



Université de Picardie Jules Verne
UFR des sciences
Licence 3 informatique – parcours MIAGE

RAPPORT FINAL DE JEU DE SERPENT

Réalisé par :

Saleh Amari
Samy Harfouche

Encadré par :

Soufia Bennai

Année universitaire : 2021/2022

SOMMAIRE

SOMMAIRE	1
I. Introduction :	2
Concept :	2
Exemple du jeu :	2
PARTIE 1 : CAHIER DES CHARGES.....	4
Besoins et objectifs :	4
1- Moyens et contraintes :a- Moyens humains :	4
b- Moyens financiers :	4
c- Moyens matériels :	4
2- Contraintes techniques et contraintes d'architecture :	4
3- Contraintes de coût, de qualité et de délais :	5
ANALYSE DE L'EXISTANT, CONCURRENCE ET POSITIONNEMENT :	5
1- Jeu Snake multi-joueurs : ²	5
2- Jeu de Snake 3D : ³	5
Offre fonctionnelle de votre produit.....	6
A- DESCRIPTION DE CHACUNE DES FONCTIONNALITÉS : ⁴	6
Scenario d'usage :	13
Partie 2 : Ébauche de planification - Décomposition en tâches et jalons :	14
A. Partie obligatoire :	14
1. Liste des tâches :	14
2. Diagramme de Gantt :	15
3. Un suivi du planning à l'aide de Trello :	15
PARTIE 3 : Analyse détaillée.....	16
PARTIE 4 : Conception détaillée	22
PARTIE 5 : Codage.....	27
PARTIE 6 : Bilan	40
Conclusion	42
REFERENCES :	43

I. Introduction :

Dans le cadre de notre troisième année licence informatique nous avons eu pour tâche la réalisation d'un projet informatique. Notre objectif était de programmer d'un Jeu de serpent jouable au clavier dans un page web en utilisant les langages : Javascript, HTML-5, CSS-3.

Dans ce dossier nous allons tout d'abord exposer l'analyse de ce jeu, puis un manuel d'utilisateur et enfin Un manuel du programmeur.

D'abord, qu'est-ce qu'un Jeu du serpent ? ¹

Le jeu de serpent ou bien Snake Game, est un genre de jeu vidéo dans lequel le joueur dirige un serpent qui grandit et constitue ainsi lui-même un obstacle. Bien que le concept tire son origine du jeu vidéo d'arcade Blockade, il n'existe pas de version standard. Son concept simple l'a amené à être porté sur l'ensemble des plates-formes de jeu existant sous des noms de clone.

Le jeu a connu un regain de popularité à partir de 1998 quand Nokia, une entreprise de télécommunication, l'a intégré dans ses produits. Avec l'émergence du nouveau support de jeu qu'est le téléphone mobile, il est devenu un classique du jeu sur appareil mobile.

Concept :

Le principe du jeu est de diriger un serpent vers des boules dont il doit se nourrir pour grandir. Le défi est faire grandir la taille du serpent sans toucher les bordures de l'écran ou sa propre queue. Les boules doivent s'afficher aléatoirement, à la consommation d'une boule, une nouvelle boule à atteindre sera générée de façon aléatoire. Un joueur peut reprendre une partie perdante en dépensant des points qu'il a cumulés durant les parties précédentes. Le jeu peut se jouer seul ou à deux. Si un des joueurs touche l'autre il perd la partie.

Exemple du jeu :



Cahier des charges



PARTIE 1 : CAHIER DES CHARGES

Besoins et objectifs :

L'Université Picardie Jules Verne est un établissement d'enseignement public à caractère scientifique, culturel et professionnel situé à Amiens au nord de la France. Elle développe des formations professionnalisantes désormais pleinement reconnues sur le marché du travail. Elle est constituée d'organes de gouvernance, de composantes (UFR, instituts, ESPE), écoles doctorales, de directions et de services administratifs.

Dans l'une de ses composantes « UFR des sciences » est dispensée une unité d'enseignement appelée Projet dont Mme BENNAI SOUFIA nous encadre dans un projet qui consiste à créer un jeu de serpent en utilisant Javascript, HTML5 et CSS3 dans un délai de deux mois

1- Moyens et contraintes :

a- Moyens humains :

Cette partie il s'agit de l'ensemble des acteurs qui interviennent sur le projet. Les ressources humaines sont fondamentales à tous projet car ce sont elles qui accomplissent l'ensemble des tâches. Dans le cas de notre projet, nous avons décidé de se partager les tâches selon les rôles suivants :

- Membre 1 : Chef de projet + développeur
- Membre 2 : Testeur + développeur

b- Moyens financiers :

Les ressources financières englobent tous les coûts et les dépenses engendrés par le projet. Pour notre projet, nous ne prenons en considération seulement la durée. Le temps possible qui va nous prendre et que nous aurons besoin par rapport au délai est presque 2 mois.

c- Moyens matériels :

Les ressources matérielles englobent tout ce qui sera nécessaire à la réalisation d'un projet (matériels, équipements, logiciels, outils...). Pour notre cas, le matériel nécessaire et utilisé lors de la réalisation de ce projet :

Lenovo Yoga

✓ Système d'exploitation : Windows 10.

✓ CPU : Core i5

✓ Mémoire : 8Go

Toshiba

✓ Système d'exploitation : Linux Ubuntu 20

✓ CPU : Core i7

✓ Mémoire : 16Go

2- Contraintes techniques et contraintes d'architecture :

Parmi les contraintes techniques qu'on doit respecter :

- Elle devra fonctionner en full web et être accessible par une URL.
- Les outils de programmation doivent être : Javascript, HTML5, CSS3.
- L'université « le service DISIP » dispose de serveurs Linux qui sera utilisé par la solution.
- Présenter les informations d'une façon simple et claire
- Le système doit prendre en considération aussi le pouvoir des utilisateurs de se familiariser rapidement avec les différentes interfaces de jeu.
- Des interfaces simples et compréhensibles
- La disposition des boutons d'une manière logique et l'organisation des rubriques.
- La maintenance continue. On sait que chaque cycle de vie d'un projet ou d'une application nécessite d'assurer la maintenance « Par DISIP »

3- Contraintes de coût, de qualité et de délais :

La qualité, le coût et le délai sont les 3 facteurs qui vont influencer un projet, et ces facteurs forment ce qu'on appelle un triangle d'or. Pour mettre en place un triangle d'or performant, il est essentiel d'utiliser une méthode de gestion de projet qui a fait ses preuves.

Dans le cas de notre projet nous allons utiliser la méthode agile qui va nous permettre d'optimiser nos délais, nos coûts ainsi que la qualité du produit le maximum possible. Ainsi que l'utilisation d'un logiciel de gestion de projet et de planification pour découper le projet en sous-tâches (GANTT OU PERT).

ANALYSE DE L'EXISTANT, CONCURRENCE ET POSITIONNEMENT :

Actuellement, il existe plein de types de jeu de serpent programmer par différent langages de programmation, ces jeux contiennent des fonctionnalités communes comme (consommer les bulles, afficher le score ...) et chaque jeu contient ses propres fonctionnalités. Dans notre analyse on a pris deux exemples :

1- Jeu Snake multi-joueurs :²

Le **jeu multijoueur Snake** vous invite à retrouver le fabuleux **jeu du serpent**, un grand classique des [jeux d'arcade](#) redécouvert sur les mobiles en 1990. Le plus de ce **jeu en ligne** c'est de jouer à plusieurs !

Dans ce **jeu multijoueur**, c'est la bagarre entre ces deux **serpents** : d'abord pour **manger plus de fruits** et autres bonus que l'autre serpent, ensuite pour **se débarrasser définitivement de l'adversaire**.

Si vous vous sentez fort et rusé comme votre serpent, affrontez vite d'autres joueurs : **celui qui remporte la partie est le premier à gagner trois rounds**. Pour vous déplacer, inutile de vous tortiller sur votre fauteuil, utilisez juste les touches directionnelles de votre clavier. Mais si vous avez envie de vous tortiller dans tous les sens, vous pouvez. Chacun sa technique pour gagner.

Nombre de joueurs : 2

Comment jouer ?  Diriger le serpent

2- Jeu de Snake 3D :³

Si vous aimez les **jeux de serpent**, vous allez pouvoir essayer de vous surpasser avec le **jeu en ligne 3D Snake** dans deux décors différents.

Le but du jeu est de **récolter un maximum de pommes** sans sortir du décor. Vous devrez également faire attention de ne pas vous rentrer dedans, cela vous fera également perdre la partie. La difficulté va rapidement augmenter car votre serpent va grandir de plus en plus et occuper toujours davantage d'espace.

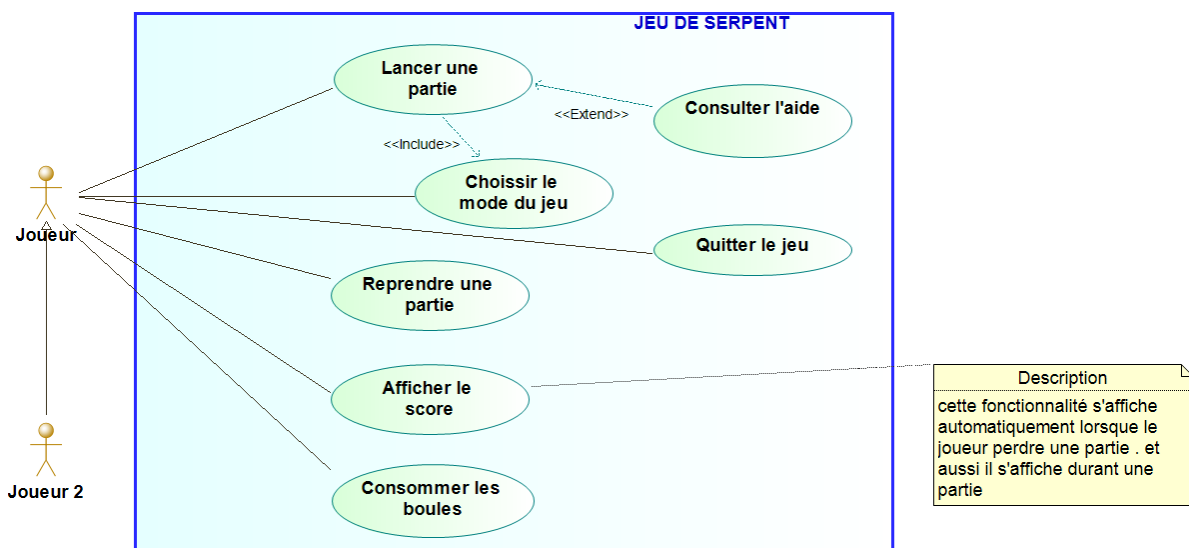
Nombre de joueurs : 1

Alors pour améliorer les fonctionnalités de jeu nous avons proposé une version plus améliorée qui sera une interface html et programmé par Javascript

Dans notre projet nous nous proposons des nouvelles fonctionnalités tel que :

- Reprendre une partie
- Choisir le mode du jeu (Joueur solo – ou multijoueur)
- Quitter le jeu
- ...

