Projet CPA

redaction du jeu "Snake Desert"

Composition du binome
Mickaël Temim
Melanie

Projet CPA

Master 1 STL
Sorbonne université
France
23 avril 2023

Sommaire

Première partie

Présentation du jeu

Ce projet a été réalisé par Mickaël et Melanie.
Ce rapport vous présentera le jeu qui est une version très proche du jeu populaire "Snake" avec quelques ajouts. Pour jouer, ouvrez le fichier "index.html".

1 Regles du jeu

Figure 1 – image du jeu

Bienvenue dans "Snake desert", le jeu où vous contrôlez un serpent à sonnette perdu dans le desert du Sahara subissant de très fortes températures. Cette race de serpent doit impérativement trouver un coin d'ombre ou un terrier afin de se protéger du climat, mais malheureusement vous ne voyez rien de semblable autour de vous. Heureusement pour vous quelques gouttes d'eau semblent tomber du ciel, votre objectif est simple : s'abreuver au maximum en évitant les différents obstacles si vous ne voulez pas érrer definitivement dans ce desert ! Dans ce jeu, vous disposez d'un score ainsi qu'un nombre de points de vies limités qui sera de 3 lorsque vous lancerez le jeu. Essayez donc de récuperer un maximum de points, mais attention, la tâche sera de plus en plus compliquée.

2 Les objets

Afin de vous aider (ou à vous mettre des batons dans les roues) plusieurs objets peuvent apparaître sur l'écran du jeu. Voici leur utilité.

2.1 Le serpent

Figure 2 – serpent

Le serpent est composé de deux parties :
-> D'abord la tête du serpent que vous controlez est représentée par un carré noir. C'est uniquement cette partie du serpent qui sera pris en compte au niveau des différentes collisions.
-> Ensuite, les différentes parties du corps du serpent sont chacune représentées par un carré vert. A chaque fois que vous buvez de l'eau, vous grandirez instantanément. Attention à ne pas vous mordre la queue et à ne pas heurter l'un des quatre cotés de l'écran sinon c'est game over !

2.2 Les gouttes d'eau

Figure 3 – goutte d'eau

La goutte d'eau est un objet essentiel pour progresser dans le jeu. A chaque fois que vous en ramassez une, votre score augmentera de un point. l'objectif est donc très simple : finir le jeu en essayant de ramasser un maximum de gouttes d'eau. Malheureusement votre serpent a très soif et le faite de boire une si petite quantité d'eau l'excitera et ce dernier aura une chance sur trois d'accelerer à chaque fois que vous en ramasserez une. Attention, ça peut aller très très vite !

2.3 Les cactus

Figure 4 – cactus

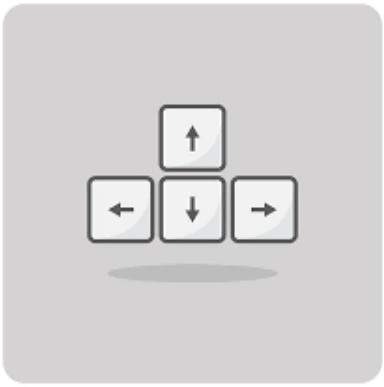
Attention à ne pas heurter un cactus. La pluie en fait pousser un tous les dix points et si vous en touchez un par malheur, vous perdrez une vie et le cactus changera de position. Si votre nombre de points de vie tombe à 0, c'est game over.

2.4 Les coeurs

Figure 5 – coeur

Les coeurs sont rares mais peuvent vous facilitez la tâche. Cet objet vous permet de récupérer un point de vie à chaque fois que vous en attrapez un. Un coeur à une chance sur 3 d'apparaître tous les 20 points. Attention, si vous en voyez un, récupérez le avant de ramasser une goutte d'eau sinon il disparaîtra.

3 Comment jouer ?



Pour jouer c'est très simple, il suffit d'utiliser les fleches directionelles pour se déplacer. Attention vous ne pouvez pas vous déplacer dans le sens inverse de la direction dans laquelle votre serpent se déplace. Par exemple, si votre serpent se dirige vers le haut, vous ne pouvez pas directement vous déplacer vers le bas.
Le bouton "restart" permet de rafraîchir la page.

