite (plus facile pour afficher)  
Le tout sera lancé dans l’ordre suivant depuis la page html  
  


**Entités du jeu :**  
-Mario (Déplacement clavier, collisions)  
-Donkey Kong (+les mains de DK qui ne lance pas)  
-Tonneaux (+les mains de DK qui lance, dépend de la position de DK)  
-Obstacle   
-Interrupteur  
-Grue  
-Crochet   
-Plateforme  
-Vie de DK  
-Vie de mario  
-Score  
  
**Manette :**  
-Bouton  
-Pad   
  
**Fonction principale :  
-**Game (gameLoop, génération des objets ci dessus)

Différents événements de jeux

**Mort :**  
Déclenchement :   
On avance sur un tonneau  
Un tonneau avance sur nous  
On avance dans le vide (2e étage)  
On loupe la grue lors d’un saut (Animation différente)  
On saute et percute un obstacle  
Conséquence :  
Perd une vie (Game over si 0 vie)  
Réinitialise la position du joueur, mais ne réinitialise pas les tonneaux (game and watch original)  
  
**Victoire :**  
Déclenchement :  
On saute sur le crochet   
Conséquence :  
Retire un crochet à DK (+20pts) (Animation)   
Si plus de crochet, Victoire (+20 pts) et réinitialise crochet

**Saut :**Déclenchement :  
Touche de saut sur un emplacement correct  
Conséquence :  
Si on saute pour la grue : Tester le crochet, victoire ou mort  
Saute au dessus d’un tonneau : 1er palier->1pt 2e palier->2pts

**Déplacement de DK :**Déclenchement :   
Au bout de x frames, DK se déplace soit à droite, soit à gauche (Si peut pas se déplacer, bouge pas)

**Apparition tonneau :**Déclenchement :  
Aléatoire, pars de la position de DK  
Conséquence :  
Apparition d’un tonneau suivant le pattern défini  
REMARQUE : Plus le temps passe, plus le jeu s’accélère !

**Apparition obstacle :**Déclenchement :   
Aléatoire  
Conséquence :  
L’obstacle parcoure la longueur, si mario le touche en sautant, mort  
REMARQUE : Déplacement ne se faisant pas en même temps que les tonneaux !!  
  
**Interrupteur :**Déclenchement :  
Mario actionne l’interrupteur en avançant dans sa direction.  
Conséquence :  
La grue est actionnée, elle va faire 2 allers retours avec son crochet (5 positions), puis se désactive.

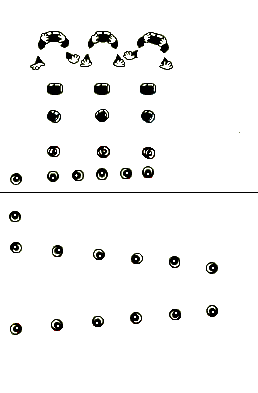
**Animation perd :**Clignote sur sa position plusieurs fois de suite

**Animation tombe :**Trébuche  
Tombe  
Clignote (Utiliser animation perd)

**Animation de victoire :**Saute sur le crochet  
Lève la grue  
Rabaisse la grue  
Retour au début  
  
**DK est battu :**DK clignote  
DK tombe avec les barres  
Cœur de la princesse qui s’anime

Positions des sprites :

#### Tonneaux (Classe barrel)

  
Classe gérant la génération et le défilement des tonneaux.  
La génération d’un tonneau dépend de la position de donkey Kong

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 (DK 0)  [38,31]  L=31 H=19 |  | 4 (DK 1)  [88,31]  L=30 H=18 |  | 8 (DK 2)  [137,32]  L=32 H=21 |
|  | 1  [47,83]  L=16 H=12 |  | 5  [94,83]  L=16 H=12 |  | 9  [140,83]  L=141 H=110 |
|  | 2  [47,110]  L=14 H=13 |  | 6  [95,109]  L=14 H=13 |  | 10  [141,110]  L=14 H=13 |
|  | 3  [47,146]  L=13 H=11 |  | 7  [97,146]  L=13 H=12 |  | 11  [142,147]  L=13 H=12 |
| 17  [10,173]  L=12 H=12 | 16  [47,171]  L=12 H=11 | 15  [72,170]  L=12 H=11 | 14  [96,168]  L=12 H=11 | 13  [120,168]  L=12 H=11 | 12  [142,166]  L=12 H=12 |
| 18  [9,211]  L=12 H=11 |  |  |  |  |  |
| 19  [10,242]  L=12 H=11 | 20  [52,245]  L=11 H=12 | 21  [93,249]  L=12 H=11 | 22  [131,249]  L=12 H=11 | 23  [169,256]  L=11 H=12 | 24  [206,262]  L=12 H=12 |
| 30  [10,325]  L=12 H=12 | 29  [51,319]  L=12 H=12 | 28  [92,316]  L=12 H=11 | 27  [130,312]  L=12 H=12 | 26  [169,309]  L=12 H=11 | 25  [206,305]  L=12 H=12 |

