



DÉPARTEMENT D'INFORMATIQUE

HMIN217 - TRAVAUX ENCADRÉS DE RECHERCHE

Programmation d'un Jeu en JavaScript

Travail réalisé et soutenu par :

ABID HAMZA

EL AYYADI IKRAM

LAKHDAR IDRISI MERYEME

RONDOT MAXENCE

VICTOR SANDRA

Professeur référent :

M.PIERRE POMPIDOR

Année universitaire : 2020-2021

Remerciements

Nous tenons à remercier M.Pierre POMPIDOR, notre professeur référent, pour sa patience, sa disponibilité et surtout ses judicieux conseils, qui ont contribué à nous améliorer dans le développement de notre projet et qui a su nous guider autant dans le code que dans l'organisation .

Nous tenons à remercier aussi les membres de jury pour leur présence, lecture et ainsi toutes remarques adressées lors de cette soutenance pour améliorer notre travail.

Table des matières

1 Présentation du projet	1
1.1 Enjeux du TER	1
1.2 Présentation du jeu	1
2 Analyse Fonctionnelle	1
2.1 Diagramme de cas d'utilisation	1
2.2 Scénario	2
3 Conception et programmation	3
3.1 Diagramme de classes	3
3.2 Langages et technologies utilisées	3
4 Gestion de projet	4
4.1 Outils utilisés	4
4.2 Diagramme de Gantt prévisionnel	5
5 Développement	5
5.1 Éditeur de niveau	5
5.2 Graphisme	6
5.3 Gameplay	6
5.3.1 Interface	6
5.3.2 Commencement	7
5.3.3 Avatar et Déplacement	8
5.3.4 Objets et Inventaire	8
5.3.5 Ennemis	9
5.3.6 Portes et Fin	10
5.3.7 Confirm et Alert	10
5.4 Énigmes	11
5.5 Poursuivant	11
5.5.1 But	11
5.5.2 Structure de graphe	11
5.5.3 Algorithme A* et parcours en profondeur	12
5.5.4 Problèmes et améliorations possibles	14
6 Conclusion	15
6.1 Diagramme de Gantt final	15

6.2	Améliorations générales	15
6.3	Apports personnels	16
Annexes		18
A	Outils utilisés	18
B	Gameplay	19
B.1	Commencement	19
B.2	Objets et Inventaire	20
B.3	Confirm et Alert	21
C	Poursuivant	22
C.1	Structure de graphe	22
C.2	Algorithm	22
D	Editeur	23
E	Enigmes	24

1 Présentation du projet

1.1 Enjeux du TER

Nous avons eu l'opportunité de découvrir quelque chose de nouveau et qui a peut-être pu remplacer le fait que nous ne sommes pas en stage grâce au Travaux Encadrés de Recherche avec le sujet que nous avons choisi qui est de créer un jeu en JavaScript de type 2d aventure et énigme. Cet UE nous a permis de travailler autrement et d'approfondir sur un sujet qui nous intéresse et de mener ensemble l'intégralité du travail, ce qui a nécessité des réunions virtuelles fréquentes (vue le contexte actuel) sur de larges plages horaires. Autrement dit il a fallu que nous nous répartissions le travail et fixer des rendez-vous tout en sachant que nous n'avons pas les mêmes options, ce qui implique différents planning.

Grâce à ce TER, nous avons pu nous concentrer sur l'aspect pratique de JavaScript. Ceci permet également d'appréhender le fonctionnement d'un projet à plusieurs.

1.2 Présentation du jeu

Notre jeu s'inspire du prototype proposé par M.Pompidor : Un jeu 2D en JavaScript de type «Lost».

Ozou est un jeu développé en JavaScript et se joue seul à l'aide d'un navigateur.

Le jeu se déroule dans un hôtel et commence par une interface simple dans laquelle le joueur est invité à choisir le niveau et à commencer la partie. Le joueur se déplace à gauche à droite, en avant et en arrière pour éviter les ennemis. Le joueur est face à un ennemi qui essaye de l'attraper tout au long de la partie. Un inventaire d'objets est vide au commencement de la partie. D'une manière aléatoire le joueur possède à sa disposition des objets de soins pour augmenter la vie, et des objets défenses pour affronter les différentes épreuves que le joueur peut avoir : un couteau sera efficace contre un zombie par exemple. Le niveau peut changer en fonction de la réussite de niveau et cela se fait grâce à des escaliers qui nous descendent au niveau suivant.

La complexité du jeu réside dans sa liberté. De nombreuses décisions doivent être faites pour le mener à bien et accomplir l'objectif principal : avoir un jeu fonctionnel. Pour cela beaucoup de compétences ont été utilisées : développement web, algorithme et graphisme. Cela fut un challenge, car beaucoup de travail fut nécessaire pour développer ce jeu. Certaines technologies utilisées étaient au préalable inconnues et il était nécessaire de les maîtriser assez vite (Node.js par exemple).

Pour réussir, un développement en cascade était nécessaire, en découplant chaque tâche : analyse, conception, développement....etc. De plus, les tâches de développement devaient être correctement distribuées entre nous pour une meilleure organisation et productivité.

2 Analyse Fonctionnelle

2.1 Diagramme de cas d'utilisation

L'intérêt des diagrammes utilisés est de permettre une meilleure compréhension et visualisation du fonctionnement de notre jeu, ils permettent aussi de fournir une description indépendante de l'implémentation des types utilisés dans notre code.

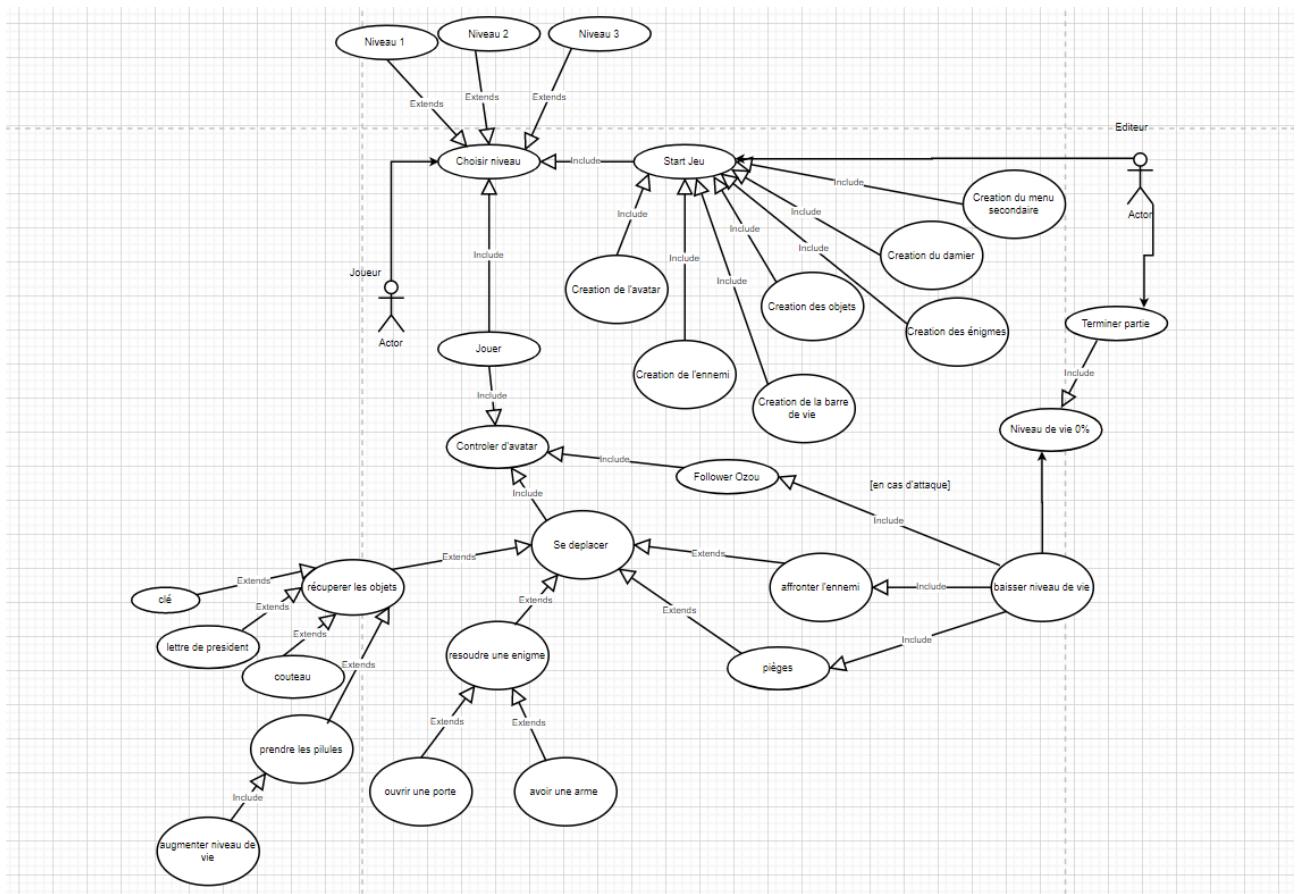


FIGURE 1 – Diagramme cas d'utilisation

2.2 Scénario

Ozou est donc un jeu de survie avec une ambiance d'horreur. Il se déroule dans un vieil hôtel du petit village français d'Ozouer-le-Voulgis. Un général de division de la Grande Armée de Napoléon aurait disparu lors de la visite de ce dernier dans cette bourgade. Depuis son cadavre possédé traque en permanence toute présence humaine qui ose entrer dans la vieille bâtie, prenant comme dénomination les deux premières syllabes du village.// Horatio Nelson, jeune homme d'une vingtaine d'année, se retrouve malheureusement dans cet hôtel suite à un enterrement. C'est en plein milieu de la nuit qu'il se réveille, démarrant ainsi la partie pour le joueur qui l'incarne.// Un exemple type de partie pour le niveau 1 pourrait se dérouler ainsi :