Offre fonctionnelle de votre produit



Nous aurions deux acteurs « Joueur humain » et « Joueurs IA », qui vont interagir avec notre futur jeu comme le montre le diagramme ci-dessus.

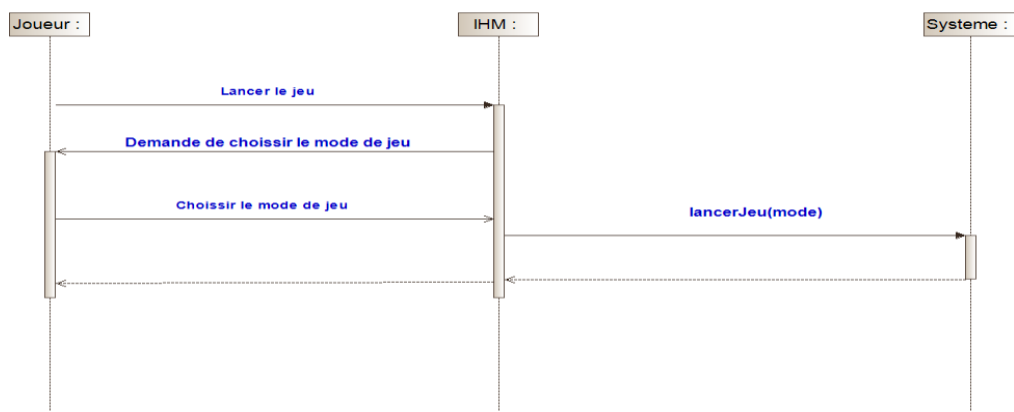
A- DESCRIPTION DE CHACUNE DES FONCTIONNALITÉS :⁴

Dans cette partie, nous allons décrire les différentes fonctionnalités attendues par le client du futur produit et de les hiérarchiser d'une manière rigoureuse.

Nom de la fonctionnalité : Lancer une partie

Utilisateurs : Joueur

Description fonctionnalité



Règles de gestion

- Entrée : il faut saisir un URL
- Sortie : Dans le cas où saisie l'url correct, le joueur accède au jeu. Sinon, un message d'erreur se déclenche (Peut être erreur 404)

Autres contraintes

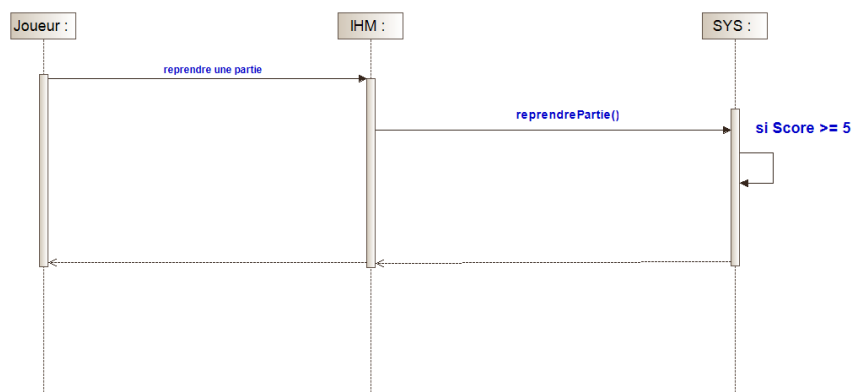
Priorité

Must : Lancer le jeu est obligatoire pour accéder au jeu

Nom de la fonctionnalité : reprendre une partie

Utilisateurs : Joueur

Description fonctionnalité



Règles de gestion

- Entrée : cliqué sur le bouton continue
- Sortie : si le joueur a bien collecté 5 points ou plus de score il peut continuer le jeu sinon il recommence à zéro

Autres contraintes : le joueur doit cumulée au moins 5 points de score

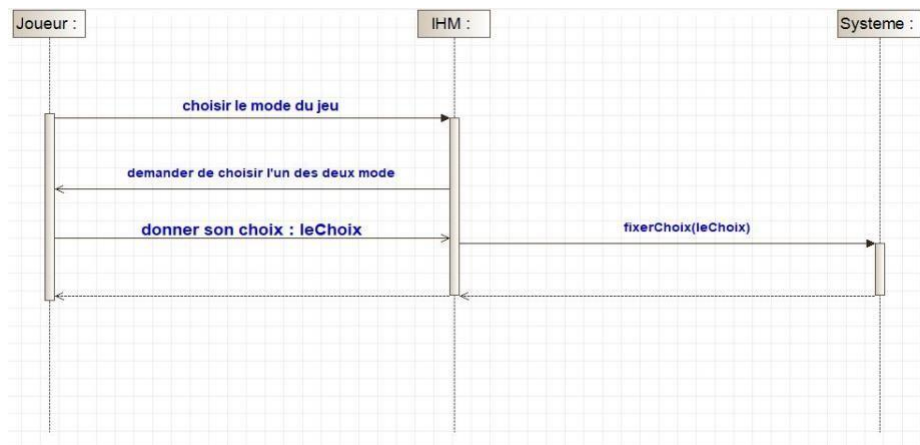
Priorité

Must : Lancer le jeu est obligatoire pour accéder au jeu

Nom de la fonctionnalité : Choisir le mode du jeu

Utilisateurs : Joueur

Description fonctionnalité



Règles de gestion

- Entrée : il faut choisir entre les deux modes « solo » ou « multijoueur »
- Sortie : après le choix le jeu lance avec le mode demandé

Autres contraintes

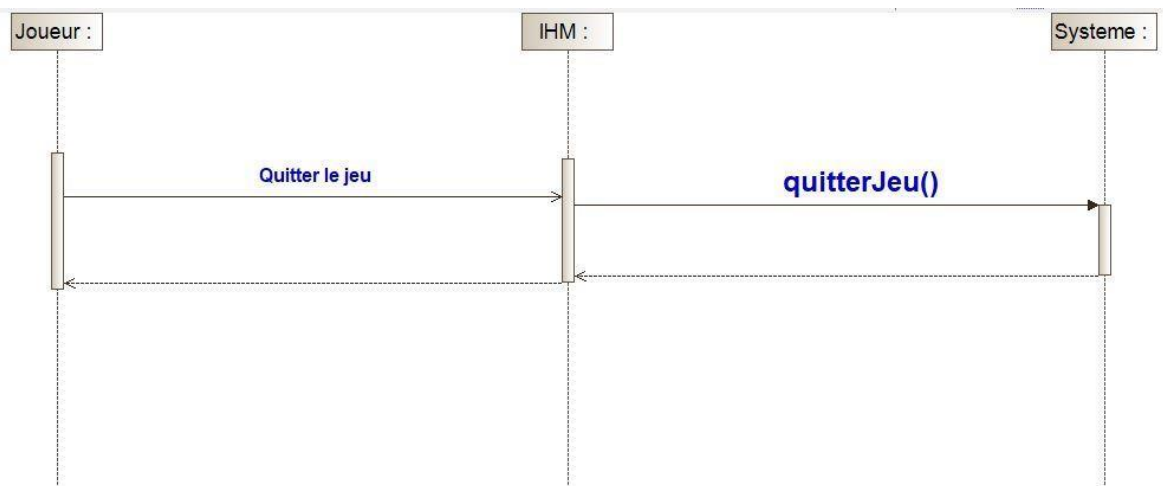
Priorité

Must : cette fonctionnalité est obligatoire avant de lancer le jeu

Nom de la fonctionnalité : Quitter le jeu

Utilisateurs : Joueur

Description fonctionnalité



Règles de gestion

- Entrée : le joueur peut fermer le jeu en tout moment en cliquant sur un bouton
- Sortie : un message pour savoir si le joueur est sûr de son choix

Autres contraintes

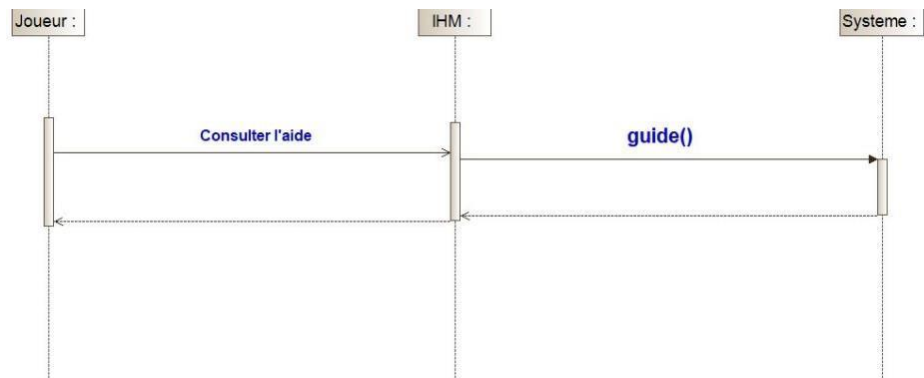
Priorité

Must : cette fonctionnalité est obligatoire pour fermer le jeu

Nom de la fonctionnalité : consulter l'aide

Utilisateurs : Joueur

Description fonctionnalité



Règles de gestion

- Entrée : le joueur peut consulter le guide le jeu en tout moment en cliquant sur un bouton
- Sortie : un texte montrant comment jouer