Deuxième partie

Structure et fonctionnalités du jeu Snake Desert

4 Squelette du projet

4.1 Langage, images et musiques

N'ayant pas l'habitude de programmer en Javascript, nous avons pris la décision d'utiliser ce langage de programmation pour la résalisation de ce projet. Cela nous a permis de nous familiariser d'avantage avec ce dernier.
La goutte d'eau, le cactus ainsi que le coeur sont des images que nous avons déssiné avec l'outil "piskelapp" qui permet de dessiner des sprites. Les images de fond du jeu ont été récupéré sur internet.
La totalité des sons et musiques du jeu ont été réalisé par un étudiant en musique.

4.2 Arborescence

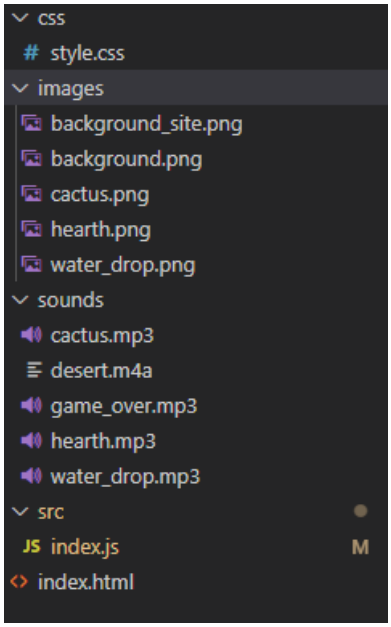


Figure 6 – arborescence

Contenu de l'arborescence :
index.html contient le code html du projet
css/style.css contient le code CSS
src/index.js contient le script javascript
images contient les différentes images
sounds contient les différents sons et musiques présents dans le jeu

5 Algorithmes de collisions

```
function checkCollisionWithTwoSquares(x1, y1, size1, x2, y2, size2) {
    if((x1 + size1 > x2) &&
        (x1 < x2 + size2) &&
        (y1 + size1 > y2) &&
        (y1 < y2 + size2)) {
            return true;
    }
    return false;
}
```

Figure 7 – Algorithme de collision entre deux carrés

Cet algorithhme prend en paramètre les coordonnées ainsi que les tailles des cotés des deux carrés. Il renvoie vrai si ces derniers rentrent en collison, et dans le cas contraire, il renvoie faux. Cela permettra de vérifier constamment si deux objets présents dans le jeu rentrent en collision ou non (comme la tête du serpent et les goutte d'eau par exemple).
Afin de déterminer si deux carrés entrent en collision, il est indispensable de réaliser quatre tests : deux pour examiner les abscisses des carrés et deux autres pour examiner leurs ordonnées. Il convient de souligner que les coordonnées d'un carré sont déterminées par celles du coin supérieur gauche de celui-ci.

Essayons d'analyser l'algorithme d'un peu plus près afin de comprendre comment fonctionne ce dernier.
Soit les coordonnées des deux carrés a(x1, x2) et b(x2, y2). Dans un premier temps, analysons l'axe des abscices. On vérifie si l'absice du coin superieur droit de a est superieur au coin superieur gauche de b. On verifie egalement si l'abscide du coin superieur gauche de a est inferieur au coin superieur gauche de b.
Nous devons faire la même chose avec les ordonnées de a et b. Si ces quatre tests sont valides, alors les deux carrés sont en collision.

```
function checkCollisionWithOneSquareAndAnArray(x, y, sizeSquare, arraySquares, sizeElementOfArray) {
    for(let i = 0; i < arraySquares.length; i++) {
        if(checkCollisionWithTwoSquares(x, y, sizeSquare, arraySquares[i].x, arraySquares[i].y,
            sizeElementOfArray))
            return true;
    }
    return false;
}
```

Figure 8 – Algorithme de collision entre un carré et une liste de carrés

Il existe dans le code, un algorithme partiellement similaire au précédent qui cette fois ci verifie si un carré est en collision avec au moins un carré appartenant à une liste. Par exemple, cela est pratique dans le cas où on souhaite vérifier si la tête du serpent est en collision avec un cactus présent dans le jeu.