- Le joueur commence dans son lit, dans sa chambre, l'inventaire vide. Il part explorer le meuble dans sa chambre, et trouve dedans 2 objets. Il choisit de récupérer ces 2 objets et les mets dans son inventaire. Le premier est un couteau et le second une lettre de l'un de ses proches. Le joueur lit cette note, et la conserve.
- Il sort ensuite de sa chambre, il remonte jusqu'au meuble dans le couloir, et trouve des pilules qu'il conserve également. Il passe ensuite dans la pièce à gauche et décide d'aller directement voir le meuble le plus lointain. Il tombe cependant face à un zombie, qu'il peut cependant abattre avec son couteau, et sans sort sans blessure, mais n'a également plus de couteau.
- Il arrive finalement au meubles, qui est cependant vide. Il prend alors la décision de sortir de la pièce en ignorant le second meuble, traverse le couloir et se retrouve dans la troisième pièce.
- Dans celle-ci il marche sur un piège à ours, qui réduit alors sa santé d'un certain montant. Il passe au travers de la porte ouverte en bas de la pièce, et rentre dans cette petite pièce. En s'avancant pour explorer le meuble, il tombe dans un piège de piques, réduisant encore

sa santé, mais parvient tout de même au meuble. Il récupère dans celui-ci une clé et une armure de fortune. Il prend ces 2 objets, puis repasse par les piques, cette fois-ci protégé par l'armure. Il remonte, et décide d'aller directement à la porte de l'escalier.

- Il rencontre cependant Ozou dans le couloir. Ce dernier arrive à l'attraper et lui infliger des dommages. Cependant, le joueur parvient à s'enfuir, rentre dans la salle avec les 2 meubles, et se déplace le temps qu'Ozou rentre dans la pièce et s'avance. Le joueur parvient alors à le contourner, sort de la pièce, et ferme la porte derrière lui. Il consomme ses pilules pour regagner de la vie puis se déplace jusqu'à la porte. Il ouvre cette dernière avec la clé, et prend les escaliers, réussissant ce niveau.

3 Conception et programmation

3.1 Diagramme de classes

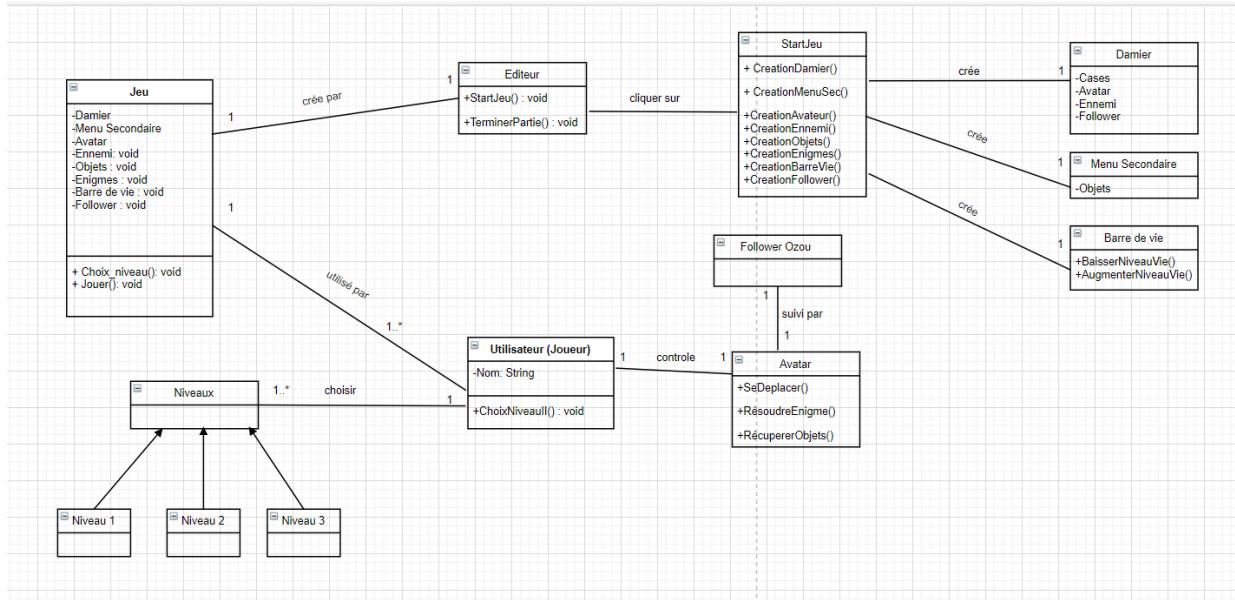


FIGURE 2 – Diagramme de classes

3.2 Langages et technologies utilisées

En premier lieu, c'est le défi technologique qui nous a attirés. Notre jeu fait intervenir plusieurs fonctionnalités. Nous avons voulu à travers ce projet, tester notre capacité à faire retourner des rendus graphiques. C'est pour cela qu'on a utilisé JavaScript (version 6) puisqu'il est le seul langage de haut niveau pouvant être directement interprété par le navigateur et permettant d'utiliser toutes les fonctionnalités de celui-ci .

Outre la technique, Javascript dispose d'une des plus grandes communauté dans le développement web. En effet comme évoqué précédemment, nous avons besoin des rendus graphiques mais par le biais de D3.js qui est "une bibliothèque JavaScript pour la manipulation de documents basés sur des données". Elle permet une visualisation interactive et dynamique de ces derniers. Nous avons été amené à travailler avec cette librairie tout au long du projet, c'est une librairie qui est très grande, riche et regroupe plusieurs concepts. Elle peut zoomer, déplacer, trier, filtrer les éléments qui constituent un graphe.

D'un point de vue backend, nous avons utilisé Node.js qui est un environnement d'exécution permettant d'utiliser JavaScript côté serveur. Grâce à son fonctionnement non bloquant, il permet de concevoir des applications en réseau performantes, tel qu'un serveur web, une API ou un job CRON.

4 Gestion de projet

4.1 Outils utilisés

Pour réaliser ce projet, nous avons été amené à utiliser de nombreux outils afin de pouvoir coordonner et partager notre travail. Parmi ces outils, la plate-forme de développement en ligne Github nous a permis de nous partager de manière régulière et efficace nos différents codes. Cela nous a permis de conserver les dernières modifications de chaque membre du groupe ainsi que de conserver une version stable de notre jeu.

Draw.io est un logiciel en ligne permettant de réaliser de nombreux diagrammes. Nous l'avons utilisé pour réaliser nos différents diagrammes, ainsi que pour schématiser nos idées.

Pour nous permettre de communiquer efficacement et à tout moment, nous avons créer un serveur privé sur la plate-forme de communication digital Discord(voir annexe : A). Nous avons pu ainsi échanger au travers de messagerie instantanée, mais également réaliser certaines de nos réunions et tenir au courant les autres de notre avancement ou des difficultés rencontrées. Chaque partie du développement que nous nous étions attribuée avait ainsi ses propres channels de messagerie, permettant un échange visible pour chacune de ces parties.

Il nous est également arrivé d'utiliser l'outil de conférence vidéo Zoom lors de nos réunions, car il s'est avéré que certains membres du groupes avaient de gros problèmes de connexion lors de discussions vocales sur Discord, mais aucun sur Zoom, nous avons donc dû bouger à un certain point vers cette plateforme pour mener nos réunions à bien, même si Discord restait notre outil de messagerie instantanée.

Pour réaliser nos diagrammes de Gantt, nous avons utilisé l'extension gratuite de Google Sheet SlickGantt qui nous a permis de réaliser des diagrammes que nous pouvions modifier de manière dynamique, au travers d'un fichier Google sheet partagé facilement entre nous. Nous nous sommes tourné vers ce logiciel car il est gratuit, plutôt facile d'accès, surtout en ayant l'habitude de l'utilisation de tableur, et nous donner à tous un accès facile aux diagrammes.

Overleaf est l'outil de création collaboratif que nous avons utilisé pour réaliser ce rapport. La plupart d'entre nous l'avaient déjà utilisé, et la maniabilité du fichier et la capacité de pouvoir réaliser un travail collaboratif facilement nous ont convaincu de l'utiliser pour rédiger notre rapport.