**Variables globales :**  
barrels[] : tableaux de sprite  
barrelX : valeur x pour chaque case ci-dessus (constante)  
barrelY : valeur y pour chaque case ci-dessus (constante)  
barrelL : valeur L pour chaque case ci-dessus (constante)  
barrelH : valeur H pour chaque case ci-dessus (constante)

**Objet Sprite**  
-positionBarrel : entre 0 et 30  
-positionDK : entre 0 et 2 (position d’où part le tonneau)  
+update : fais avancer le barrel selon le trajet défini (utilise tableaux ci-dessus)  
 +fais le test de collision avec mario  
+render : affiche le barrel

Fonction génererBarrel(positionDK) : génère un barrel en fonction de la position de DK

#### Classe DK : C:\Users\loick\Documents\GitHub\gameAndWatch\img\sprites\old-dk.png

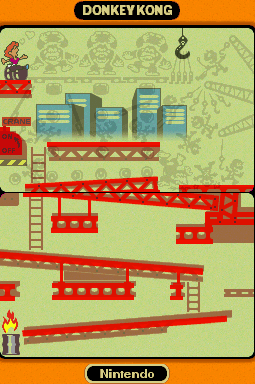
Classe gérant le déplacement de donkey kong et de ses mains lorsqu’il ne lance pas de barrel.  
DK dispose de 4 positions. Il ne peut accéder à la position 4 que quand il n’a plus de vie (il tombe)

**Variables globales :**dk, mains

Position DK :  
dkX=[42,89,129]  
dkY=[44,44,45]  
dkL=[33,29,35]  
dkH=[30,31,31]

Position mains :  
mainX=[30,66,80,113,123,159]  
mainY=[56,52,54,54,50,57]  
mainL=[15,14,12,12,15,14]  
mainH=[14,12,13,13,10,13]  
dk   
mains de dk  
  
function spriteDK : Gere le sprite de DK et le déplace toutes les 50 frames  
function spriteMain : Fait le rendu des mains de DK en fonction de sa position

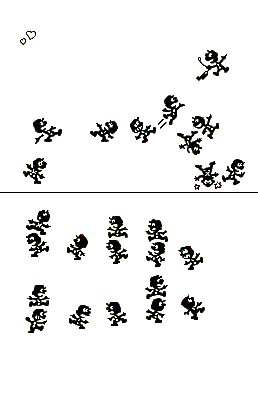
#### Classe game :



**function gameLoop :**   
 boucle du jeu faisant l’ensemble des update et render sur les différentes entités  
**function update :**  
 gère quand DK lance un barrel   
 gère l’apparition des obstacles  
  
Canvas : gère les sprites  
Canvas2 : gère le background  
Canvas3 : gère le pad  
Canvas4 : gère le bouton de saut

#### 

#### Classe mario



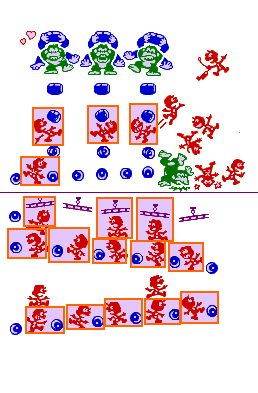
Gestion des déplacements et des sauts de mario  
On initialise ici les événements clavier

**Touche directionnelles et effets sur le déplacement:**

**Droite:** 0123 (+1) 6789 (-1) 12,13(+1) 14(chute) **Gauche :** 1234 (-1) 5678(+1) 12(levier) 13,14(-1) **Haut :** 4,9,10,11(+1) **Bas :** 5,10,11,12(-1) **Saut : 0,3,6,7**