Autres contraintes

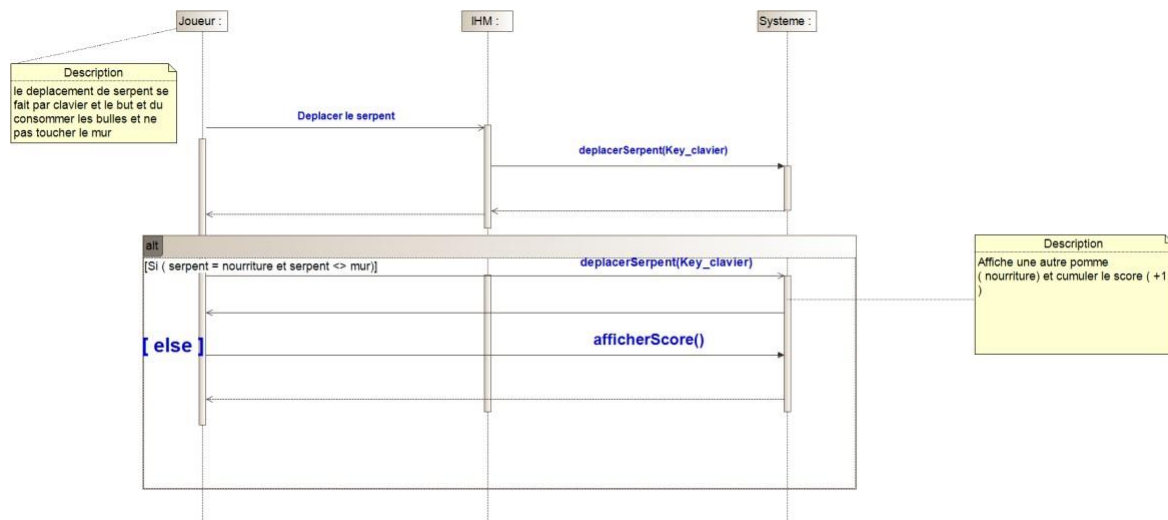
Priorité

Could : cette fonctionnalité est optionnelle

Nom de la fonctionnalité : consommer les bulles

Utilisateurs : Joueur, Joueur IA

Description fonctionnalité



Règles de gestion

- Entrée le joueur doit faire le serpent déplacer en utilisant le clavier, et il doit consommer des bulles sans toucher sa queue et le mur
- Sortie : le score se cumule et la nourriture apparaitre aléatoirement, sinon l'une des règles ci-dessus n'ont pas été respectées, le Game is over et le score finale s'affiche

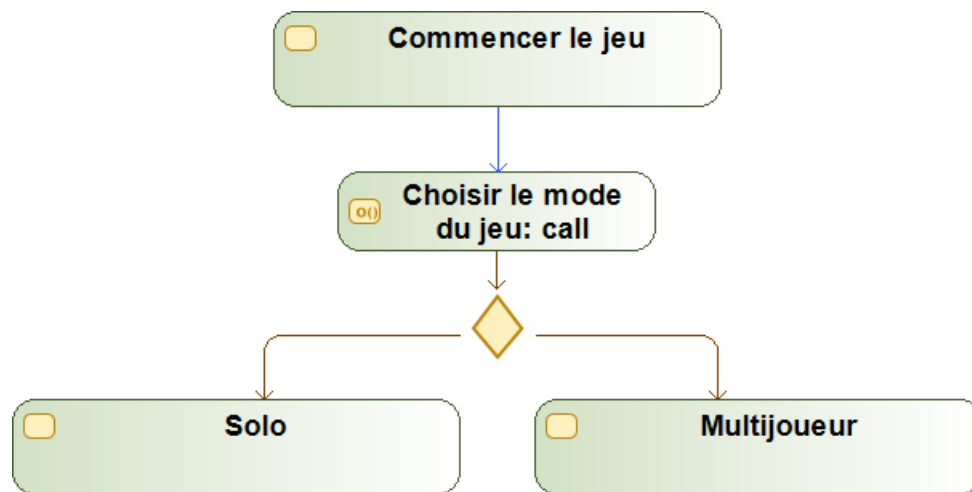
Autres contraintes

Priorité

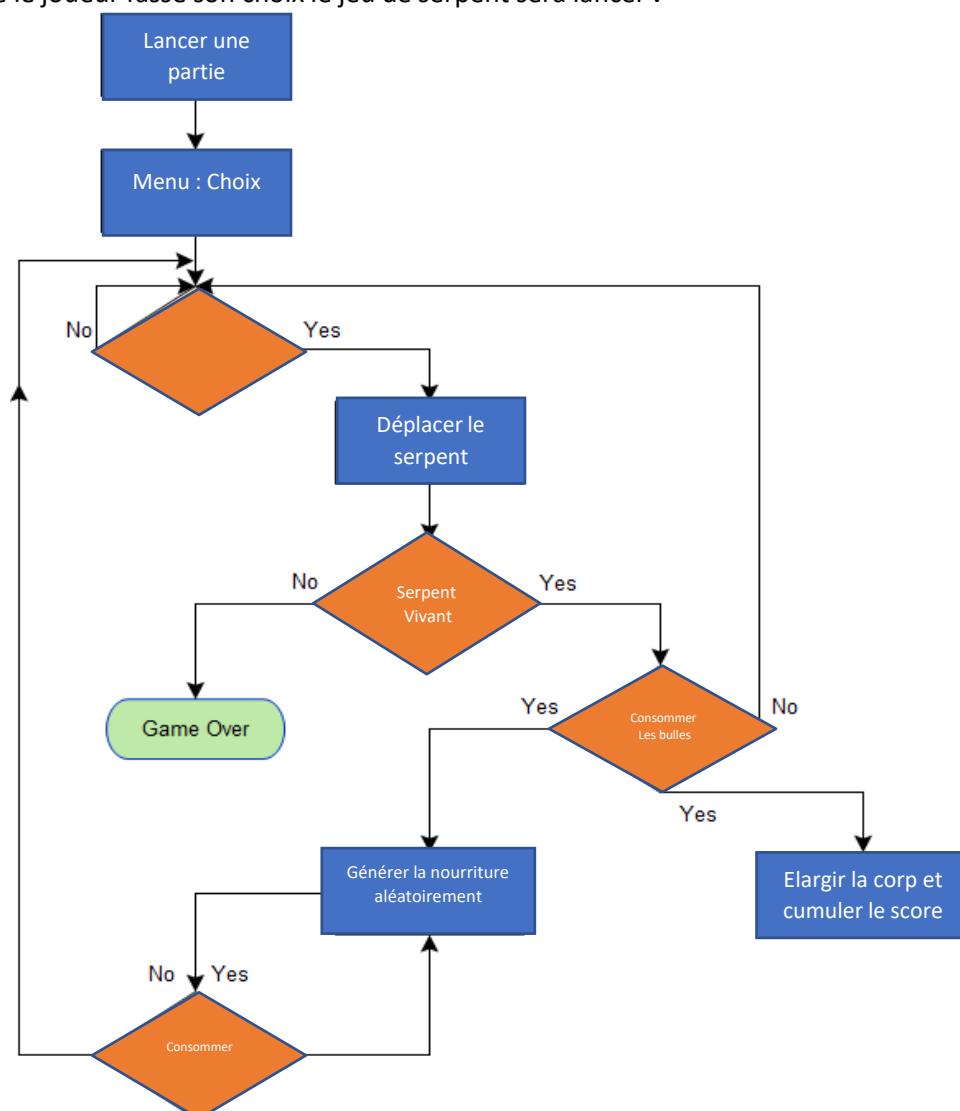
Must : Must : cette fonctionnalité est obligatoire, c'est le cœur de jeu

Scenario d'usage :

Lorsque le joueur lance le jeu un menu s'affiche :



Après que le joueur fasse son choix le jeu de serpent sera lancer :



Partie 2 : Ébauche de planification - Décomposition en tâches et jalons :

Afin d'organiser le déroulement des étapes de notre projet dans le temps. La planification est une phase indispensable pour atteindre l'objectif fixé et une tâche fondamentale pour la maîtrise des délais.

En suivant chacune de phases nous aurons la garanti que notre projet sera une réussite et nous éviterons les possibles malentendus et problèmes.

A. Partie obligatoire :

La date de livraison du projet final est déterminée pour le 08/04/2022.

Pour cela, la première chose que nous avons faite est de déterminer la date de livraison finale, ensuite nous avons subdivisé la livraison en plusieurs parties et tout au long de la durée du projet.

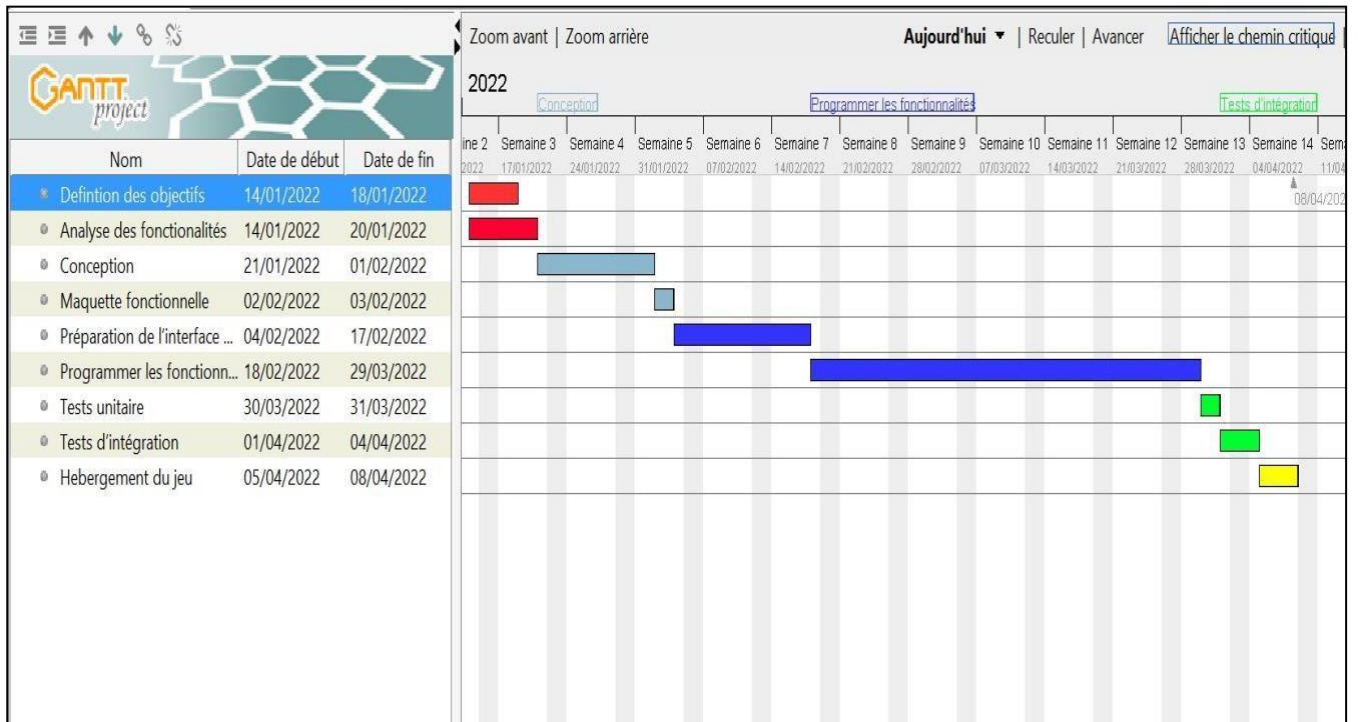
1. Liste des tâches :

Numéro	Description de la tâche	Durée allouée	Tâches antérieures
Cahier de charge :			
1	Définition et objectif	2 jours	/
2	Analyse fonctionnelle	5 jours	/
Analyse vers conception :			
3	Conception	8 jours	2
4	Maquette fonctionnelle	2 jours	3
Développement du Jeu :			
5	Préparation de l'interface HTML, CSS	10 jours	4
6	Programmer les fonctionnalités	25 jours	5
Faire des tests :			
7	Tests unitaire	2 jours	6
8	Tests d'intégration	2 jours	7
Mise en production :			
9	Hébergement de jeu	3 jours	8

Nous savons que c'est compliqué de connaître l'avenir. Néanmoins, il y a des choses qui peuvent se prévoir à l'avance, comme les jours fériés et les périodes de vacances par exemple.