4.2 Diagramme de Gantt prévisionnel

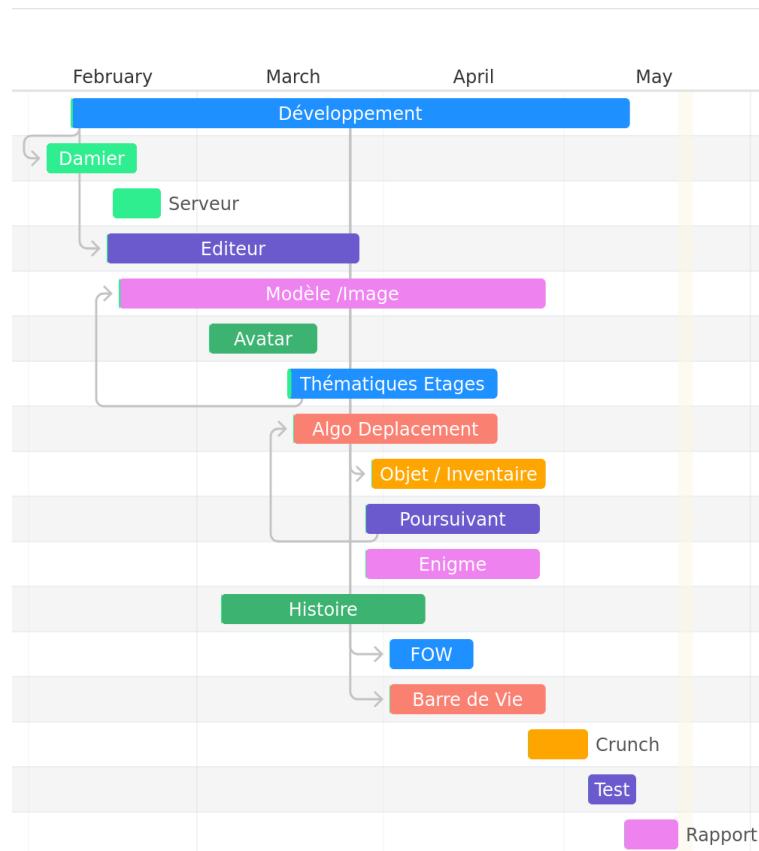


FIGURE 3 – Diagramme de Gantt

Initialement, nous avions prévu de terminer le développement 1 à 2 semaines avant la fin pour nous laisser une phase de test confortable afin de corriger les différents bugs que l'on pouvait rencontrer. Pour le développement du jeu, nous nous sommes répartis les tâches en identifiant un leader pour chaque partie importante du projet. Hamza se chargeait de l'éditeur de niveau, Ikram s'occupait du côté graphique du jeu. Meryeme réfléchissait sur le développement des énigmes que ça soit pour les objets ou ennemis. Maxence s'est concentré sur l'interface du jeu et donc les interactions que l'on va retrouver lors d'une partie et Sandra a travaillé sur l'implémentation de l'ennemi poursuiveur et donc de la partie algorithme du projet. Cependant, le travail restait commun sur certaines tâches.

5 Développement

5.1 Éditeur de niveau

L'éditeur est une interface pour créer des niveaux avec toutes les possibilités offertes par le jeu, et permet à une personne de créer d'une manière simple son niveau sans devoir le programmer. Cette personne est nommée "level designer". L'éditeur permet de concevoir une scène sous forme d'un niveau. Le menu principale permettant d'effectuer cette tâche se compose en deux volets principaux. À gauche, un damier qui se remplit au fur et à mesure avec des images, contenues dans le volet de droite. Ce dernier volet contient deux types d'images :

1. Les images de fond
2. Les images des objets

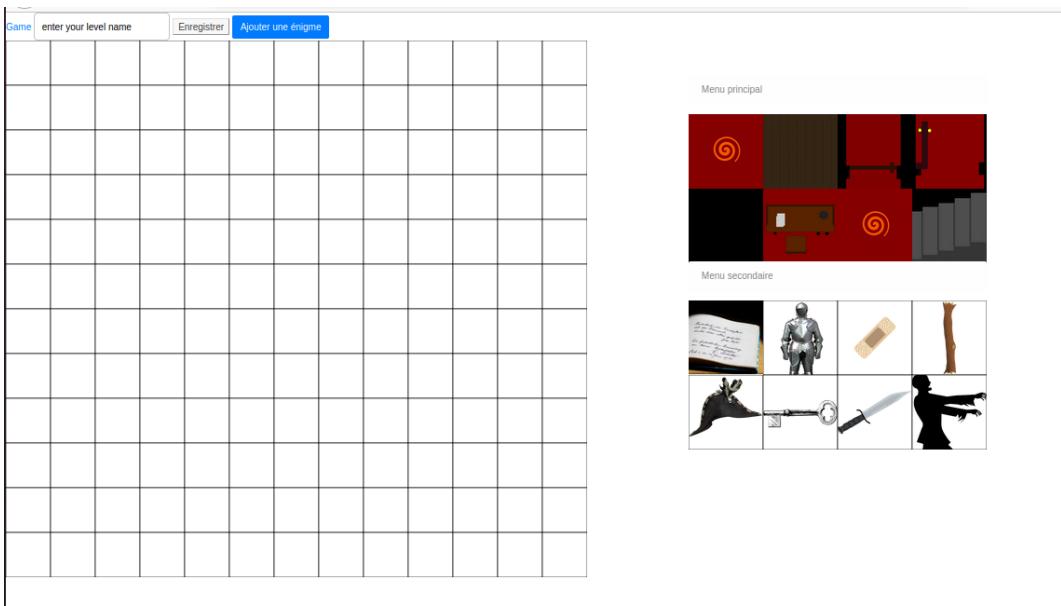


FIGURE 4 – Interface éditeur de niveaux

Toutes ces images possèdent l’attribut onClick, qui permet d’exécuter une fonction JavaScript (voir figure 21), dès que l’utilisateur clique sur une image. cette fonction permet de stocker le type et chemin relatif de l’image dans sessionStorage. Ces informations stockés seront réutilisées pour ajouter l’image préalablement sélectionnée à la case choisie. Pour cela nous avons utilisé la fonction append() et attr.

Le serveur web implémente une route <POST /niveau/>. Cette route récupère tout ce qui est envoyé par l’éditeur et le stocke simplement dans un fichier.

Ce stockage se fait d’une manière très simple. En effet, il suffit d’ajouter un niveau avec un nom déjà existant. L’ancien niveau sera simplement écrasé. La partie conception de niveaux n’implémente pas d’authentification.

5.2 Graphisme

Nous avons pensé à créer nos propres images pour donner au jeu un aspect original.

Les images sont sous format 2d et le logiciel utilisé est Inkscape. C’est un logiciel de dessin vectoriel qui permet la création de graphisme vectoriel, gratuit et libre pour apporter des modifications. Inkscape est puissant en vue de la flexibilité de ses outils de dessins, compatible avec tous les formats de fichiers les plus courants (SVG, EPS, PDF, PNG...) et les courbes qu’ils proposent notamment les courbes de Bézier et spirographiques. Ses nombreux avantages ont fait de lui un choix évident. Inkscape demande un peu de temps pour la prise en main, mais faisant gagner du temps par la suite. Cette expérience était vraiment fructueuse, j’ai appris beaucoup de choses, ne connaissant pas le domaine du graphisme cela m’a permis d’avoir une simulation visuelle en s’adaptant au thème du jeu.

5.3 Gameplay

5.3.1 Interface

Comme énoncé précédemment, le modèle Json est une liste de liste d’objets JavaScript qui permet de récupérer l’intégralité des données de nos niveaux. Nous utilisons pour cela un menu déroulant (1) qui permet au joueur d’aller chercher le niveau voulu dans ceux disponibles au niveau du serveur.

On note sur la droite de l’écran du haut vers le bas la présence de :

-
- La barre de vie (2) du joueur qui permet de visualiser les points de vie restants du joueur.
 - L'inventaire de case (3) dans lequel est affiché le contenu de la case sur lequel le joueur se trouve, que ce soit les objets dans les meubles, ou les ennemis dans les cases pièges.
 - L'inventaire (4) du joueur, où le joueur peut stocker des objets trouvés et utilisés ces derniers.

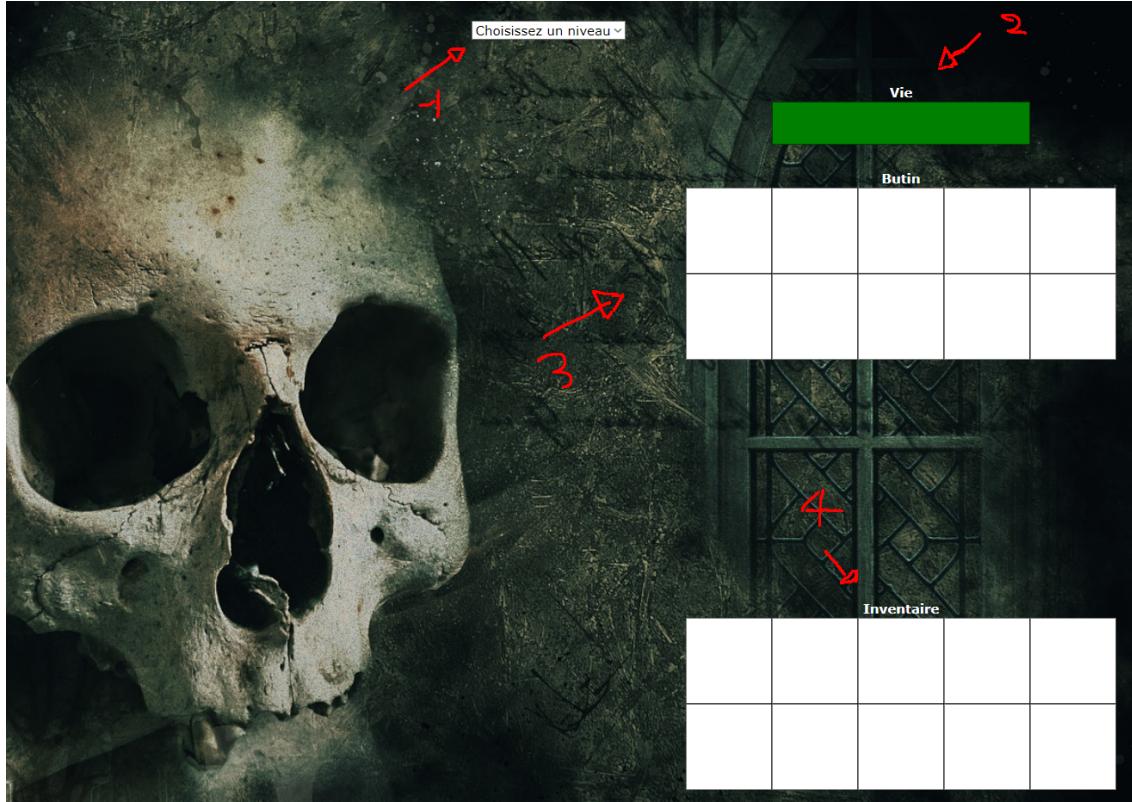


FIGURE 5 – Interface début de partie

5.3.2 Commencement

Lors de la sélection du niveau grâce au menu déroulant, les données des objets Javascript contenues dans le fichier JSON sont exploitées par la fonction genereModele (voir annexe B.1). Cette dernière va nous permettre de lire les données extraites du JSON, et qui va attribuer à chaque case une image, en fonction de la valeur de la propriété "type" de l'objet lié à cette case. On utilise également l'attribut d3 "transform" afin d'orienter de la bonne façon les différentes portes, en se basant sur la présence des murs adjacents. Une introduction et une petite aide sont également proposés au joueur s'il le souhaite.