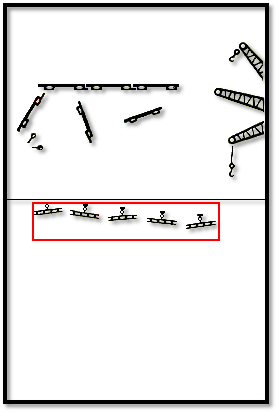
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Saut | Grue | Chute | Safe zone | Chemin |  | 19 |
|  |  |  |  |  | 18 |  |
| 12-> | -> | 13 -> | -> | 14 ^ -> |  | 20 |
| ^11 |  |  |  |  | 15 |  |
| ^10 |  | 24 | 23 |  | 16 |  |
| ^9 | <- 8 | <- 7 | <- 6 | <-5 |  |  |
| 21 |  |  | 22 | ^ |  |  |
| 0 -> | 1 -> | 2 -> | 3 -> | ^4 |  |  |

MarioX=[25,69,106,146,181,  
178,146,106,67,26,26,  
23,33,90,130,  
191,177,194,  
158,196,222,  
27,145,146,107]  
MarioY=[309,305,302,299,294,  
246,242,240,234,233,210,  
158,118,120,117,  
115,131,161,  
95,50,159,  
285,275,219,216]  
MarioL=[22,22,21,20,24,  
25,19,22,21,23,25,  
24,29,28,26,  
29,25,28,  
27,33,23,  
23,23,22,21]  
MarioH=[24,24,24,24,26,  
24,25,24,26,24,25,  
26,26,22,26,  
24,24,29,  
34,31,26,  
22,23,23,22]

**Collisions**  
  
  
Si tab[posMario]==posBarrel -> collision  
var collisionBarrel=[29,28,27,26,25,24,23,21,20,19,-1,16,2,6,10];

Collisions à établir pour quand mario se déplace et quand les barrels se déplacent :  
Barrel -> Vérifie à chaque fois déplacement  
Mario -> En fonction du déplacement :  
 Si avance (+1) : test avec case actuelle  
 Si recule (-1) : Doit vérifier la collision avec la case précedente  
  
On réalise de la même manière les collisions avec les obstacles

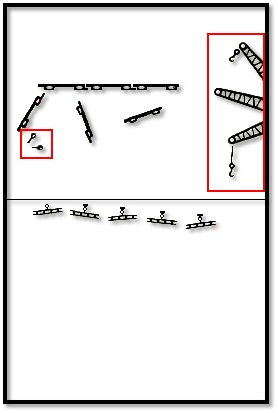
#### Obstacles



Se déclenche toutes les 30 secondes. (instance créée)  
Effet : Si mario se trouve sur l’emplacement approprié, il perd une vie et repart à la position initiale.

var obstacleX= [179,140,101,63,26]  
var obstacleY= [207,204,200,197,197]  
var obstacleL= [30,30,29,29,30]  
var obstacleH= [15,14,14,15,11]

#### Grue et bouton



**Grue**  
3 positions (désactivé, activé, gagne)  
1 instance toujours présente

var grueX=[221,207,222] ;  
var grueY=[114,81,35] ;  
var grueL=[36,50,34] ;  
var grueH=[56,24,39] ;  
  
**Bouton**2 positions (activé désactivé)Si mario avance en direction du bouton, activation du bouton   
-> Active la grue et démarre le crochet  
  
var boutonX=[20,25]  
var boutonY=[126,138]  
var boutonL=[9,11]  
var boutonH=[10,5]

Crochet :

On utilise ici une feuille de sprite qui n’est pas directement bien positionné car on aurait des chevauchements (On en reparlera plus loin dans les problèmes et solutions).  
On multiplie la longueur par la position pour obtenir l’image souhaitée.  
  
C:\Users\loick\Documents\GitHub\gameAndWatch\ressources\DIAPO\sprites\crochets.png

5 positions, instancie lorsque le bouton est actionné.  
Fais 2 aller retours avant de se détruireEffet : Débloque la possibilité de gagner  
Effet destruction : Désactivation du bouton

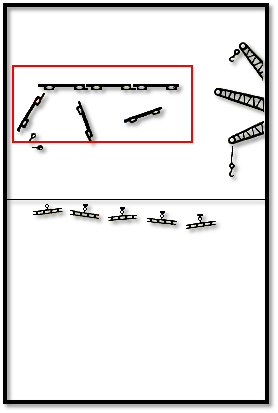
var spriteCrochetX= [0,21,42,63,84] ;  
var spriteCrochetY= [0,0,0,0,0] ;  
  
var crochetX=[183,192,205,212,218] ;  
var crochetY=[86,89,91,93,94] ;  
var crochetL=[21,14,6,11,16] ;  
var crochetH=[17,22,24,21,13] ;