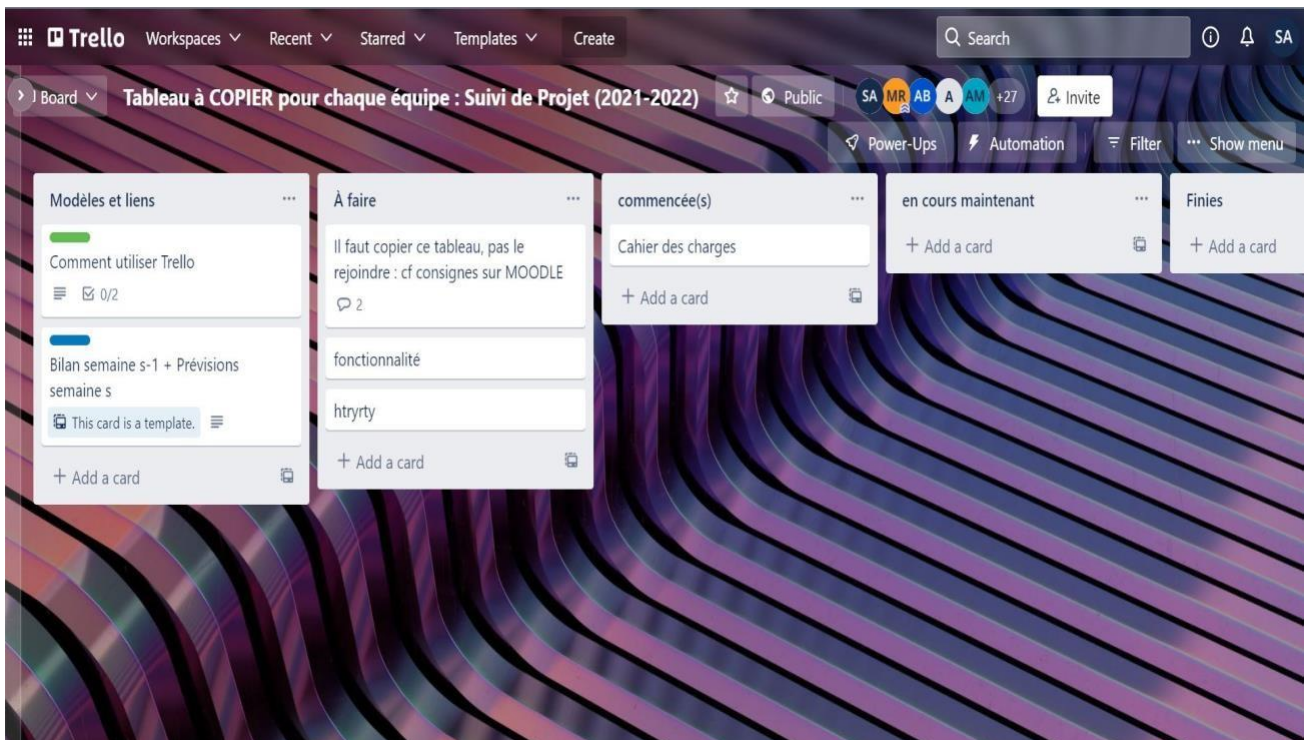
Pour cela nous avons pris en compte certains jours fériés qui tombent pendant la réalisation de notre projet et qu'il est important de les prendre en compte, sinon quoi il pourrait y avoir certains retards dans la planification du projet.

2. Diagramme de Gantt :



3. Un suivi du planning à l'aide de Trello :

Pour ne pas tomber dans un écueil en phase de planification, nous allons utiliser au quotidien un tableau De bord projet (Trello) pour suivre l'avancement de notre projet



PARTIE 3 : Analyse détaillée



1- Choix de la solution :

Afin d'offrir un jeu de serpent innovant aux joueurs, le jeu devra être constitué de plusieurs fonctions tel que :

- Lancer le jeu
- Consulter le guide
- Reprendre une partie (en consommant le score)
- Quitter le jeu
- Choisir le mode de jeu

1-1 Les langages de programmation Web :

En respectant nos contraintes techniques et contraintes d'architecture nous avons pu choisir notre pile technologique :

Composants	Fonction
HTML 	HTML indique à un navigateur comment afficher le contenu des pages Web Gratuit
CSS 	CSS formate le contenu (couleur, taille de police). Gratuit
JAVASCRIPT 	JS rend les pages Web interactives. Il existe de nombreuses Framework (tels que Angular, Vue, Back Bône et Ember) pour un développement Web plus rapide et plus facile Gratuit
Bootstrap 	Bootstrap est une collection d'outils utiles à la création du design de sites et d'applications web. C'est un ensemble qui contient des codes HTML et CSS, des formulaires, boutons, outils de navigation et autres éléments interactifs, ainsi que des extensions JavaScript en option. Gratuit

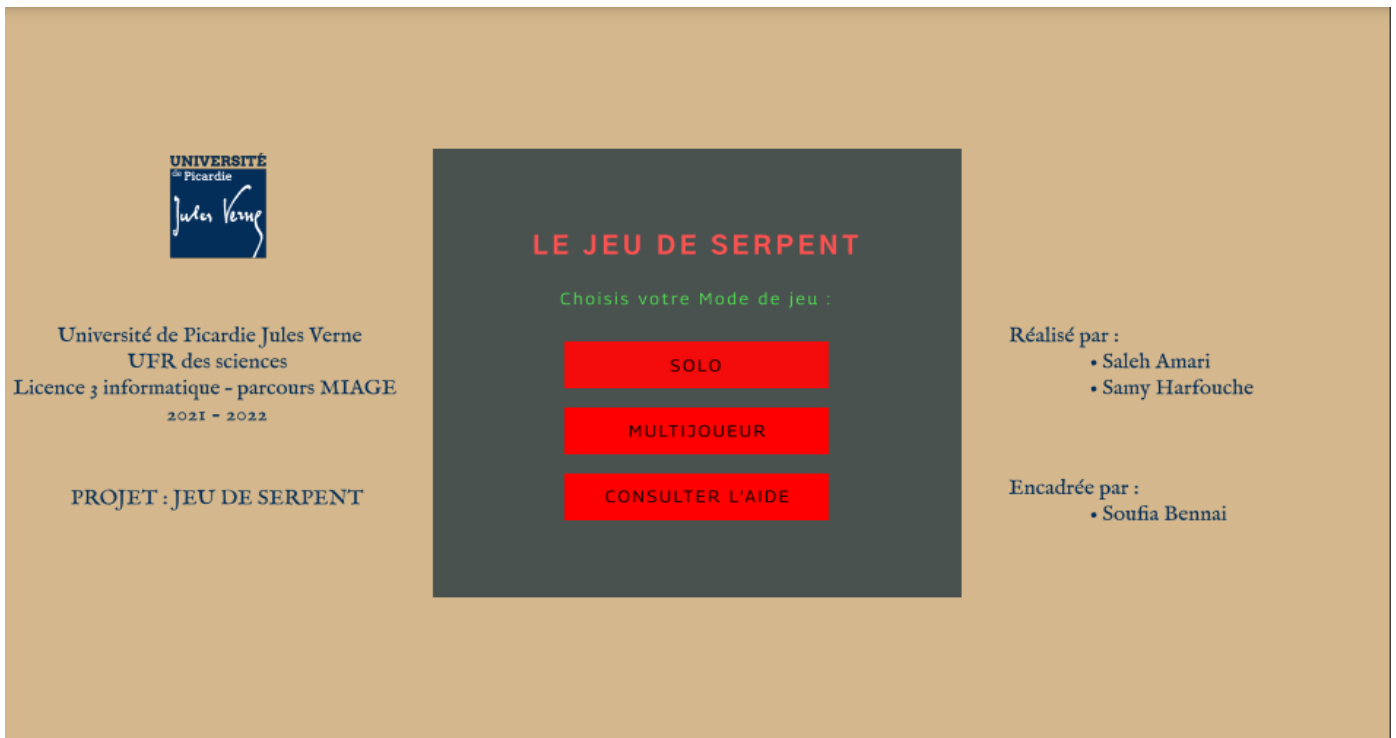
2- Maquette fonctionnelle :

1-2. Charte graphique :

- **Ecritures** (blanc) : Calibri – Georgia #f5f5f5
- **boutons** (**rouge** – **vert** - **jaune**) : #f32013 - #046104 - #ffc107
- **Fond d'écran de la page** : (IMAGE) bp.jpg

2-2. Interfaces ⁵:

- 1- La première page affichée lors de lancement du jeu est la page qui permet de choisir le mode de jeu :