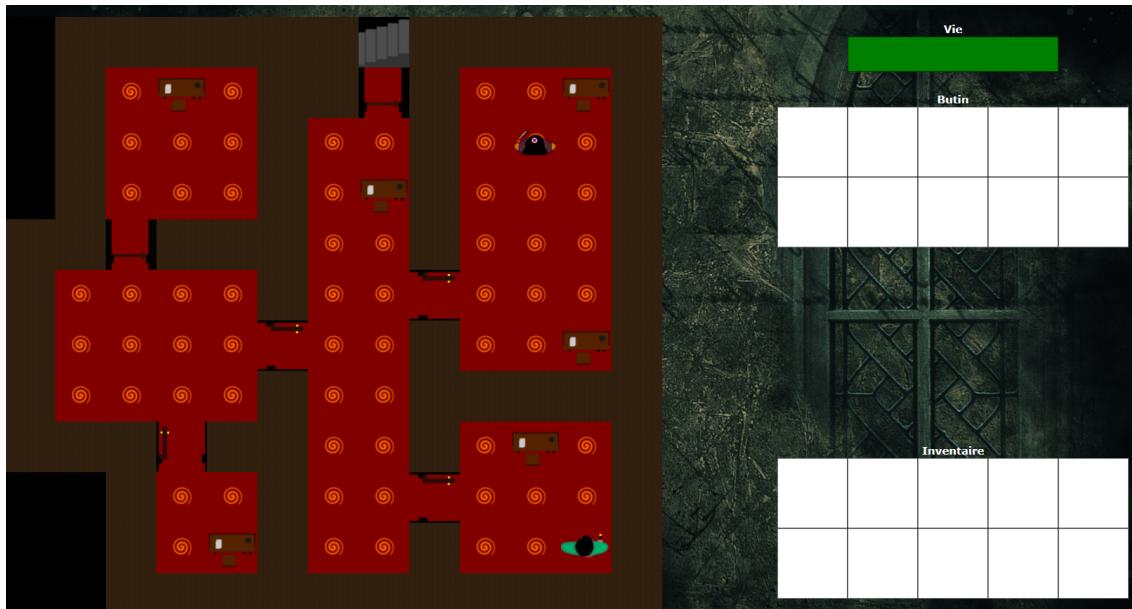


FIGURE 6 – Début d'une partie

5.3.3 Avatar et Déplacement

Notre avatar, Horatio Nelson, se déplace grâce aux touches "ZQSD". Son déplacement se fait en regardant en permanence la case vers laquelle il se dirige. Ainsi, tout déplacement est proscrit sur les cases infranchissables, tel les murs ou les portes verrouillées. Chaque déplacement se fait grâce à une fonction différentes, en fonction de la direction empruntée. Une fois le déplacement effectué, on utilise la fonction "Interaction" qui regarde le type de la case visitée pour déterminer l'interaction possible.

5.3.4 Objets et Inventaire

Ainsi, si la case visitée est un meuble, la fonction "genereObjet" est utilisée pour afficher dans l'inventaire de case les différents objets présents dans cette dernière. Chacun de ces objets possède la fonction "getItem" (voir annexe "B.2") en évènement "onclick". Cette dernière permet au joueur de récupérer dans son inventaire les objets en les sélectionnant, les supprimant ce faisant du modèle. On peut trouver différents types d'objets. Un des principaux problèmes de l'implémentation de cette fonction, fut le besoin de passer l'objet de la case en paramètre. En effet, il fut impossible d'écrire dans l'attribut html "onclick" l'objet en question. La décision prise a été de mettre en place une variable générale "obj", qui permet de sauvegarder l'objet case utilisé, afin de pouvoir le réutiliser plus tard lorsque que celui ci est récupérer, pour éviter une multiplication des objets. Les différents objets trouvables sont d'ailleurs tous caractérisés dans le fichier "ressources.js" (voir annexe "B.2") sous la forme d'objet javascript, ayant chacun un constructeur afin de faciliter le rajout d'élément à l'avenir. Leurs propriétés caractérisent leur actions. La valeur de soin pour les pilules par exemple, ou le nom de chaque objet.

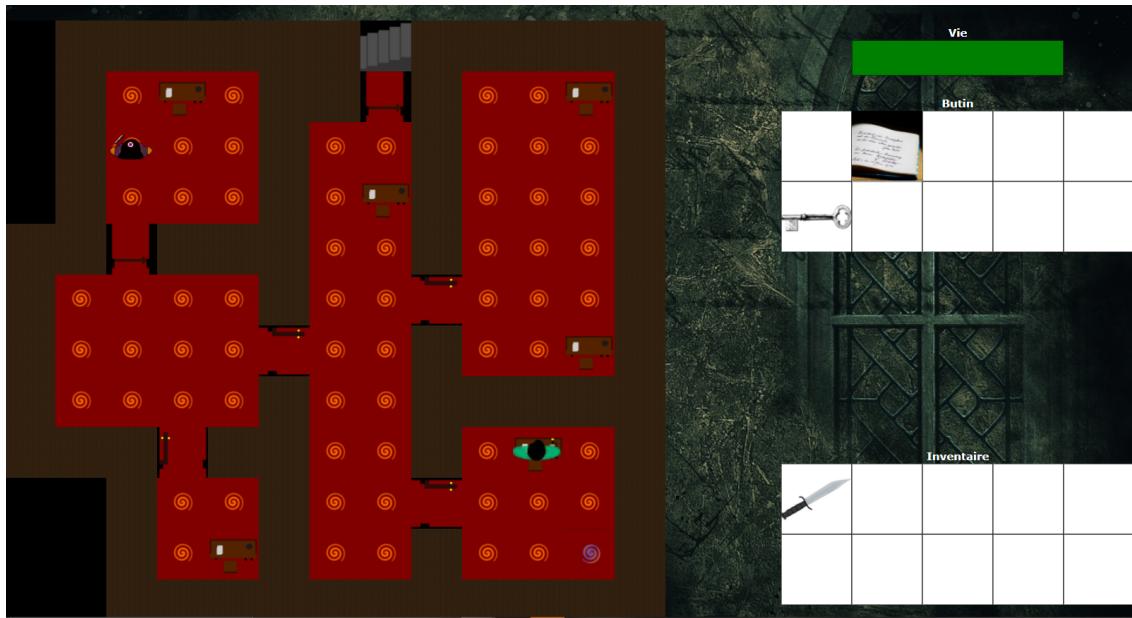


FIGURE 7 – Interaction avec le meuble, avec des objets dans les deux inventaires

Tout d'abord on trouve les différents objets d'aide, comme des informations sur l'histoire ou en aide des énigmes, qui sont consultables autant de fois que voulu en les sélectionnant depuis l'inventaire. Les objets de soin sont également utilisables depuis l'inventaire. Ils lancent la fonction "updateHealth" avec un paramètre positif, rendant de la vie au joueur, mais sont à usage unique. Les clefs sont les éléments les plus rares et importants car ceux sont elles qui permettent d'ouvrir les portes verrouillées. Le dernier groupe d'objet sont les objets défensifs. Également à usage unique, ils défendent le joueur si ce dernier tombe dans le piège dont ils sont les contres, ils ne sont donc utilisé que en contre attaque et ne peuvent être utilisés activement depuis l'inventaire.

5.3.5 Ennemis

Les ennemis sont présents dans le modèle de la même manière que les objets avec la méthode "genereEnnemi", mais uniquement sur les cases piéges. Ils réduisent les points de santé d'Horatio d'un montant fixe, grâce à la méthode "updateHealth", sauf si ce dernier possède l'objet défensif associé, se protégeant mais éliminant également la menace à l'avenir.