Troisième partie
Dessin des objets

6 Le serpent

6.1 Variables

```
class snakePart {
    constructor(x, y) {
        this.x = x;
        this.y = y;
    }
}

let xSnake = canvas.width / 2;
let ySnake = canvas.height / 2;
let health = 3;
let dx = 0;
let dy = 0;
const sizeSnake = 20;
let snakeTail = [];
let sizeTail = 0;
```

Figure 9 – variables représentées par le serpent

La classe **snakePart** représente une partie du corps du serpent (sans compter la tête bien sûr), elle est représenté par ses coordonnées.
Les coordonnées de la tête du serpent sont représentées par **xSnake** et **ySnake** qui sont initialisé au centre de l'écran.
On utilisera également une liste qui contiendra les différentes parties du corps du serpent (on rappelle que la tête ne fait pas partie de la liste vu qu'elle existe forcément). Cette dernière est représenté par **snakeTail** et est associé à sa taille contenu dans **sizeTail**.
Le serpent a également un nombre de point de vie contenu dans **health** qui est initialisé à 3 ainsi qu'une taille qui sera elle fixé à 20 par l'intermediaire de **sizeSnake**.
dx et **dy** sont deux entiers permettant de déplacer le serpent. dx sera positif si le serpent se dirige vers la droite, négatif si il va vers la gauche ou valera 0 si jamais il ne se déplace pas dans l'une de ses directions. Pour dy, la startégie est exactement la même mais dans le cadre du déplacement vertical.

6.2 Dessin

La tête du serpent (carré noir) spawn à chaque itération de la boucle principal du jeu en fonction des coordonnés de cette dernière lors de l'itération précédente ainsi qu'aux valeurs des variables **dx** et **dy** qui désignent le déplacement du serpent.

```
function spawnSnakePart() {
    if(sizeTail > snakeTail.length && sizeTail > 1) {
        let lastIndex = snakeTail.length - 1;
        snakeTail.push(new snakePart(snakeTail[lastIndex].x,
            snakeTail[lastIndex].y));
    }

    for(let i = snakeTail.length - 1; i >= 0; i--) {
        if(i == 0) {
            snakeTail[i].x = xSnake;
            snakeTail[i].y = ySnake;
        }
        else {
            snakeTail[i].x = snakeTail[i - 1].x;
            snakeTail[i].y = snakeTail[i - 1].y;
        }
    }

    if(sizeTail == 1 && sizeTail > snakeTail.length)
        snakeTail.push(new snakePart(xSnake, ySnake));
}
```

Figure 10 – algorithme de dessin du serpent

Avant d'analyser cet algorithme, notez que les fonctions de dessins ne sont pas visibles afin d'améliorer la visibilité de l'algorithme et de nous consacrer uniquement sur les notions plus technique. Il faut savoir qu'à chaque fois que la tête du serpent est en collision avec une goutte d'eau, la variable **sizeTail** est incrémenté. De plus, je vous rappelle que la liste **snakeTail** contient les parties du corps du serpent (les carrés verts), la tête n'est donc pas un élément de la liste et possede ces propres variables que nous avions vu précédemment. Sachez également que la gestion des coordonnées du premier élément de cette liste est différent des autres.

Nous allons comparer la taille de la liste avec la variable **sizeTail** afin de savoir si nous devons ajouter un élément dans notre liste ou non.
Les coordonnées du premier élément de la liste prend les mêmes coordonnées que la tête du serpent. Le déplacement de la tête du serpent est la dernière instruction de la boucle principal du jeu, ce qui permet au premier élément de cette liste d'avoir les mêmes coordonnées que la tête du serpent mais lors de l'itération précédente. C'est cette technique qui nous a permis de créer le serpent.

Veuillez constatez que les autres parties du serpent ont les mêmes coordonnées que l'élément qui les précède. Il est très important de faire cette itération sur chaque élément en partant du dernier jusqu'au premier élément de la liste.