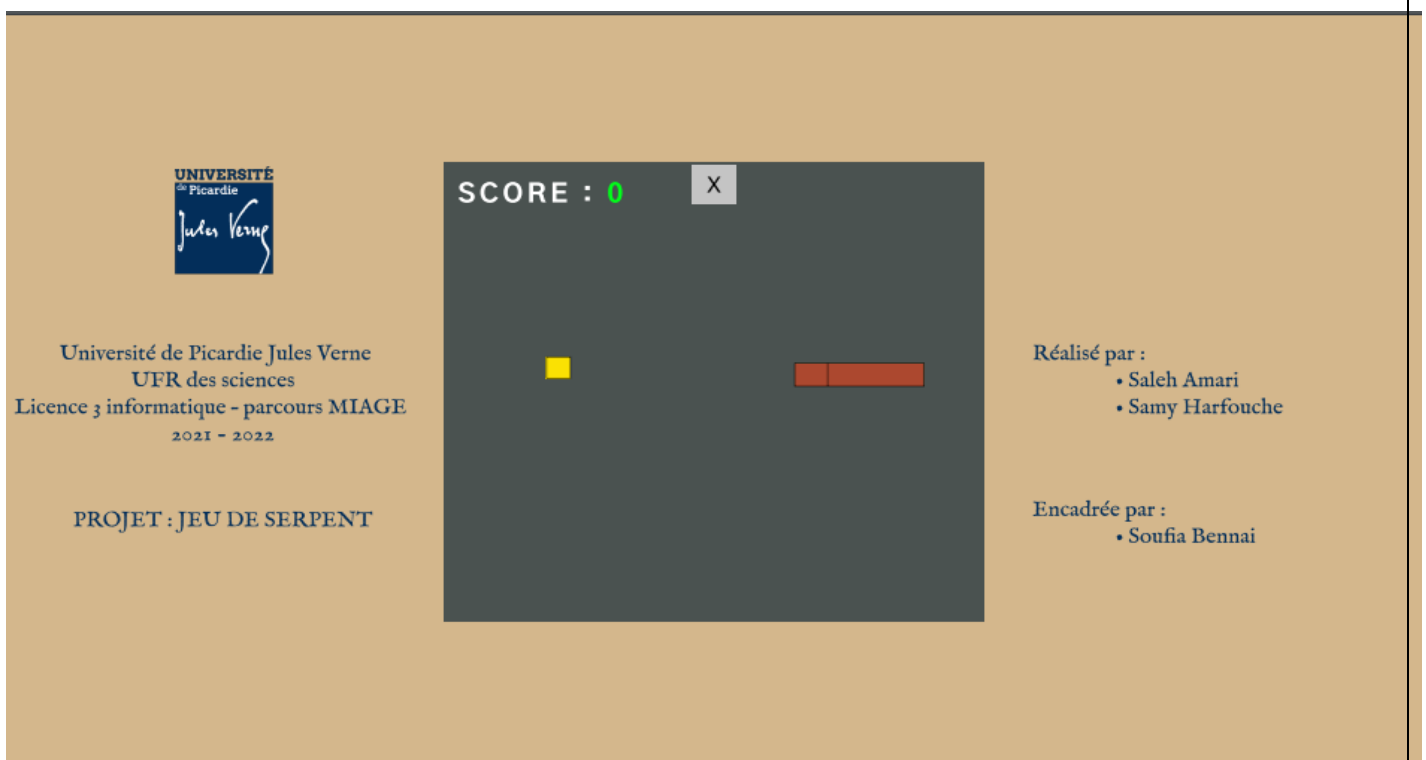
Après le lancement de jeu, le joueur a le choix entre jouer une partie seule ou bien en deux. Il a aussi la possibilité de consulter le guide de jeu pour comprendre le principe du jeu.

2- Lorsque le joueur clique sur Consulter l'aide cette interface s'affiche :



Ci-dessus les instructions pour joueur au jeu on y retrouve les boutons du clavier qu'il faut manipuler pour faire bouger le serpent.

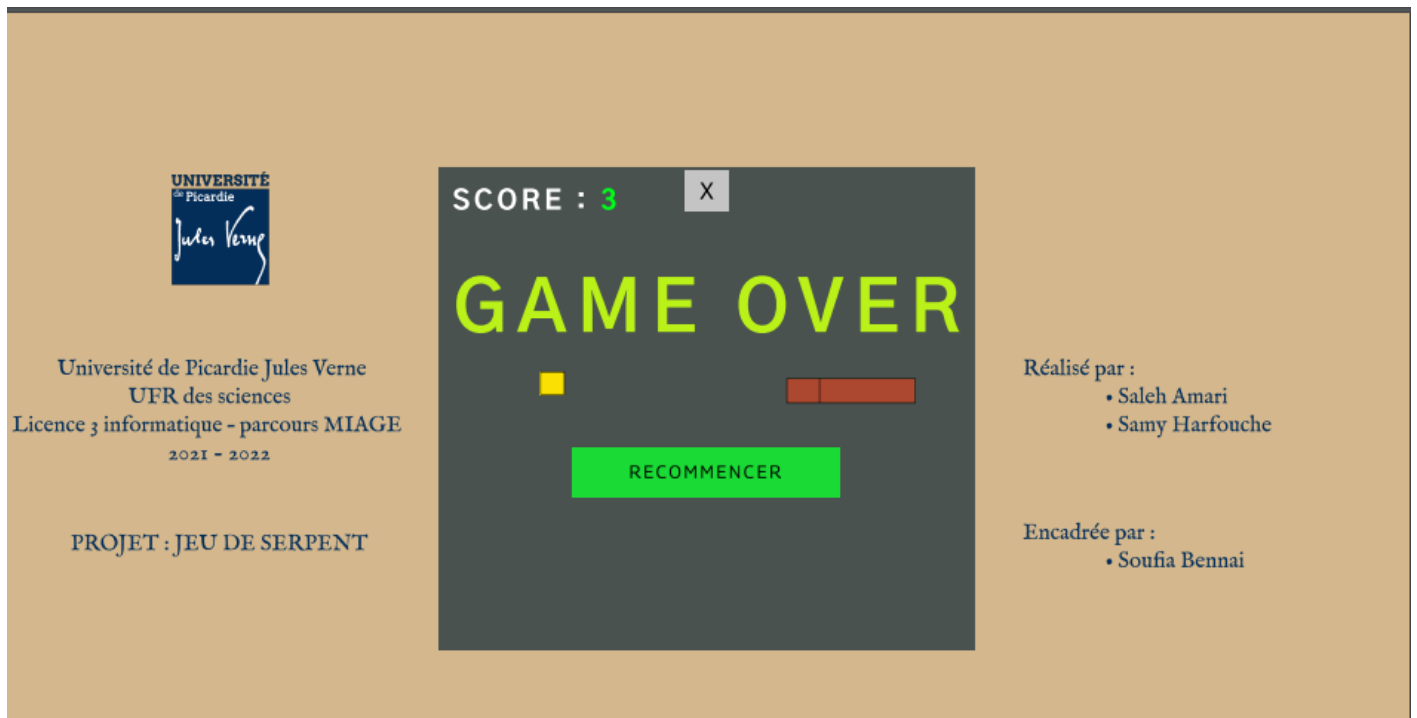
3- Une partie Solo :



Lorsque le joueur choisit le mode solo cette interface s'affiche, il doit utiliser les flèches du clavier pour faire bouger le serpent.

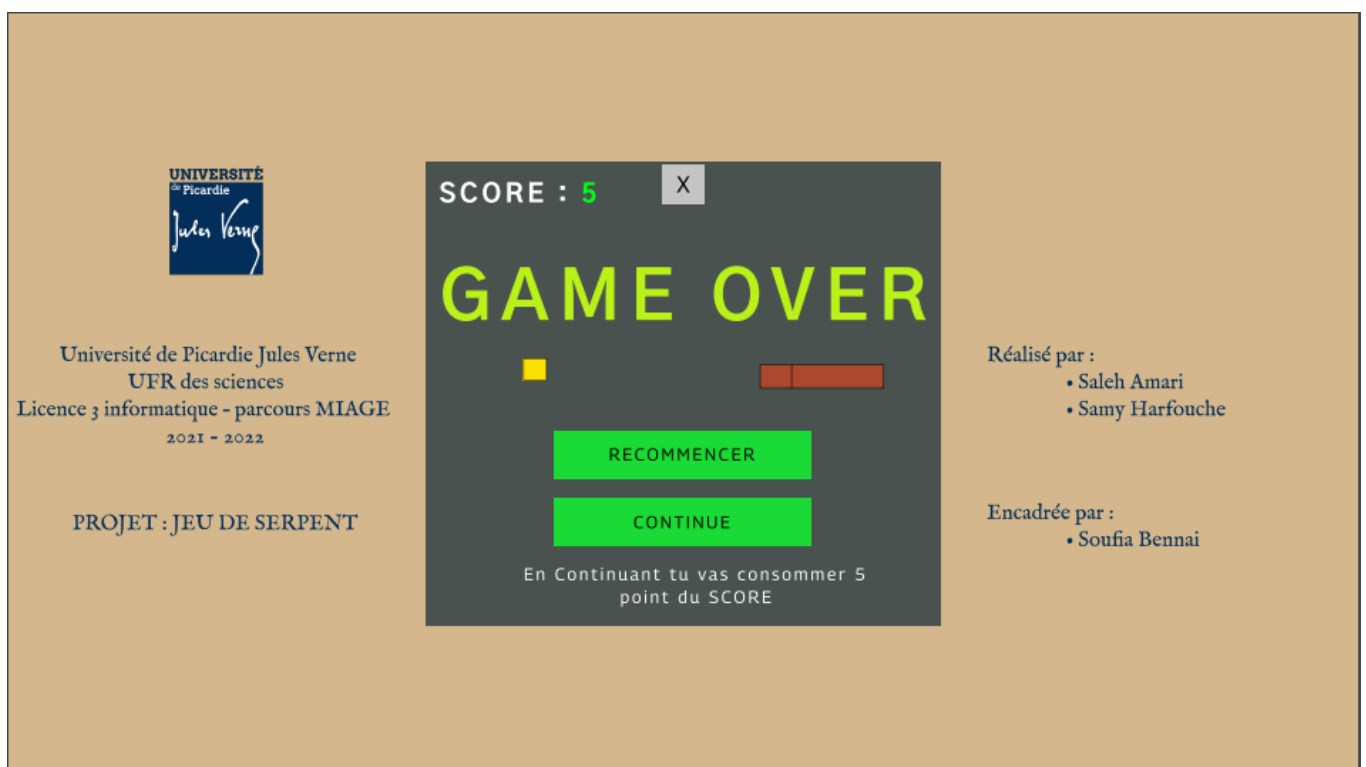
4- GAME OVER S1 :

Quand le serpent touche le mur ou bien une partie de son propre corps et qui n'a pas encore cumulé 5 points du score, Il perd la partie.