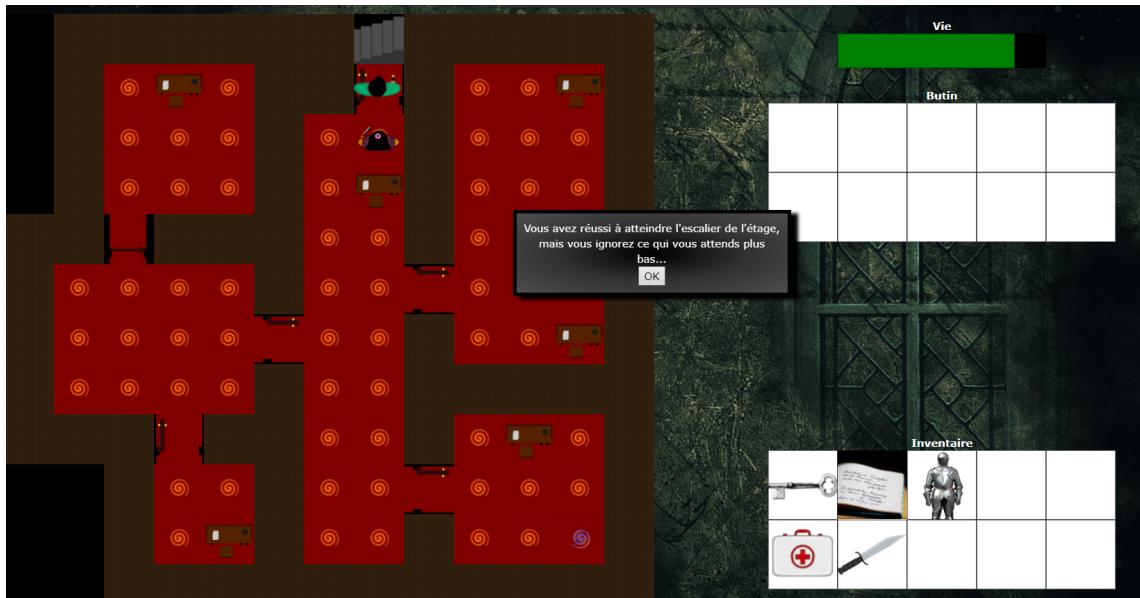


FIGURE 8 – Interaction avec un ennemi, ici un zombie

5.3.6 Portes et Fin

Le joueur peut effectuer trois interactions avec les portes, dépendant du fait quelles soient ouvertes, déverrouillées ou verrouillées. Il peut les ouvrir ou les fermer au travers de différentes fonctions qui modifient directement le type de l'objet responsable de la case en question, permettant de sauvegarder efficacement les actions du joueur, ce qui aura son importance face à Ozou. S'il parvient à survivre à l'étage et atteindre l'escalier, Horatio conservera sa vie et ses objets. Ozou le suivra également à l'étage en dessous, jusqu'à l'étage final.

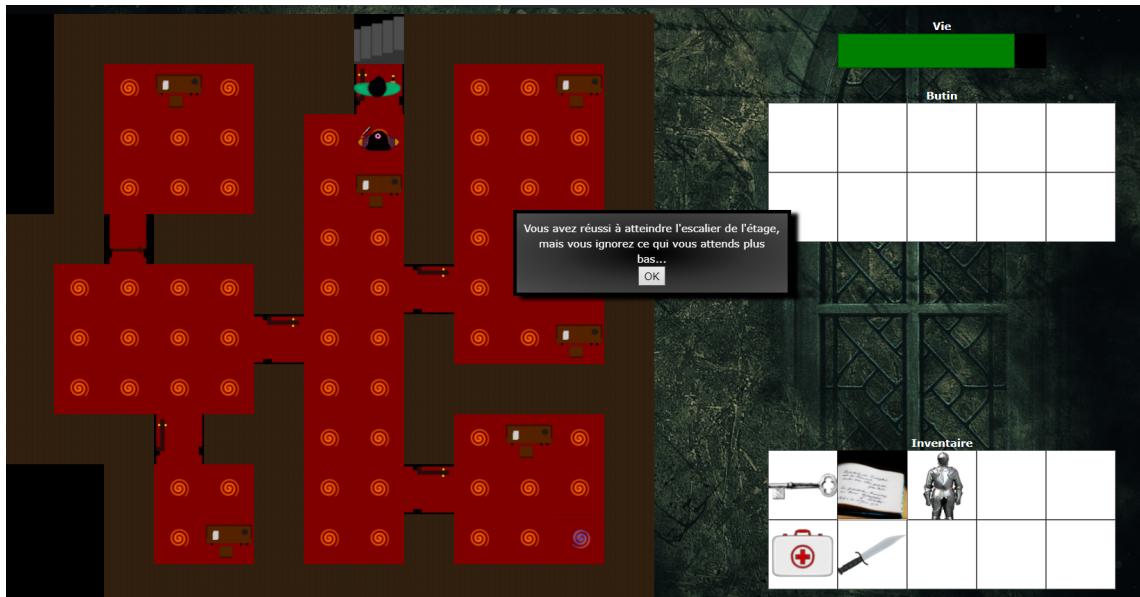


FIGURE 9 – Fin de niveau

5.3.7 Confirm et Alert

Le principal problème rencontré au cours de cette partie du développement fut sans aucun doute le problème lié aux blocages violents réaliser par "alert" et "confirm", qui sont beaucoup utilisés

par les différentes fonctions. En effet, en plus de hacher l'expérience de jeu, elles peuvent créer de gros problèmes, comme arrêter un déplacement en rencontrant un ennemi,, forçant un deuxième mouvement, ou une répétitions des interactions avec les portes. Si le problème des "alert" a pu être remplacé par la fonction "dbox" (voir annexe "B.3"), qui utilise un div non-display pour réaliser des annonces moins invasives et plus esthétiques, aucune réponse n'a pu être trouvé pour les "confirm" par manque de temps. Cependant, on peut penser que l'utilisation de fonction "async" de Javascript, et ou de l'utilisation de l'objet "Promise" pourrait permettre efficacement pallier ce problème, bien que nous n'ayons eu le temps de maîtriser suffisamment ces éléments pour les utiliser dans notre code.

5.4 Énigmes

Les énigmes catalysent la collaboration et l'intelligence collective, surtout quand elles sont imbriquées, ce qui permet une immersion active aux participants. Leur but est de permettre au joueur de pouvoir s'échapper d'une pièce en les résolvant en un temps imparti, ils peuvent aussi servir à ouvrir une porte, avoir une arme, augmenter la barre de vie de l'avatar .. etc. La première chose que nous faisons, c'est de créer des énigmes (avec leur réponses) et les placer dans un fichier. Dans la partie éditeur, L'interface permet de lier une énigme à une ou plusieurs cases du damier. Pour cela, il suffit d'interagir avec le formulaire accessible depuis le bouton nommé "Ajouter une énigme". Les données saisies sont ensuite stockées dans le sessionStorage. Dès que l'utilisateur clique sur une case du damier, l'énigme préalablement insérée sera affecté à cette case là. En effet, un attribut nommée <énigme> sera ajouté au carré en question. Dans la partie utilisateur, nous avons ajouté une fonction qui, une fois que l'avatar arrivera sur la case, où il y a l'énigme, il visualisera le texte et aura la possibilité d'y répondre.

5.5 Poursuivant

5.5.1 But

Le poursuivant est un agent autonome qui a pour but de suivre l'avatar tout au long de son périple. Il se déplace suivant une structure de graphe et avec un algorithme de parcours en profondeur. Si l'avatar rentre en contact avec lui, il perd des points de vie. Son but est de rendre le jeu un peu plus complexe et d'empêcher le joueur de pouvoir se déplacer à sa guise et de refaire les niveaux en esquivant les pièges dont il se rappellerai de l'emplacement. Le joueur doit idéalement pouvoir bloquer son poursuivant pour lui permettre de s'éloigner de lui. Le poursuivant pourrait prendre 1 tour pour ouvrir la porte ou resterai bloqué.

La difficulté a implémenté cette caractéristique dans le jeu est dans le fait d'avoir un algorithme qui doit pouvoir connaître la position du joueur et de trouver le chemin le plus court.

5.5.2 Structure de graphe

Une structure de graphe est un type abstrait de graphe faisant référence à la théorie des graphes qu'on retrouve en mathématiques. Un graphe est donc une structure composée d'objets représentés par des noeuds, pour lesquels des liens sont établis et représentés par des arêtes. Une heuristique peut être créée pour identifier le coût d'une arrête entre 2 noeuds, et ainsi identifier si possible, quel chemin serait le plus court.

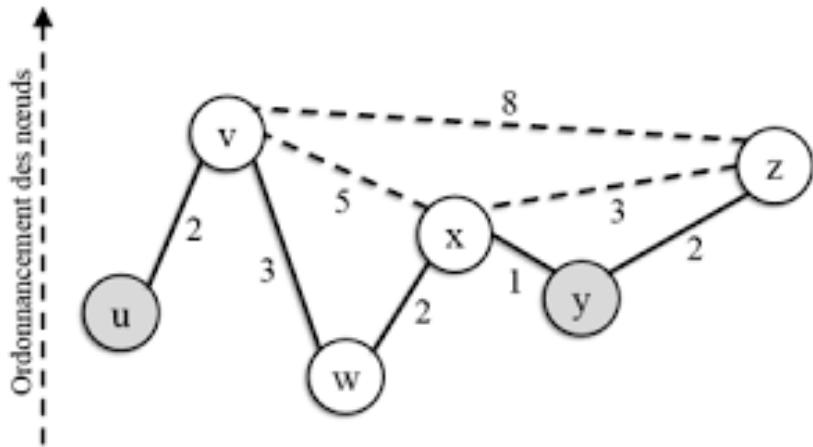


FIGURE 10 – Exemple de Graphe

Dans notre cas, l'utilité d'une structure de graphe fut essentielle pour que le poursuivant puisse se déplacer sur le damier. Implémenté sous un dictionnaire ayant comme clé la liste de coordonnées de la case courante, et en valeur les cases voisines et un entier $n \in [-1, 1]$, qui permet d'identifier si la case voisine correspond à un sol, une porte verrouiller ou ouverte. Il faut noter que le graphe de contient que les cases où le joueur est susceptible d'aller, donc il ne prend en compte ni les murs, ni le vide. Ce format nous permet donc de retrouver tous les déplacements possible sur le damier.