**Vie DK :**

C:\Users\loick\Documents\GitHub\gameAndWatch\ressources\DIAPO\sprites\crochets.png  
  
De la même manière que les crochets, on utilise une feuille de sprite différentes pour éviter les chevauchements.  
4 instances, se détruise une par une lorsqu’on gagne  
Si les 4 instances détruites, animation de victoire

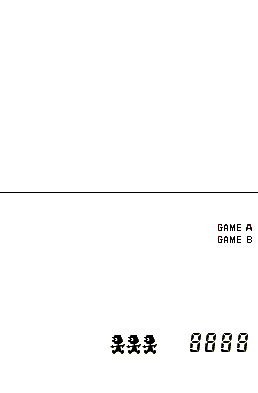
var plateformeX = [143,10,69,117] ;  
var plateformeY = [87,86,94,100] ;  
var plateformeL = [141,28,17,38] ;  
var plateformeH = [6,39,42,16] ;

var vieDKX = [172,178,184,189] ;  
var vieDKY = [51,51,51,51] ;  
var vieDKL = [10,6,6,7] ;  
var vieDKH = [33,33,33,33] ;

Plateforme  


Classe à but purement décoratif.   
Pas d’incidence sur les autres classes.  
Change de position que lorsque DK tombe

**Vie Mario :**

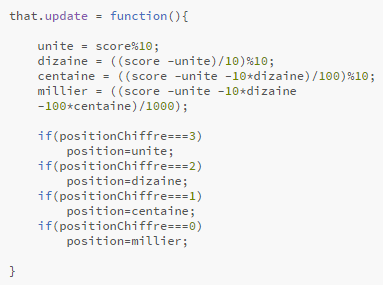
var vieX = [108,124,140] ;  
var vieY = [341,341,341] ;  
var vieL = [15,15,15,15] ;  
var vieH = [19,19,19,19] ;

Simple compteur de vie, on fait un tableau de 3 vie en début de partie, on splice le tableau dès qu’on retire une vie, et si le tableau est vide, game over.

#### Score

**C:\Users\loick\Documents\GitHub\gameAndWatch\ressources\DIAPO\sprites\score.png**

ScoreX= [188,203,219,235]  
ScoreY= 339  
ScoreL=11  
ScoreH=19  
  
Quelques petites difficultés pour obtenir un à un les chiffres du score.

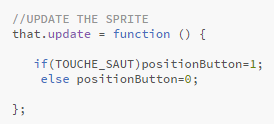


#### Bouton + Pad

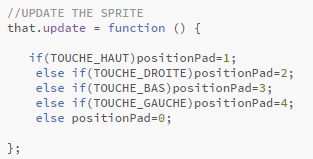
  
C:\Users\loick\Documents\GitHub\gameAndWatch\img\sprites\pad\button.png

Il a fallu ici créer des nouveaux canvas et les positionner correctement pour pouvoir créer une animation sur les boutons de la console.  
  
Ici on a un simple update/render qui change en fonction de la touche appuyée.

**Update du bouton :**



**Update du pad :**



**Note :**Décalage à faire entre les deux écrans + rogner le canvas / Mettre transparence sur le fond ?

Difficultés rencontrées:

**Mise en place du git :**Au début du S4, nous n’avons pas encore été initiés correctement à git.  
Il a fallu apprendre par nous même à utiliser cet outil de gestion de versions.  
Nous n’avons utilisés que les fonctions principales (add, commit, push, pull)

Le projet était déjà bien avancé lorsque nous l’avons abordé de manière plus poussé en cours.  
  
**Décalage des sprites  
Background qui ne charge pas**

Améliorations possibles:

**CALENDRIER**

**6/02 :** tuto pong + canvas (son+différentes entités sur différents canvas+evenement clavier)

**9/02 :** tuto sprite (structure update, canvas)

**11/02 :** Test de GIT (add, commit, push, pas encore vu en cours)

**15/02 :** Récupération des sprites (Emulateur ds (desmume) game and watch collections) permet d’extraire les sprites manuellement (fallait tous les faire apparaitre pour reconstruire la feuille de sprite)  
Ensuite, il a fallu séparer les sprites pour chaque classes