5- GAME OVER S2 :

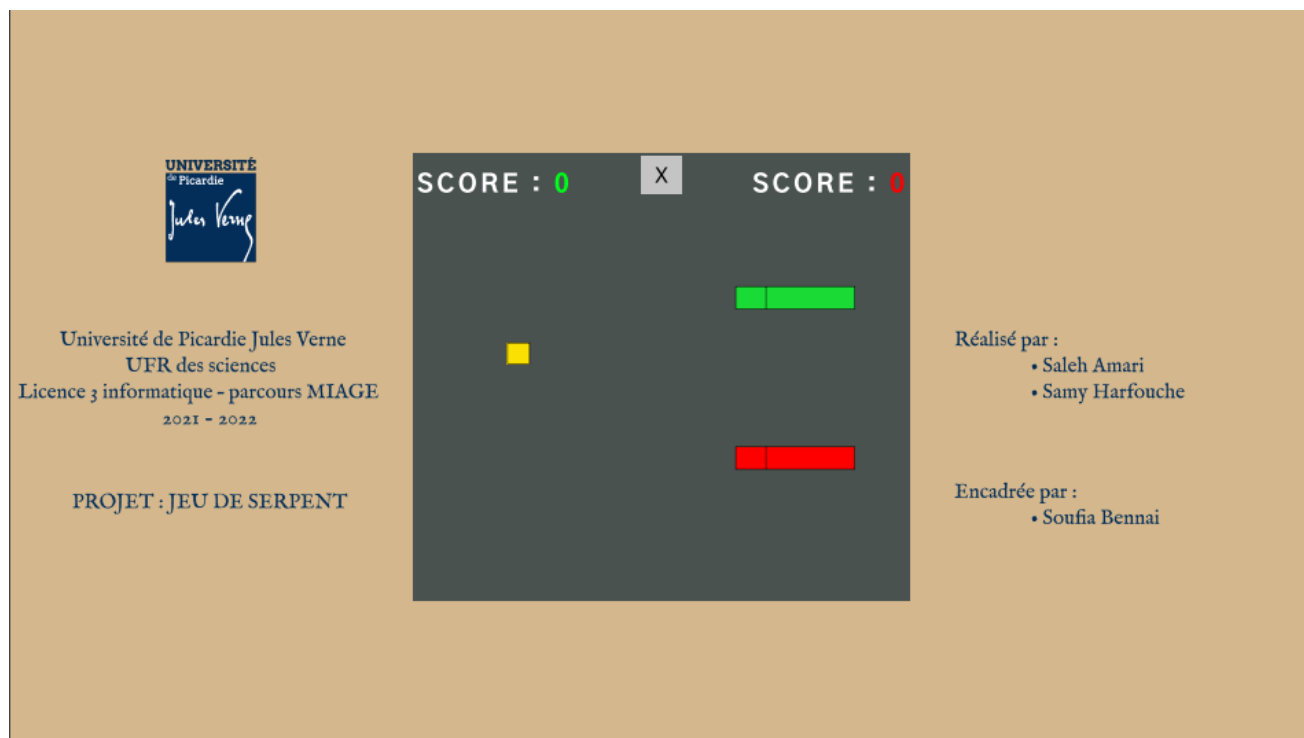
Quand le serpent touche le mur ou bien une partie de son propre corps et qui a déjà cumulé 5 points du score, Il a la possibilité de dépenser ses 5 point et continue à jouer.



6- Une partie Multijoueur :

Lorsque le joueur choisit le mode Multijoueur cette interface s'affiche, le 1er joueur doit utiliser les flèches du clavier pour faire bouger le serpent. Et le deuxième doit utiliser les touches suivantes :

Z : en haut — S : en bas — D : à droite — Q : à gauche



7- GAMEOVER M :

Quand l'un des serpents touche le mur, une partie de son propre corps ou bien l'un touche l'autre, la partie est finie et le score final des deux s'affiche.

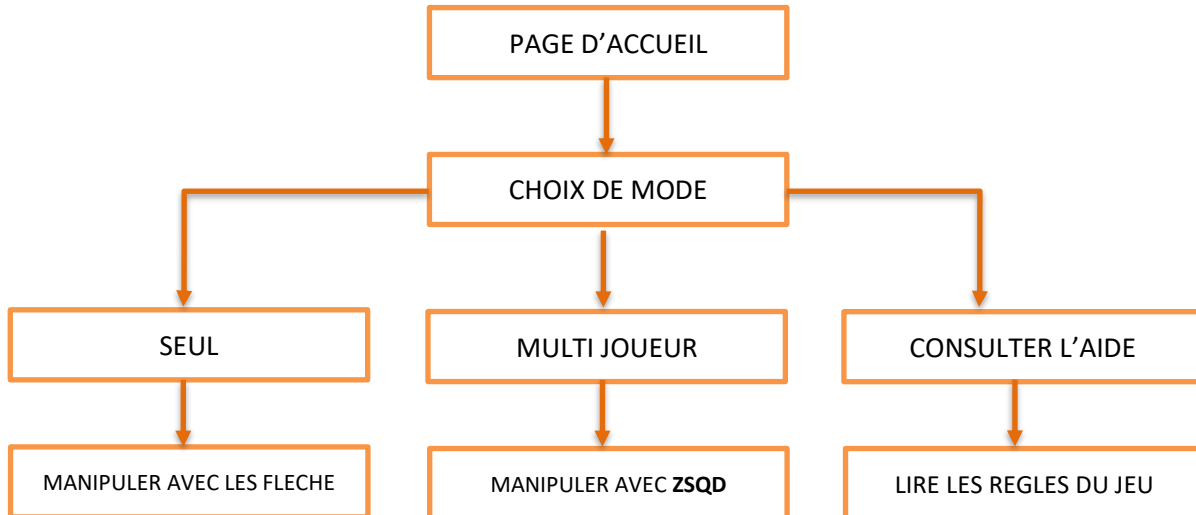


PARTIE 4 : Conception détaillée



Dans la phase de conception, nous devons répondre à la question "COMMENT" notre jeu fonctionne Et comment est réalisée chaque fonctionnalité.

Ci-dessus on a dessiné une représentation graphique de l'architecture de jeu :



1- Les tests fonctionnels : ⁶



Les **tests fonctionnels** sont destinés à **s'assurer** que, dans le contexte d'utilisation réelle, le **comportement fonctionnel** obtenu est bien **conforme** avec celui attendu.

Un test fonctionnel a donc pour objectif de **dérouler un scénario** composé d'une liste d'actions, et pour chaque action d'**effectuer** une liste de **vérifications** validant la conformité de l'exigence avec l'attendu




Il existe deux types de test fonctionnel.

Le **test fonctionnel de progression**, réalisé sur une exigence qui vient d'être développée. C'est donc la **première fois** que ce test est réalisé, avant que le produit ne soit livré en production.

Le **test fonctionnel de non-régression**, réalisé sur une exigence existant déjà avant les développements en cours. Il permet de **s'assurer** que les **évolutions** et les **corrections** effectuées dans une nouvelle version n'ont pas entraîné de **régressions** sur les **fonctionnalités existantes**. Au cours d'un test fonctionnel, si une action échoue ou qu'une vérification n'est pas satisfaite, on est alors en présence d'un signe de non-qualité dans le système, plus communément appelé bug.

ACTEUR	ACTION	RESULTAT	VERIFICATION
PAGE D'ACCUEIL			
En tant que joueur je veux accéder au jeu	Saisir l'url (Cas d'hébergement) Clique sur INDEX.HTML (cas échéant)	Affichage de la page d'accueil (MODE DE JEU)	✓
LE CONCEPT DE JEU			
En tant que joueur je veux me renseigner sur les règles du jeu	Je clique sur Consulter L'aide	Une page qui affiche le principe de jeu Bouton pour rédiger à la page de mode du jeu	✓
PARTIE SOLO			
En tant que joueur je veux lancer une partie solo	Je clique sur le bouton SOLO	Affichage de d'un seul serpent sur le Canvas	✓
Diriger le serpent vers la nourriture		Le serpent se dirige par des flèches	✓
Dans le cas où le serpent touche le mur ou sa propre queue (sachant que le score est moins de 5)	Diriger le serpent par des flèches vers le mur, vers sa queue	Affichage de Game over et le score finale Bouton Recommencer (à zéro) Quitter la partie et rédiger à la page de mode du jeu	✓
Dans le cas où le serpent touche le mur ou sa propre queue (sachant que le score est \geq de 5)	Diriger le serpent par des flèches vers le mur, vers sa queue	Affichage de Game over et le score finale Bouton Recommencer (a zéro) Bouton Continue (consommer 5 points du score et garder la même taille) Quitter la partie et rédiger à la page de mode du jeu	✓
Quitter le jeu en tout moment	Cliquer sur 	Quitter la partie et rédiger à la page de mode du jeu	✓

PARTIE MULTIJOUEUR

En tant que joueur je veux lancer une partie Multijoueur	Je clique sur le bouton Multijoueur	Affichage de deux serpents sur le Canvas	✓
Diriger le serpent vers la nourriture	<p>1^{er} serpent</p>  <p>2eme serpent</p> 	<p>Les serpents se dirigent par des flèches et par ZSQD</p> <p>Z= en haut</p> <p>S= en bas</p> <p>D= a droite</p> <p>Q= a gauche</p>	✓
Dans le cas où le serpent touche le mur, sa propre queue ou bien l'un des serpents touche l'autre	Diriger le serpent par des flèches ou ZSQD vers le mur, sa queue ou bien vers l'autre serpent	<p>Affichage de Game over</p> <p>Bouton Recommencer (à zéro)</p> <p>Affichage de score finale X-Y</p> <p>Quitter la partie et rédiger à la page de mode du jeu</p>	✓
Quitter le jeu en tout moment	<p>Cliquer sur</p> 	Quitter la partie et rédiger à la page de mode du jeu	✓
<h2 style="text-align: center;">La nourriture et le score</h2>			
Affichage de nourriture aléatoirement	Consommer la première boule	En fois le serpent mange une boule, une autre boule s'affiche automatiquement et aléatoirement	✓
Le score se cumule par 1	Consommer la nourriture	Le score se cumule par 1	✓

PARTIE 5 : Codage



1- Le squelette HTML

Créons un fichier html appelé Index.html dans lequel nous mettrons le contenu d'une page vide, et on l'a lié à une feuille de style CSS et a Bootstrap.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="style.css">
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
  <title>Serpent</title>
</head>
```

Figure 1 : Lier HTML à CSS ET BS

```

57
58 <script src="script.js"></script>
59 </body>
60 </html>

<body>
<div class="container-fluid">
<h3 class="disabled" id="red_serp">
<h3 class="disabled" id="red_fine">
<h1 id="title" class="text-danger">

<button id="start-btn" class="btn-danger rounded">SOLO JEU</button><br>
<button id="restart-btn" class="disabled btn-success rounded">Recommencer</button>
<button id="restart-2-btn" class="disabled btn-success rounded">Recommencer</button>
<button id="quitter-btn" class="disabled btn-danger rounded">Quitter la partie</button>
<button id="x-btn" class="disabled btn-danger rounded">X</button>

<canvas id="game-canvas"></canvas>

<button id="continue-btn" class="disabled btn-success rounded">Continue</button><br>

<p id="instruction" class="disabled text-muted"><b>Remarque :</b> <br>En continuant tu vas consommer 5 point du score </p>
<p id="gameover" class="disabled text-warning"> GAME OVER </p>

<button id="multi-start-btn" class="btn-danger rounded">MULTI-JOUEUR</button>
<button id="guide-btn" class="btn-success rounded">CONSULTER L'AIDE</button>
<button id="back-btn" class="disabled btn-warning rounded">CHOISIS TON MODE</button>

<h3 class="disabled" id="green_serpent">Score : <span id="score2">0</span></h3>
<h3 class="disabled" id="both-final-score">Vos Score finale sont : <span id="score-green">0</span> - <span id="score-red">0</span></h3>

<p id="univirsite" class="text-secondary">Université de Picardie Jules Verne<br>UFR des sciences<br>Licence 3 informatique - parcours MIAGE <br> 2021 - 2022<br><br> <span id="projet" class="text-dark">PRO.
<div id="realisateur" class="text-secondary">
<b>Réaliser par :</b>
<ul>
<li>Saleh Amari</li>
<li>Samy Harfouche</li>
</ul>
<br>
<b>Encadrée par :</b>
<ul>
<li>Soufia Bennai</li>
</ul>
</div>
<div id="logo">

</div>
</div>

```

Figure 2: notre page HTML

Figure 3: lier notre HTML à JAVASCRIPT

2- Notre Zone de jeux, Canvas et contexte

La zone de jeux sera un Canvas auquel nous lui donneront un ID, Un Canvas est un élément HTML permettant de faire des dessins et de modifier le contenu via un script écrit généralement en javascript. Nous choisirons donc pour notre Canvas une taille de 600 pixels x 600 pixels.