7 La goutte d'eau

7.1 Variables

```
let xWaterDrop = 0;
let yWaterDrop = 0;
const sizeWaterDrop = 40;
let waterIsThere = false;
```

Figure 11 – variables de la goutte d'eau

liste des variables représentant la goutte d'eau :
-> **xWaterDrop** et **yWaterDrop** sont les coordonnées de la goutte d'eau.
-> **sizeWaterDrop** est la taille du coté du carré qui contient l'image de la goutte d'eau
-> **waterIsThere** est une variable booléenne qui indique si la goutte d'eau est présente ou non sur la carte du jeu.
La goutte d'eau est représenté par l'image contenu dans le dossier "images".

7.2 Dessin

```
function spawnWaterDrop() {
    if(waterIsThere == false) {
        do {
            xWaterDrop = Math.floor(Math.random() *
                (canvas.width - sizeWaterDrop));
            yWaterDrop = Math.floor(Math.random() *
                (canvas.height - sizeWaterDrop));
        }while(checkCollisionWithOneSquareAndAnArray(xWaterDrop - 10,
            yWaterDrop - 10, sizeWaterDrop + 20, arrayCactus, sizeCactus));
    }

    waterIsThere = true;
}
```

Figure 12 – algorithme de dessin de la goutte d'eau

La variable **waterIsThere** est fausse à chaque fois que la tête du serpent est en collision avec une goutte d'eau. Dans ce cas, on choisis de manière aléatoire les coordonnées de cette dernière en evitant que la goutte d'eau soit en collision avec l'un des cactus présents sur la carte du jeu. Nous avons choisis d'élargir la taille de la goutte d'eau afin que cette dernière soit assez éloignée des cactus. La dernière instruction de cette fonction permet de nous assurer qu'une seul et unique goutte d'eau soit toujours présente dans le jeu.

8 Les cactus

8.1 Variables

```
class cactus {
    constructor(x, y) {
        this.x = x;
        this.y = y;
    }
}

let arrayCactus = [];
const sizeCactus = 40;
let nbCactus = 0;
```

Figure 13 – variables du cactus

De la même manière que les parties du serpent, chaque cactus est représenté par ses coordonnées grâce à la classe **cactus**.
arrayCactus est la liste contenant les cactus présents sur la carte du jeu.
nbCactus est le nombre de cactus présents sur la carte.
sizeCactus est la taille du coté du carré représentant le cactus.
Chaque cactus est représenté par une image présent dans le dossier "images".

8.2 Dessin

```
function spawnCactus() {
    let maxCactus = Math.floor(score / 10);
    if(nbCactus < maxCactus) {
        do
            newCactus = new cactus(Math.floor(Math.random() * (canvas.width - sizeCactus)),
                                   Math.random() * (canvas.height - sizeCactus));
        while(checkCollisionWithTwoSquares(newCactus.x, newCactus.y, sizeCactus,
                xSnake - 100, ySnake - 100, sizeSnake + 200) &&
              checkCollisionWithTwoSquares(newCactus.x, newCactus.y, sizeCactus,
                xWaterDrop - 10, yWaterDrop - 10, sizeWaterDrop + 20) &&
              checkCollisionWithOneSquareAndAnArray(newCactus.x - 10,
                newCactus.y - 10, sizeCactus + 20, arrayCactus, sizeCactus));
        arrayCactus.push(newCactus);
        nbCactus++;
    }
}
```

Figure 14 – algorithme de dessin du cactus

Un cactus apparaît à chaque fois que le joueur gagne diz points. Lorsque nous séléctionnons les coordonnées du nouveau cactus de manière arbitraire, nous évitons toutes les collisions avec la tête du serpent, la goutte d'eau ainsi que les autres cactus présents dans le jeu. Notez que si le joueur est collision avec un cactus, la variable **nbCactus** sera décrémenté, ce dernier réapparaîtra donc immédiatement à des coordonnées différentes. A notez qu'il est important d'élargir la taille du serpent afin que la catus apparaîsse loin de ce dernier. Cela permet d'éviter les mauvaises surprises si le serpent va trop vite.