Nous avons une fonction `graphe(data)` (voir annexe C.1) qui prend en paramètres une liste de liste contenant le type de chaque case. Pour chacune de celle-ci, nous allons regarder si le poursuivant peut se déplacer dessus. Dans la positive, on garde cette case courante et ses voisins dans le graphe, en y ajoutant un indice 0. Puis pour chacune de ses cases voisines, on garde les coordonnées et l'indice 0,1 où -1 correspondant . Une fois le damier entier parcouru, on garde le graphe dans une variable globale. Cette fonction est appelé au chargement du niveau, mais aussi lorsque le joueur ferme ou ouvre une porte, pour que le poursuivant puisse prendre en compte ce changement.

5.5.3 Algorithme A* et parcours en profondeur

Dans notre problème de recherche de chemins, plusieurs algorithmes peuvent être utilisés tel que Djikstra , best-first ou encore A*. Nous cherchons un algorithme rapide et efficace d'où le choix de l'algorithme A* qui allie l'efficacité de best first et la rapidité de Djikstra. En effet, best-first essaye d'aller vers le but en utilisant une heuristique, et Djikstra explore l'espace et essaie de trouver le meilleur chemin à partir de ce que l'on a exploré.

Algorithm 1 Pseudo-code de l'algorithme A*

```

1: On ajoute départ dans openList
2: while openList n'est pas vide do
3:   NodeCourant = trouve le plus petit f de OpenList // $f(x) = g(x) + h(x)$  f correspond à la
   somme du cout et du temps total pour atteindre ce noeud
4:   if NodeCourant == Objectif then
5:     Retourne le bon chemin
6:   end if
7:   On ajoute NodeCourant dans closedList
8:   On supprime NodeCourant de openList
9:   for chaque voisin du NodeCourant do
10:    if le voisin n'est pas dans openList then
11:      On garde g, h et f et le parent courant
12:      On ajoute le voisin dans openList
13:    end if
14:    if le voisin est dans openList mais le g courant est meilleur que le g précédent then
15:      On garde g et f et le parent courant
16:    end if
17:   end for
18: end while

```

Le graphe peut se parcourir soit en profondeur, soit en largeur. Dans notre cas, nous préconisons un parcours en profondeur afin de descendre plus rapidement jusqu'à l'objectif et plus facile à implémenté. IL utilise le principe de LIFO pour une pile, où le dernier rentré sera le premier sorti. Malgré qu'un parcours en largeur permettrait d'obtenir le chemin le plus court, mais plus compliqué à concevoir.

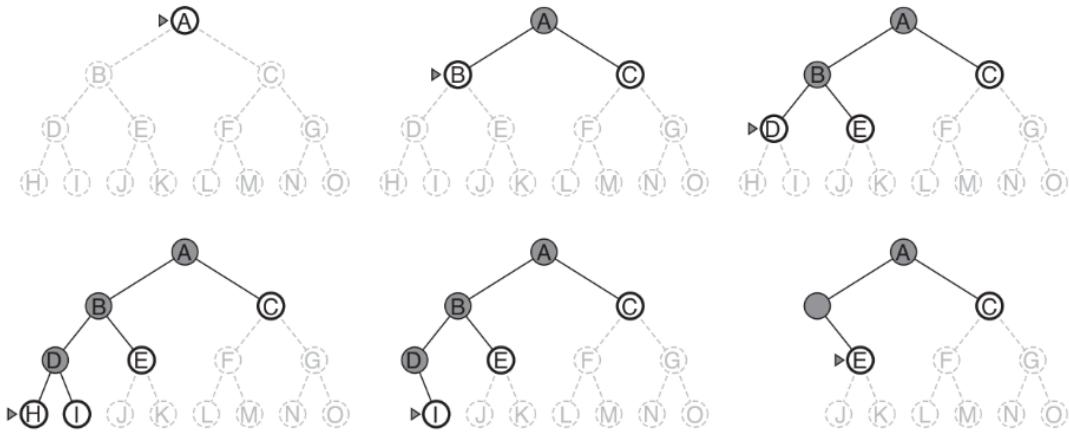


FIGURE 11 – Exemple d'un parcours en profondeur

Nous avons une fonction récursive `depthsearch(position_avatar, position_follower, historique du follower)` qui est appelé à chaque mouvement du joueur. (voir annexe C.2) Celle-ci part de la position initiale du poursuivant et calcule sa distance maximal entre lui-même et le joueur. Pour chacune de ses cases voisines, il calcule la distance jusqu'à l'arrivée et garde les coordonnées qui accède au plus court chemin. Une fois que la case voisine correspond à la position du joueur, la fonction retrace le chemin parcouru et le retourne dans une liste. Une fonction `mouv()` se charge ensuite du déplacement, sachant que le poursuivant ne fera qu'un pas sur deux et s'il touche le joueur, il sera aussi stopper pendant un tour.

5.5.4 Problèmes et améliorations possibles

Finalement, nous pouvons noter que notre algorithme se rapproche d'un algorithme best-first plutôt que A*, vu que l'on prend seulement en compte l'heuristique et pas la longueur du chemin . Idéalement, l'algorithme trouve plusieurs chemins à comparer et en comparant la longueur de chaque chemin, on obtiendrait le meilleur.

Cela peut aussi poser problème quand le poursuivant arrive dans un minimum local, il reste bloqué jusqu'à ce que l'algorithme retrouve le chemin final, après que le joueur ait changé de position. Dans l'exemple suivant, le poursuivant avance vers le chemin le plus court (bleu), mais ne peut pas traverser les murs et reste bloqué, car il prend seulement la case voisine qui le rapproche. Il devrait suivre le chemin vert, pour cela il faudrait rajouter un cas lorsqu'il ne trouve pas de case plus proche. Nous avions pensé initialement à choisir un des voisins aléatoire sans considérer la distance, mais par manque de temps nous n'avons pas pu implémenter cette solution.

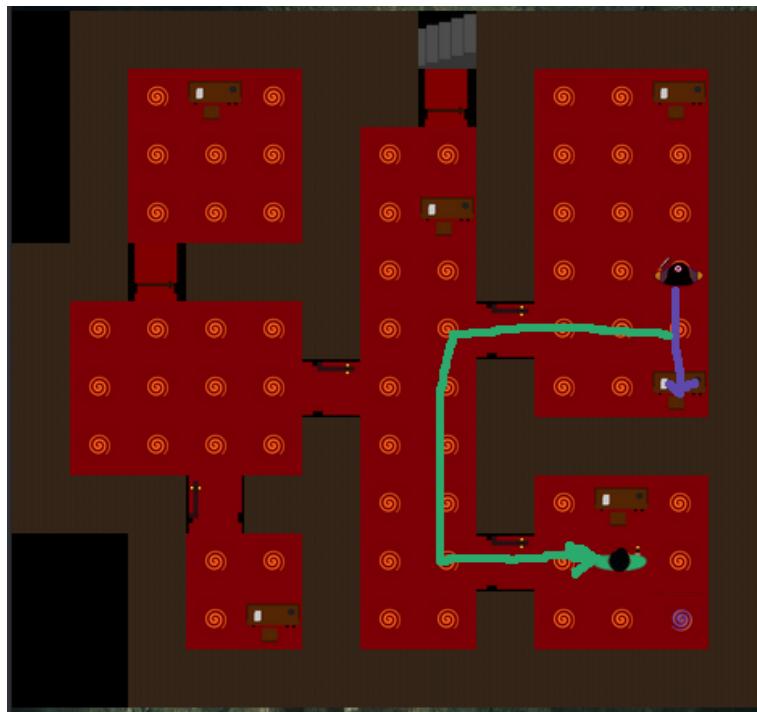


FIGURE 12 – Exemple d'un chemin

Une amélioration notable serait que le poursuivant puisse ouvrir une porte qui aurait été fermée par le joueur. Pour l'instant il reste bloqué, et le joueur peut facilement se déplacer sans le prendre en compte. Pour cela il faudrait récupérer la case qui concerne la porte et changer son indice de la même façon que le joueur ouvre une porte.

6 Conclusion

6.1 Diagramme de Gantt final

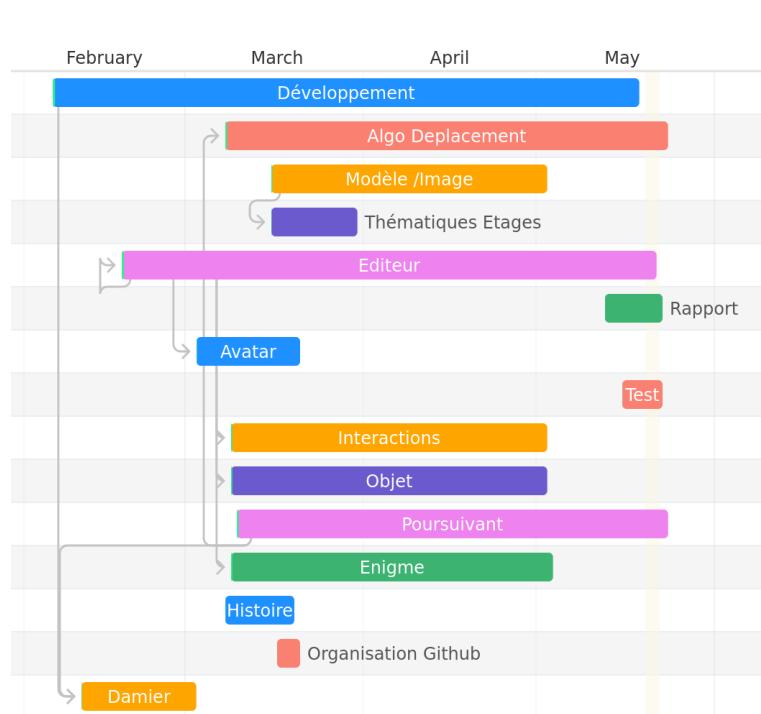


FIGURE 13 – Diagramme de Gantt

En comparaison avec le diagramme de Gantt initial, nous remarquons très facilement que nous avons sous-estimé le temps de complétion de chaque tâche. Une meilleure organisation et délégation des rôles aurait peut-être pu arranger cela.