**16/02 :** Tonneaux qui s’écoulent simplement

**17/02 :** DK + background   
DK génère tonneaux en fonction de sa position (DK + mains)

**18/02 :** Mario déplacement

**20/02 :** Mario déplacement, saut + collisions tonneaux

**03/03 :** Animations, Simplification du code, plateforme

**05/03 :** Ajout du score, Simplification code

**09/03 :** Mise en forme du site / Adaptation Sprite à la console (décalage)

**14/03 :** SQLite + tableau score

**19/03 :** Répétition des touches, vies de DK, Son

**24/03 :** Canvas des boutons

**25/03 :** Prototype 2 joueurs

**29/03 :** Mise en place HighScore

**30/03 :** Correction background, position score

Concernant la page des High Scores

Cette page est destinée a accueillir un classement des meilleurs scores réalisé.

Nous avons donc vu la chose comme une simple base de donnée avec une colonne id pour différencier chaque score, une colonne nom ou pseudo pour pouvoir repérer les pseudos des différents joueur ayant réalisé un score, et enfin un score qui indique le résultat obtenu a la fin de votre partie.

Comme notre jeu devait être en JavaScript, nous voulions créer une base de données en JavaScript grâce a sql.js qui est une adaptation de Sqlite3 mais en JavaScript.

Sqlite3 est une autre méthodes de gérer une base de donnée, le gros avantage qu'elle a est d'être vraiment minuscule et de se stocker dans un fichier qui ne pèse pas plus de 10ko (Cela dépend de la taille de la base de donnée bien entendu).

Elle utilise le langage SQL et permet de faire toute sorte de requête pour gérer une base de donnée.

Cela correspondais totalement à nos attentes à savoir, manipulable avec du JavaScript, une mise en place relativement simple, et sur ultra compacte.

Après avoir créer la base de donnée grâce a un addons Firefox (SqliteManager) nous nous sommes attaqué a la manipulation de données.

Dans un premier temps nous avions rempli la base de donnée avec des score et des noms aléatoire histoire de la remplir.

Ensuite est arrivé la partie JavaScript, nous avons eu quelques soucis étant donnée que la solution était un tableau de tableau associatif de tableau de tableau, alors pour récupérer la bonne valeur et la mettre la ou l'on veut il faut chercher un peu.

Mais nous en sommes parvenu et l'affichage réalisé correspondais a nos attentes.

En revanche, pour ce qui était de la sauvegarde de la base de donnée dans un fichier, cela était en fait impossible.

En effet car étant débutant dans le JavaScript et ne connaissant pas toutes les différentes facette de cet univers, j'ai appris après pas mal d'heure de recherche sur différents forum que l'on ne pouvez pas écrire dans un fichier en JavaScript.

Cela nous a donc posé un grand problème qui est que l'on ne pouvais pas sauvegarder la base de données. Donc a chaque actualisation de notre page internet la base de donnée revenait dans son état initial c'est à dire, vide.

Suite à cet échec, nous décidons de mettre en place une base de donnée mais cette fois non pas en JavaScript mais en python qui paraissait tout aussi simple syntaxiquement même peut-être plus simple.

Sauf que pour installer une base de donnée en python il est obligatoire d'avoir un serveur local pour exécuter le code python.

J'ai pour ce faire installer easyPHP un serveur php local qui permet d'herberger un serveur.

J'avais tout pour travailler en Python et faire ma base de donnée, je créer le script en local l’exécute, enfin cela fonctionne. Mais comme c'est un affichage en local cela s'inscrit dans ma console python et non dans ma page web.

Et la encore un autre problème, mon serveur local n'accepte apparemment pas l’exécution de mon script.

Après toute ces déceptions je me résigne donc de faire ma base de donnée en Php toujours grâce a Sqlite, et la déception en déception encore un autre problème mon serveur ne gère pas Sqlite…

Je finis donc à utiliser de manière « classique » la base de donnée sql c'est à dire grace au objet PDO (vu au semestre 3), la encore j'ai eu quelques problèmes pour récuperer les différentes valeur car celle ci sont en javascript. J'ai pour cela utiliser un simple formulaire ou j'ai attribuer à deux « input » différent le pseudo et le score du joueur.

J'utilise la méthode submit() de JavaScript qui permet d' éviter de passer par un bouton submit qui serait gênant pour le joueur.

J'envoie tout cela vers ma page save.php qui insert les valeurs dans ma base de données et ensuite je renvoie l'utilisateur sur la page des highscores pour qu'il puisse voir la où il se situe par rapport aux autres joueurs.