9 Le coeur

9.1 Variables

```
let xHearth = 0;
let yHearth = 0;
const sizeHearth = 40;
let hearthIsThere = false;
let isHearthSpawnOnce = false;
```

Figure 15 – variables du coeur

Le coeur à une chance sur trois d'apparaître à chaque fois que le joueur obtient vingt points.
-> **xHearth** et **yHearth** sont les coordonnées du coeur.
-> **sizeHearth** est la taille du carré qui représente le coeur.
-> **hearthIsThere** est une valeur booléenne qui indique si le coeur est présent sur la carte.
-> **isHearthSpawnOnce** est une valeur booléenne qui indique si l'algorithme a tenté de faire apparaître le coeur lorsque le joueur à obtenu ving point. Cela permet de s'assurer que l'algorithme tente seulement une seule et unique fois de faire apparaître le coeur lorsque le joueur a gagné assez de points.

9.2 Dessin

```
function spawnHearth() {
    let rand = Math.random();
    if(!isHearthSpawnOnce && !hearthIsThere &&
            score > 0 && score % 20 == 0 && rand < 0.333) {
        hearthIsThere = true;
        do {
            xHearth = Math.floor(Math.random() * (canvas.width - sizeCactus));
            yHearth = Math.floor(Math.random() * (canvas.height - sizeCactus));
        } while(checkCollisionWithTwoSquares(xHearth, yHearth, sizeHearth,
                xSnake - 50, ySnake - 50, sizeSnake + 100) &&
                checkCollisionWithTwoSquares(xHearth, yHearth, sizeHearth,
                xWaterDrop - 10, yWaterDrop - 10, sizeWaterDrop + 20) &&
                checkCollisionWithOneSquareAndAnArray(xHearth - 10,
                    yHearth - 10, sizeHearth + 20, arrayCactus, sizeCactus));
    }

    if(score > 0 && score % 20 == 0 && !isHearthSpawnOnce)
        isHearthSpawnOnce = true;

    if(score % 20 == 1 && score > 1) {
        isHearthSpawnOnce = false;
        hearthIsThere = false;
    }
}
```

Figure 16 – algorithme de dessin du coeur

La première condition est vrai lorsque le joueur a obtenu vingt points et que l'algorithme tente pour la première fois de faire apparaître le coeur. Les coordonnées du coeur sont choisis de manière aléatoire et ne sera pas en collision avec le serpent, la goutte d'eau ainsi que les cactus. La variable **isHearthSpawn** est remis à faux lorsque le joueur récupère une goutte d'eau après qu'un coeur soit apparu (ou pas). Cela permet à l'algorithme de rééssayer de faire apparaître lorsque le joueur obtiendra vingt points supplémentaires.

Quatrième partie

Conclusion

la réalisation de ce jeu en JavaScript a été une expérience très enrichissante. Elle nous a permis de renforcer nos compétences en programmation, notamment en ce qui concerne l'affichage de rendus à l'écran et la gestion de collisions que nous avions peu travaillé lors de nos années d'études précédentes. Nous avons ainsi appris à utiliser les outils de développement web pour créer un jeu interactif fonctionnel.

Cependant, il y a des améliorations à apporter pour rendre le jeu plus agréable à jouer. On peut constater que les images clignotent, ce qui peut être dû à un problème de rafraîchissement de l'écran ou à une mauvaise mise en page des éléments. Cela est un problème car elle peut nuir à l'experience des joueurs.
L'ajout de nouveaux niveaux peut enrichir le gameplay et rendre le jeu plus intéressant. Cela pourrait être plus stimulant pour le joueur d'ajouter des challenges et des obstacles supplémentaires.

Nous sommes fier de ce que nous avons accompli avec ce projet, mais nous sommes conscient des limites actuelles du jeu.
Ce projet nous a permis de nous entraîner sur des aspects techniques importants de la programmation de jeux, mais il nous a également permis de développer notre créativité et notre capacité à trouver des solutions innovantes pour résoudre les problèmes rencontrés comme pour la gestion des coordonnées des parties du corps du serpent qui nous a beaucoup compliqué la tache dans la réalisation de ce projet.

Nous vous remercions d'avoir pris le temps de lire notre rapport.