6.2 Améliorations générales

De nombreuses améliorations globales peuvent être soulevées :

- Le code peut être optimisé dans sa clarté de ses variables qui ne sont pas toujours évidentes. Une mauvaise déclaration des variables peut être amélioré avec une meilleure compréhension entre let, var et const. D'autres déclarations avec le même nom entre variables globales et locales, ont tendance à rendre le code confus. Des boucles et des conditions auraient pu être simplifiées afin de baisser la complexité du code.
- Nous pourrions élaborer une base de données qui stockerait tous les niveaux, ainsi que les données de l'utilisateur pour qu'il puisse sauvegarder sa partie, par exemple.
- Le serveur Node.js pourrait être utilisé davantage en ayant d'autres routes qui afficheraient le menu du jeu, ou encore une page de création d'avatar.
- Un objectif initial était d'implémenter un "fog of war", c'est-à-dire un brouillard afin que le joueur n'ait pas accès à l'entièreté du niveau dès le début, mais seulement au cours de son exploration, ou lors de la découverte d'une carte. L'apport d'un tel élément de jeu est conséquent à bien des niveaux dans ce jeu.
- En ce qui concerne les interactions, un atout principal serait de pouvoir déposer le contenu

-
- de notre inventaire dans un meuble. Le point le plus intéressant à développer semble être l'utilisation des fonctions asynchrone et de l'objet Promise.
- Pour le graphisme, des animations 2d pourraient être mieux envisagées pour les déplacements des personnages comme l'avatar et les ennemis. Cela nécessite d'avoir une bonne vision en trois dimensions. Jouer sur la saturation et choix de couleurs pour une thématique plus riche et avec une bonne qualité. Plusieurs thèmes pour avoir de différents niveaux, pourrait animer mieux notre jeu.

6.3 Apports personnels

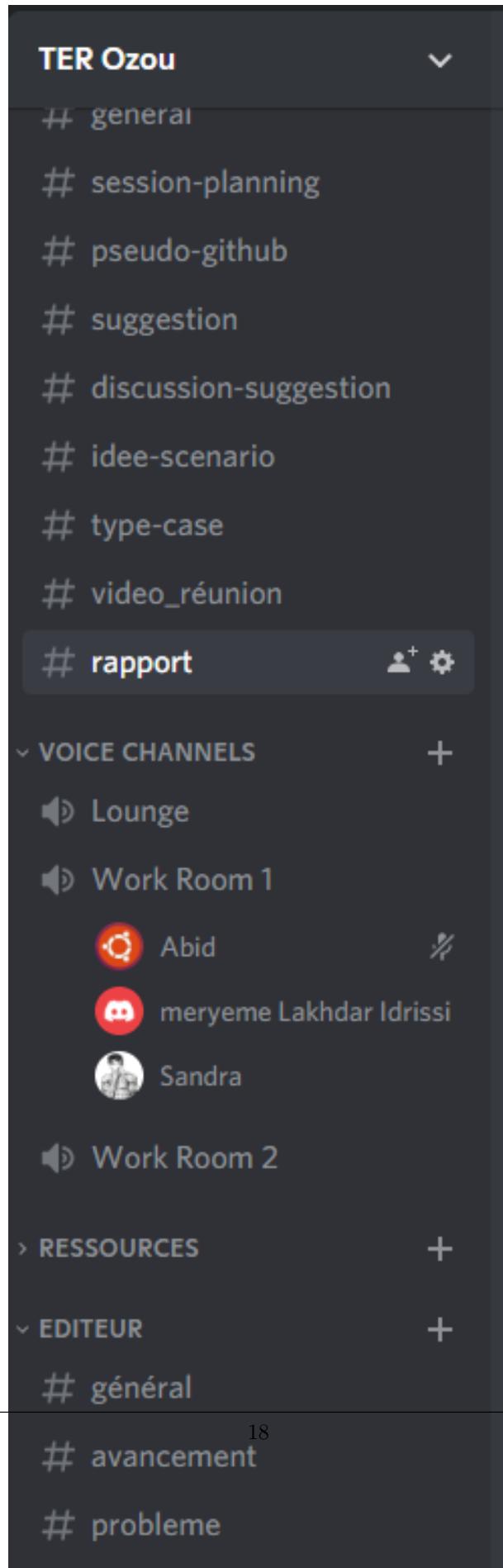
En conclusion, ce projet de TER a apporté des connaissances en terme d'organisations pour tout ce qui est diagrammes, gestion de temps de charge de travail, et de travail d'équipe. Notamment dans l'échange de schéma de réflexion, où il est judicieux de se remettre en question, mais aussi de garder sa ligne de pensée pour ne pas trop diminuer sa productivité.
D'un point de vue plus technique, on a pu découvrir des bibliothèque comme D3.js, mais aussi des outils d'hôtes de développement en logiciel comme github. Le principal apport pour nous s'est fait dans l'approfondissement de nos connaissances en JavaScript, ainsi que notre capacité à développer un projet dans cet environnement.

Références

BSD LICENSE, Mike Bostock (p. d.). *Documentation D3.js*. URL : <https://d3js.org/>.

Annexes

A Outils utilisés



B Gameplay

B.1 Commencement

```
function genereModele(data,rayon, nbLignes, nbColonnes){\n\n    let width = nbColonnes*rayon;\n    let height = nbLignes*rayon;\n    var img = "images/\n    d3.select("#damier").append("svg").attr("width", width).attr("height", height);\n    var svg = d3.select("#damier").select("svg").attr("width", width).attr("height", height);\n    svg.selectAll("image").remove();\n    for(var x=0; x<nbColonnes; x++){\n        var left=-1;\n        var right=-1;\n        var up=-1;\n        var down=-1;\n        for(var y=0; y<nbLignes; y++){\n            var rotate ="";\n\n            if (x>0 && x<nbColonnes-1){\n                left = data[y][x-1].type;\n                right = data[y][x+1].type;\n            }\n            if (y>0 && y<nbLignes-1){\n                up = data[y+1][x].type;\n                down = data[y-1][x].type;\n            }\n            if(data[y][x].type==6 && up==1 && down==1){\n                rotate = 'translate(' +rayon+ "," +0+ ') rotate(90, ' +x*rayon+ ',' +y*rayon+')';\n            }\n            if(data[y][x].type==5 && up==1 && down==1){\n                rotate = 'translate(' +rayon+ "," +0+ ') rotate(90, ' +x*rayon+ ',' +y*rayon+')';\n            }\n            svg.append("image")\n                .attr("id", "D"+y+x)\n                .attr("xlink:href", img+types[data[y][x].type]+".png")\n                .attr("width", rayon)\n                .attr("height", rayon)\n                .attr("x",x*rayon)\n                .attr("y",y*rayon)\n                .attr('transform',rotate);\n            if(data[y][x].type==8){\n                var start = [x*rayon,y*rayon]\n            }\n        }\n    }\n}
```

FIGURE 15 – Fonction genereModele

B.2 Objets et Inventaire

```
// Permet de récupérer les objets provenant du Loot
function getItem(objet,idl){
    let o = objets[objet];
    for(let p=1;p<11;p++){
        let elem = document.getElementById("I"+p);
        if(elem.hasAttribute('objet')){
            console.log("Emplacement "+p+" déjà pris")
        }
        else{
            var x = d3.select("#I"+p).attr("x");
            var y = d3.select("#I"+p).attr("y");
            d3.select("#I"+p).attr('objet',o.nom);
            d3.select("#invent").append("image")
                .attr("id","IM"+p)
                .attr("xlink:href", "images/"+o.nom+".png")
                .attr("width", 100)
                .attr("height", 100)
                .attr("x",x)
                .attr("y",y)
            var id = "#IM"+p;
            if(o.type=='soin'){
                d3.select(id).attr("onclick", "updateHealth("+o.valeur+"), deleteObj(IM"+p+", I"+p+ ")");
            }
            if(o.type == 'aide'){
                d3.select(id).attr("onclick", "aide("+o.nom+")");
            }
            let str=idl.getAttribute("id");
            d3.select(idl).remove();
            let nobj = "objet"+str.charAt(str.length-1);
            delete obj[nobj];
            return console.log("objet placé dans l'inventaire.");
        }
    }
    dbox("Inventaire plein")
}
```

FIGURE 16 – Fonction getItem

```

//Objet de Soin
function objectSoin(type,nom,valeur,image){
    this.type=type
    this.nom=nom
    this.valeur=valeur
    this.image=image
}

var pillules = new objectSoin('soin','pillules',10,'image')
var bandages = new objectSoin('soin','bandage',30,'image')
var kit = new objectSoin('soin','kit',50,'image')

```

FIGURE 17 – Exemple Objets dans fichier Ressources

B.3 Confirm et Alert

```

//(B) - TOGGLE-DIALOG-BOX
function dbox(msg){
    // (B1) - GET-ELEMENTS
    let dbox = document.getElementById("dbox"),
        dboxm = document.getElementById("dboxm");

    // (B2) - SHOW/HIDE
    dboxm.innerHTML = (msg === undefined) ? "" : msg;
    dbox.style.display = (msg === undefined) ? "none" : "block";
}