```

const CANVAS_SIZE = 600;
const CANVAS_BACKGROUND_COLOUR = "#232b2b";

var TITLE = document.getElementById("title");

var score = document.getElementById("score");
var aff_score = document.getElementById("red_serpent");
var cpt = 0;
var final_score = document.getElementById("red-final-score");
var score_val = document.getElementById("score-val");
var final_score2 = document.getElementById("both-final-score");
var score_val1 = document.getElementById("score-green");
var score_val2 = document.getElementById("score-red");

var score2 = document.getElementById("score2");
var aff_score2 = document.querySelector("#green_serpent");
var cpt2 = 0;

const GAME_SPEED = 100;
const SNAKE_SIZE = 15;
const SNAKE_DEFAULT_POSITION = [{x: 150, y: 150}, {x: 150, y: 135}, {x: 150, y: 120}];
const FOOD_DEFAULT_POSITION = {x: 135, y: 135};
const DEFAULT_DIRECTION = {x: 15, y: 0}; // Va a droite

const SNAKE_2_SIZE = 15;
const SNAKE_2_DEFAULT_POSITION = [{x: 210, y: 210}, {x: 210, y: 195}, {x: 210, y: 180}];
const DEFAULT_2_DIRECTION = {x: 15, y: 0}; //Va a droite

const START_BTN = document.querySelector("#start-btn");
const MULTI_START_BTN = document.querySelector("#multi-start-btn");
const GUIDE_BTN = document.querySelector("#guide-btn");
const BACK_BTN = document.querySelector("#back-btn");
const RESTART_BTN = document.querySelector("#restart-btn");
const RESTART_2_BTN = document.querySelector("#restart-2-btn");
const CONTINUE_BTN = document.querySelector("#continue-btn");
const QUITTE_BTN = document.querySelector("#quitter-btn");
const X_BTN = document.querySelector("#x-btn");
const GAMEOVER = document.querySelector("#gameover");
const INSTRUCTION = document.querySelector("#instruction");
const MODE_JEU = document.querySelector("#mode-jeu");

const CANVAS = document.querySelector("#game-canvas");
const CONTEXT = CANVAS.getContext("2d");

```

Figure 4 : Déclaration des variables & constantes

```

RESTART_BTN.addEventListener("click", function() {
  this.classList.add("disabled");
  CONTINUE_BTN.classList.add("disabled");
  GAMEOVER.classList.add("disabled");
  QUITTE_BTN.classList.add("disabled");
  aff_score.classList.remove("disabled");
  X_BTN.classList.remove("disabled");
  INSTRUCTION.classList.add("disabled");
  final_score.classList.add("disabled");
  cpt = 0 ;
  score.innerHTML = cpt;
  startGame();
})

```

```

RESTART_2_BTN.addEventListener("click", function() {
  this.classList.add("disabled");
  CONTINUE_BTN.classList.add("disabled");
  QUITTE_BTN.classList.add("disabled");
  GAMEOVER.classList.add("disabled");
  aff_score.classList.remove("disabled");
  aff_score2.classList.remove("disabled");
  X_BTN.classList.remove("disabled");
  final_score2.classList.add("disabled");
  cpt = 0 ;
  cpt2 = 0 ;
  score.innerHTML = cpt;
  score2.innerHTML = cpt2;
  startGame2();
})

```

```

CONTINUE_BTN.addEventListener("click", function() {
  this.classList.add("disabled");
  RESTART_BTN.classList.add("disabled");
  QUITTE_BTN.classList.add("disabled");
  GAMEOVER.classList.add("disabled");
  INSTRUCTION.classList.add("disabled");
  aff_score.classList.remove("disabled");
  X_BTN.classList.remove("disabled");
  final_score.classList.add("disabled");
  continueGame();
})

```

```

QUITTE_BTN.addEventListener("click", function() {
  location.reload();
})

```

```

START_BTN.addEventListener("click", function() {
  this.classList.add("disabled");
  MULTI_START_BTN.classList.add("disabled");
  aff_score.classList.remove("disabled");
  X_BTN.classList.remove("disabled");
  TITLE.classList.add("disabled");
  GUIDE_BTN.classList.add("disabled");
  startGame();
})

```

```

MULTI_START_BTN.addEventListener("click", function() {
  this.classList.add("disabled");
  START_BTN.classList.add("disabled");
  aff_score.classList.remove("disabled");
  X_BTN.classList.remove("disabled");
  aff_score2.classList.remove("disabled");
  TITLE.classList.add("disabled");
  GUIDE_BTN.classList.add("disabled");
  startGame2();
})

```

```

GUIDE_BTN.addEventListener("click", function(){
  this.classList.add("disabled");
  START_BTN.classList.add("disabled");
  MULTI_START_BTN.classList.add("disabled");
  TITLE.classList.add("disabled");
  BACK_BTN.classList.remove("disabled");
  var maxWidth = 570;
  var lineHeight = 25;
  var x = (CANVAS.width - maxWidth) / 2;
  var y = 100;
  var text = 'Le principe du jeu est de diriger un serpent vers des boules dont il doit se

  CONTEXT.font = '22px Calibri';
  CONTEXT.fillStyle = 'white';
  wrapText(CONTEXT, text, x, y, maxWidth, lineHeight);
  CONTEXT.font = "50px fantasy";
  CONTEXT.fillStyle = "#f32013";
  CONTEXT.textAlign = "center";
  CONTEXT.fillText("CONCEPT DU JEU", (CANVAS.width/2)-7, 50);
  drawImg();
})

```

Figure 5 : Les fonctions des boutons


```

✓ function drawCanvas() {
  CONTEXT.canvas.width = CANVAS_SIZE;
  CONTEXT.canvas.height = CANVAS_SIZE;
  CONTEXT.fillStyle = CANVAS_BACKGROUND_COLOUR;
  CONTEXT.fillRect(0, 0, CANVAS_SIZE, CANVAS_SIZE);
}

```

Figure 7: dessiner le canvas

```

//Dessiner 1 snake :
✓ function drawSnake(snake) {
✓   snake.forEach(part => {
    CONTEXT.fillStyle = '#4BB543';
    CONTEXT.strokeStyle = 'darkgreen';
    CONTEXT.fillRect(part.x, part.y, SNAKE_SIZE, SNAKE_SIZE);
    CONTEXT.strokeRect(part.x, part.y, SNAKE_SIZE, SNAKE_SIZE);
  });
}

//Dessiner 2eme snake :

✓ function drawSnake2(snake2) {
✓   snake2.forEach(part => {
    CONTEXT.fillStyle = '#F32013';
    CONTEXT.strokeStyle = 'darkred';
    CONTEXT.fillRect(part.x, part.y, SNAKE_2_SIZE, SNAKE_2_SIZE);
    CONTEXT.strokeRect(part.x, part.y, SNAKE_2_SIZE, SNAKE_2_SIZE);
  });
}

```

Figure 6: dessiner les serpents

```

function startGame() {
  const snake = [...SNAKE_DEFAULT_POSITION];
  const food = {...FOOD_DEFAULT_POSITION};
  const direction = {...DEFAULT_DIRECTION};

  setKeyBoardControl(direction);

  const interval = setInterval(() => {
    drawCanvas();

    updateSnake(snake, food, direction);
    drawSnake(snake);
    drawFood(food);

    checkGameOver(snake, interval);
  }, GAME_SPEED)
}

function continueGame() {
  const snake = [...SNAKE_DEFAULT_POSITION];

  const newSnakeHead = {...getSnakeHead(snake)};

  for(var i=0; i<cpt;i++){
    snake.unshift(newSnakeHead);
  }

  const food = {...FOOD_DEFAULT_POSITION};
  const direction = {...DEFAULT_DIRECTION};

  cpt = cpt-5;
  score.innerHTML = cpt ;

  setKeyBoardControl(direction);

  const interval = setInterval(() => {
    drawCanvas();

    updateSnake(snake, food, direction);
    drawSnake(snake);
    drawFood(food);

    checkGameOver(snake, interval);
  }, GAME_SPEED)
}

```

Figure 8 : les fonctions de jeu

```

function startGame2() {
  const snake = [...SNAKE_DEFAULT_POSITION];
  const snake2 = [...SNAKE_2_DEFAULT_POSITION];
  const food = {...FOOD_DEFAULT_POSITION};
  const direction = {...DEFAULT_DIRECTION};
  const direction2 = {...DEFAULT_2_DIRECTION};

  setKeyBoardControl(direction);
  setKeyBoardControl2(direction2);

  const interval = setInterval(() => {
    drawCanvas();

    updateSnake(snake, food , direction);
    updateSnake2(snake2, food , direction2);
    drawSnake(snake);
    drawSnake2(snake2);
    drawFood(food);

    checkGameOver2(snake, snake2, interval);
  }, GAME_SPEED)
}

```

Figure 9: multijoueur

Les fonctions pour diriger les serpents

```
function setKeyBoardControl2(direction2) {
  window.addEventListener("keydown", (e) => {
    const oldDirection = direction2;

    switch (e.key) {
      case 'z':
        if (oldDirection.y !== 0) return;
        direction2.x = 0;
        direction2.y = -SNAKE_SIZE;
        break
      case 's':
        if (oldDirection.y !== 0) return;
        direction2.x = 0;
        direction2.y = SNAKE_SIZE;
        break
      case 'q':
        if (oldDirection.x !== 0) return;
        direction2.x = -SNAKE_SIZE;
        direction2.y = 0;
        break
      case 'd':
        if (oldDirection.x !== 0) return;
        direction2.x = SNAKE_SIZE;
        direction2.y = 0;
        break
    }
  });
}
```

Figure 11: 2émé Serpent (Multi)