```

FIGURE 18 – Fonction dbox

C Poursuivant

C.1 Structure de graphe

```
//fonction qui crée une structure de graphe à partir d'une liste de liste contenant les types de cases
function graphe(data){
    let json = {};
    let depart;let droite;let gauche;let haut;let bas;
    let position;
    for(let ligne=0; ligne<data.length; ligne++){
        for(let colonne=0; colonne<data[ligne].length; colonne++){
            if((ligne>0 && ligne<data.length-1) && (colonne>0 && colonne<data[ligne].length-1)){
                depart = data[ligne][colonne];
                droite = data[ligne][colonne+1];
                gauche = data[ligne][colonne-1];
                haut = data[ligne-1][colonne];
                bas = data[ligne+1][colonne];
                if(depart!=1 && depart!=5 && depart!=2 && depart!=9){
                    position=[ligne,colonne];
                    json[position]=[];
                    if(droite!=1){
                        json[position].push([goRight(droite,ligne,colonne),jonction(droite)]);
                    }
                    if(gauche!=1){
                        json[position].push([goLeft(gauche,ligne,colonne),jonction(gauche)]);
                    }
                    if(haut!=1){
                        json[position].push([goUp(haut,ligne,colonne),jonction(haut)]);
                    }
                    if(bas!=1){
                        json[position].push([goDown(bas,ligne,colonne),jonction(bas)]);
                    }
                }
            }
        }
    }
    chemins=[];
    graph=json;
}
```

FIGURE 19 – Fonction qui retourne une structure de graphe

C.2 Algorithme

```
//fonction récursive qui depuis la position du poursuivant cherche l'avatar avec une fonction de coût et parcours le graphe
//si l'algorithme trouve la position d'arrivée elle retourne le chemin
function deepSearch(pos_avatar,pos_follower,hist){
    if(pos_follower==undefined || graph[pos_follower]==undefined){
        console.log("pas de chemin")
    }else {
        let casevoisine = {};
        let distanceMax = Math.sqrt( Math.pow( pos_avatar[0] - pos_follower[0], 2 ) + Math.pow( pos_avatar[1] - pos_follower[1], 2 ) )
        for (let i = 0; i < graph[pos_follower].length; i++) {
            //calcule la distance entre la position actuelle du poursuivant et la position du joueur pour chacune des cases voisines
            let distance = Math.sqrt( Math.pow( pos_avatar[0] - graph[pos_follower][i][0], 2 ) + Math.pow( pos_avatar[1] - graph[pos_follower][i][1], 2 ) )
            //si il trouve une distance inférieure à la distanceMax il garde cette casevoisine
            if (distance < distanceMax) {
                casevoisine[i] = graph[pos_follower][i][0]
                casevoisine[i] = distance
            }else{
                //sinon il ferait un détour pour ses voisins
                //never random(pos_follower)
            }
        }
        //avite de retourner au même endroit en ne gardant que les cases par où il n'est pas déjà passé
        if (!hist.includes(casevoisine[i])) {
            hist.push(casevoisine[i])
        }
        if (JSON.stringify(casevoisine[i]) === JSON.stringify(pos_avatar)) {
            //si il arrive jusqu'à l'avatar il retourne le bon chemin
            chemin=[];
            const iterator = hist.values();
            for(const value of iterator){
                chemin.push(value)
            }
        } else {
            //sinon il continue la recherche à partir de sa nouvelle position
            deepSearch(pos_avatar, casevoisine[i], hist)
        }
        hist.pop()
    }
}
```

FIGURE 20 – Fonction récursive qui retourne le chemin entre la position du poursuivant et celle du joueur

D Editeur

```
17     function appending(rect,height,width,rayon){
18         isObj = false;
19         if (sessionStorage.getItem("type") == null ){
20             var type = sessionStorage.getItem("objet");
21             isObj = true;
22         }
23         else{
24             var type = sessionStorage.getItem("type");
25         }
26
27         let x = d3.select(rect).attr("x")
28         var y = d3.select(rect).attr("y")
29         var svg = d3.select("#dmier").select("svg").attr("width", width).attr("height", height);
30         var img_toadd = svg.append("image")
31             .attr("xlink:href", img+type+".png")
32             .attr("width", rayon)
33             .attr("height", rayon)
34             .attr("x",x)
35             .attr("y",y)
36             .attr("isObjet",isObj)
37             .attr("id",[x/rayon,y/rayon])
38             .attr("name",[x/rayon,y/rayon])
39             .attr("type",types.indexOf(type))
40             .on("click",function(){
41                 if(sessionStorage.getItem('action') == 'add_enigme'){
42                     let id_rect = "*"+x/rayon+"*"+y/rayon+"*";
43                     let tmp_rect = document.getElementById(id_rect);
44                     tmp_rect.setAttribute('enigme',sessionStorage.getItem('enigme'));
45                     sessionStorage.removeItem('action');
46                     alert('OK!');
47                 }
48                 else{
49                     appendImg(this,height,width,rayon);
50                     appendattr(this)
51                 }
52             })
53     })
54 }
```

FIGURE 21 – fonction d'ajout des images

```

63 function genereDamier(rayon, nbLignes, nbColonnes) {
64     let width = nbColonnes*rayon;
65     let height = nbLignes*rayon;
66     d3.select("#damier").append("svg").attr("width", width).attr("height", height);
67     for(var x=0; x<nbColonnes; y++){
68         for(var y=0; y<nbLignes; y++){
69             let rect = d3.select("svg")
70                 .append("rect");
71
72             rect.attr("class","board")
73                 .attr("x", rayon*x)
74                 .attr("y", rayon*y)
75                 .attr("id", x+','+y)
76                 .attr("width", rayon)
77                 .attr("height", rayon)
78                 .style("fill", "white")
79                 .style("stroke", "black")
80                 .attr("type", "mur")
81                 //.attr("id", [x,y])
82                 .on ("click", function() {
83                     /*
84                     le damier est générée carré par carré, chaque carré contient
85                     des attributs.
86                     Dès qu'un utilisateur clique sur un carré, la fonction JS
87                     ci-dessous est exécuté.
88                     */
89                     console.log(sessionStorage)
90                     /*
91                     Cette fonction JS va vérifié ce est stocké dans sessionStorage
92                     pour savoir qu'était la dernière action : ajout d'un énigme ou bien d'une image
93                     ou bien aucune quand la page est chargée au début.
94                     */
95                     if (sessionStorage.length == 0) {
96                         return;
97                     }
98                     else if(sessionStorage.getItem('action') == 'add_enigme'){
99                         /*
100                         S'il s'agit de l'ajout d'un énigme, je vais tout simplement ajouté ce qui
101                         est stocké dans le sessionStorage préalablement et le mettre
102                         dans un attribut.
103                         Il faut savoir que ce qui est stocké dans le sessionStorage est un objet JSON contenant
104                         la question et réponse.
105                         */
106                         rect.attr('enigme',sessionStorage.getItem('enigme'));
107                         sessionStorage.removeItem('action');
108                         alert('OK!');
109
110                         /*
111                         sessionStorage.removeItem('action');
112                         */
113                     }
114                     else{
115                         /*
116                         // Sinon, ajouter une image
117                         // la récursivité ici est nécessaire pour l'ajout d'un listener onclick
118                         // sur le rectangle ainsi que tout les images inserées ausein du rectangle.
119                         appendImg(this,height,width,rayon);
120                     });
121                 }
122             }
123         }
124     }
125 }
```

FIGURE 22 – fonction GenereDamier

E Enigmes

```

217 function ajouter_enigme() {
218     let question = document.getElementById('question').value;
219     let reponse = document.getElementById('reponse').value;
220     sessionStorage.setItem("enigme",JSON.stringify({
221         "question":question,
222         "reponse": reponse
223     }));
224     sessionStorage.setItem('action', 'add_enigme')
225 }
226 }
```

FIGURE 23 – fonction d'ajout des enigmes

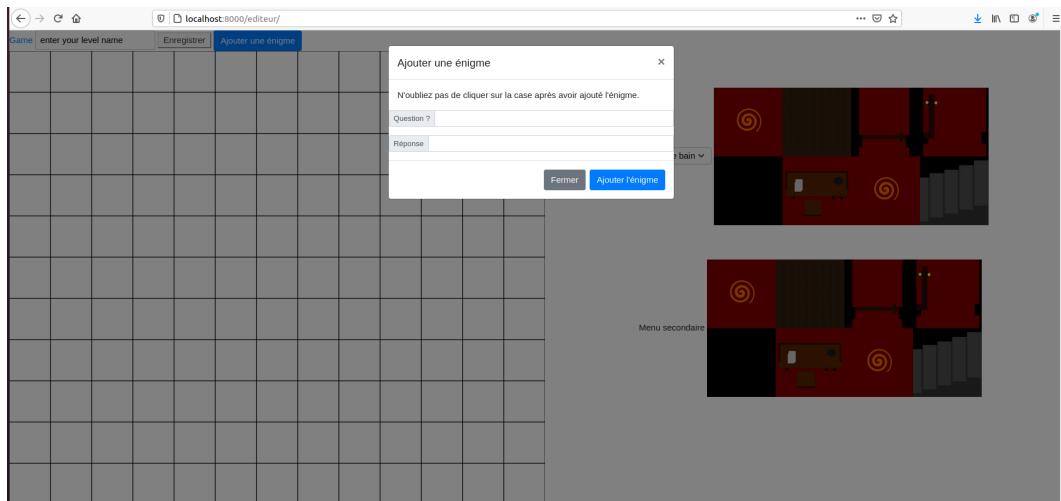


FIGURE 24 – Enigme coté Editeur