```
function setKeyBoardControl(direction) {
  window.addEventListener("keydown", (e) => {
    const oldDirection = direction;

    switch (e.key) {
      case 'ArrowUp':
        if (oldDirection.y !== 0) return;
        direction.x = 0;
        direction.y = -SNAKE_SIZE;
        break
      case 'ArrowDown':
        if (oldDirection.y !== 0) return;
        direction.x = 0;
        direction.y = SNAKE_SIZE;
        break
      case 'ArrowLeft':
        if (oldDirection.x !== 0) return;
        direction.x = -SNAKE_SIZE;
        direction.y = 0;
        break
      case 'ArrowRight':
        if (oldDirection.x !== 0) return;
        direction.x = SNAKE_SIZE;
        direction.y = 0;
        break
    }
  });
}
```

Figure 10: Premier Serpent (SOLO)

Les fonctions pour vérifier la fin du jeu

```

function checkGameOver(snake, interval) {
  if (snakeEatItself(snake) || snakeHitWall(snake)) {
    clearInterval(interval);
    GAMEOVER.classList.remove("disabled");
    RESTART_BTN.classList.remove("disabled");
    QUITTE_BTN.classList.remove("disabled");
    aff_score.classList.add("disabled");
    X_BTN.classList.add("disabled");
    final_score.classList.remove("disabled") + (score_val.innerHTML = cpt) ;
    drawCanvas()
    if(cpt>=5){
      CONTINUE_BTN.classList.remove("disabled");
      INSTRUCTION.classList.remove("disabled");
    }
  }
}

function checkGameOver2(snake, snake2, interval) {
  if (snakeEatItself(snake) || snakeHitWall(snake) ||snakeEatItself2(snake2) || snakeHitWall2(snake2) || snakeHitSnake(snake, snake2)) {
    clearInterval(interval);
    GAMEOVER.classList.remove("disabled");
    RESTART_2_BTN.classList.remove("disabled");
    QUITTE_BTN.classList.remove("disabled");
    aff_score.classList.add("disabled");
    aff_score2.classList.add("disabled");
    X_BTN.classList.add("disabled");
    final_score2.classList.remove("disabled") + (score_val1.innerHTML = cpt)+(score_val2.innerHTML = cpt2);
    drawCanvas();
  }
}

```

Figure 12: GAMEOVER 1 - 2

```

function updateSnake(snake, food, direction) {
  const newSnakeHead = {...getSnakeHead(snake)};

  newSnakeHead.x += direction.x;
  newSnakeHead.y += direction.y;

  snake.unshift(newSnakeHead);

  if (eatFoodSuccessfully(snake, food)) {
    createFood(snake, food);
    majScore(cpt+=1);
  } else {
    snake.pop();
  }
}

//mis a jour du 2éme serpent :

function updateSnake2(snake2, food, direction) {
  const newSnakeHead2 = {...getSnakeHead2(snake2)};

  newSnakeHead2.x += direction.x;
  newSnakeHead2.y += direction.y;

  snake2.unshift(newSnakeHead2);

  if (eatFoodSuccessfully2(snake2, food)) {
    createFood2(snake2, food);
    majScore2(cpt2+=1);
  } else {
    snake2.pop();
  }
}

```

Figure 13 : grandir la taille & cumulé le score

```

function eatFoodSuccessfully(snake, food) {
  const snakeHead = getSnakeHead(snake);
  return snakeHead.x === food.x && snakeHead.y === food.y;
}

function eatFoodSuccessfully2(snake2, food) {
  const snakeHead2 = getSnakeHead2(snake2);
  return snakeHead2.x === food.x && snakeHead2.y === food.y;
}

//Les fonctions pour calculer les scores:

function majScore(s){
  score.innerHTML= s;
}

function majScore2(s2){
  score2.innerHTML= s2;
}

```

Figure 14: vérifier que les serpents ont mangé la nourriture & ajoute de score

```

//Les fonctions pour verifier si le serpent se mange lui-même :

function snakeEatItself(snake) {
  const snakeHead = getSnakeHead(snake);
  const snakeBody = snake.slice(4)
  return snakeBody.some(part => snakeHead.x === part.x && snakeHead.y === part.y);
}

function snakeEatItself2(snake2) {
  const snakeHead2 = getSnakeHead2(snake2);
  const snakeBody2 = snake2.slice(4)
  return snakeBody2.some(part => snakeHead2.x === part.x && snakeHead2.y === part.y);
}

//Les fonctions pour verifier si le serpent a écraser le mur :

function snakeHitWall(snake) {
  const snakeHead = getSnakeHead(snake);

  return snakeHead.x >= CANVAS_SIZE
  || snakeHead.y >= CANVAS_SIZE
  || snakeHead.x < 0
  || snakeHead.y < 0
}

function snakeHitWall2(snake2) {
  const snakeHead2 = getSnakeHead2(snake2);

  return snakeHead2.x >= CANVAS_SIZE
  || snakeHead2.y >= CANVAS_SIZE
  || snakeHead2.x < 0
  || snakeHead2.y < 0
}

function snakeHitSnake(snake, snake2) {
  const snakeHead = getSnakeHead(snake);
  const snakeHead2 = getSnakeHead2(snake2);
  const snakeBody = snake.slice(0);
  const snakeBody2 = snake2.slice(0);
  return snakeBody2.some(part => snakeHead.x === part.x && snakeHead.y === part.y)
  || snakeBody.some(part => snakeHead2.x === part.x && snakeHead2.y === part.y);
}

```

Figure 15: vérifié les règles du jeu (si le serpent touche le mur, mange lui-même)

```

function createFood(snake, food) {
  const newPosition = randomPosition();

  while (snake.some(part => part.x === newPosition.x && part.y === newPosition.y)) {
    newPosition = randomPosition();
  }

  food.x = newPosition.x;
  food.y = newPosition.y;
}

function createFood2(snake2, food) {
  const newPosition = randomPosition();

  while (snake2.some(part => part.x === newPosition.x && part.y === newPosition.y)) {
    newPosition = randomPosition();
  }

  food.x = newPosition.x;
  food.y = newPosition.y;
}

//Dessiner la nourriture :

function drawFood(food) {
  CONTEXT.fillStyle = 'Orange';
  CONTEXT.strokeStyle = 'OrangeRed';
  CONTEXT.fillRect(food.x, food.y, SNAKE_SIZE, SNAKE_SIZE);
  CONTEXT.strokeRect(food.x, food.y, SNAKE_SIZE, SNAKE_SIZE);
}

//Pour les positions aleatoire :

function randomPosition() {
  return {
    x: Math.round((Math.random() * (CANVAS_SIZE-SNAKE_SIZE) + SNAKE_SIZE) / SNAKE_SIZE) * SNAKE_SIZE,
    y: Math.round((Math.random() * (CANVAS_SIZE-SNAKE_SIZE) + SNAKE_SIZE) / SNAKE_SIZE) * SNAKE_SIZE,
  }
}

```

```

function wrapText(CONTEXT, text, x, y, maxWidth, lineHeight) {
    var words = text.split(' ');
    var line = '';

    for(var n = 0; n < words.length; n++) {
        var testLine = line + words[n] + ' ';
        var metrics = CONTEXT.measureText(testLine);
        var testWidth = metrics.width;
        if (testWidth > maxWidth && n > 0) {
            CONTEXT.fillText(line, x, y);
            line = words[n] + ' ';
            y += lineHeight;
        }
        else {
            line = testLine;
        }
    }
    CONTEXT.fillText(line, x, y);
}

//Ajouter les images du guide:

function drawImg(){
    let img = new Image(); // Crée un nouvel élément img
    let img2 = new Image(); // Crée un nouvel élément img
    img.onload = function(){
        CONTEXT.drawImage(img, 280, 335, 120, 70);
    }
    img2.onload = function(){
        CONTEXT.drawImage(img2, 280, 425, 120, 70);
    }
    img.src = 'fleche.png'; // définit le chemin de la source
    img2.src = 'zqsd.png'; // définit le chemin de la source
}

//pour faire signaler le mode du jeu :

setInterval(function(){
    MODE_JEU.classList.remove("disabled");
    setTimeout(function(){
        MODE_JEU.classList.add("disabled");
    }, 700)
}, 700)

```

Figure 17: les fonctions du GUIDE

PARTIE 6 : Bilan



1- Etat d'avancement :

Tâches	Avancement
Cahier des charges	0% 25% 50% 75% 100%
Analyse détaillé	0% 25% 50% 75% 100%
Conception détaillé	0% 25% 50% 75% 100%
Développement Des interface HTML	0% 25% 50% 75% 100%
Développement Des Fonctionnalités en JS	0% 25% 50% 75% 100%

2- Bilan :

Ce qui a marché	Les problèmes rencontrés
Communication parfaite	Imprévu : on a travaillé avec un seul laptop
Organisations et respect de plan	Des problèmes liés à la mise en place de certaines fonctionnalités en javascript
Travail en équipe	

3- Leçons apprises :

- Dans ce projet on a appris plusieurs choses comme le travail en équipe (travailler seul on ira plus vite mais ensemble on ira plus loin).
- Ne pas se bloquer dans les taches où il y a des problèmes et passer à d'autre taches pour avancer.

Conclusion

Ainsi ce rapport permet d'avoir une vision globale du projet demandé, il rappelle dans un premier temps tout ce qu'il faut savoir avant de commencer à réfléchir à une solution, à savoir le besoin, l'existant et les moyens dont on dispose.

Et de ce fait en prenant en compte toutes les contraintes qui existent, une solution a été élaborée et avec ceci une liste de toutes les fonctionnalités qui ont été décrites dans le détail, ce qui a permis au client de valider le projet.

Nous avons alors réalisé ce projet et l'avons expliqué du début à la fin, ce qui nous a permis d'avoir une vision d'ensemble sur son fonctionnement avec le code qui a été élaboré, ainsi que toutes les étapes d'analyse qui ont mené à son aboutissement.

REFERENCES :

¹ [https://fr.wikipedia.org/wiki/Snake_\(genre_de_jeu_vid%C3%A9o\)](https://fr.wikipedia.org/wiki/Snake_(genre_de_jeu_vid%C3%A9o)) : **Jeu de de serpent**

² <https://www.jeux-gratuits.com/jeu-3d-snake.html> : **jeu Snake 3D**

³ <https://www.jeux-gratuits.com/jeu-snakes-multijoueur.html> : **Jeu Snake multijoueur**

⁴ **Cahier des charges du Module MACSI Catherine Barry et Anne Lapujade**

⁵ <https://www.figma.com/file/V0IS5uPsxv3mJrS3ZRbhYQ/JEU-DE-SERPENT?node-id=0%3A1> : **Maquettes dynamique de jeu**

⁶ <https://horustest.io/blog/definition-test-fonctionnel/> : **Les tests fonctionnels**

https://fr.wikipedia.org/wiki/M%C3%A9thode_MoSCoW : **La methode de Moscow**