

**SOUTENANCE**

**PROJET WEB**

**MIAGE GAMES**

**DELMARE Thomas**

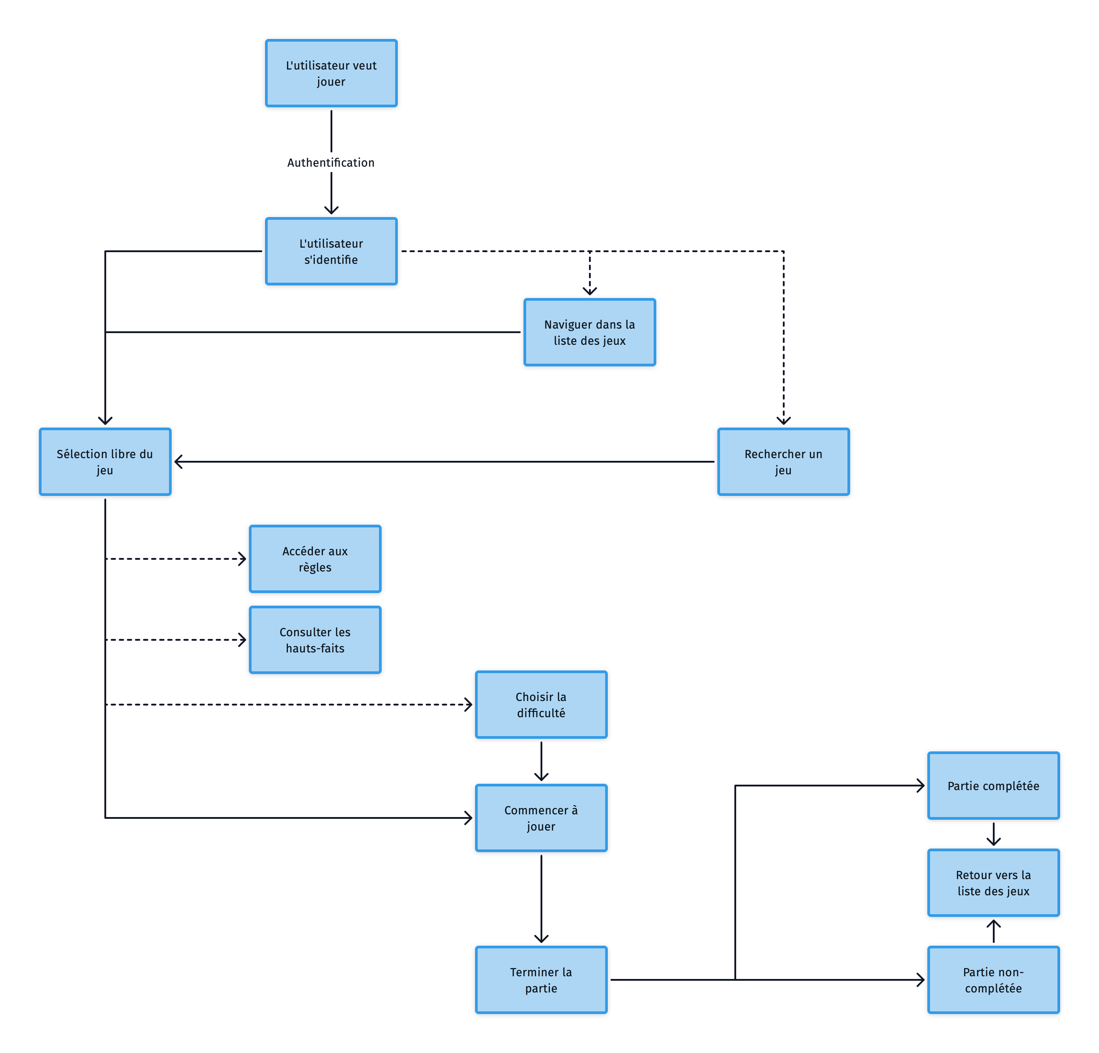
**CHAINTREUIL Benjamin**



**Etablissement / Formation :** Université Côte d’Azur / DS4H / Licence MIASHS / MIAGE – Promotion John Mac Carthy 2020/2023

***Sophia Antipolis, vendredi 19 mars 2021***

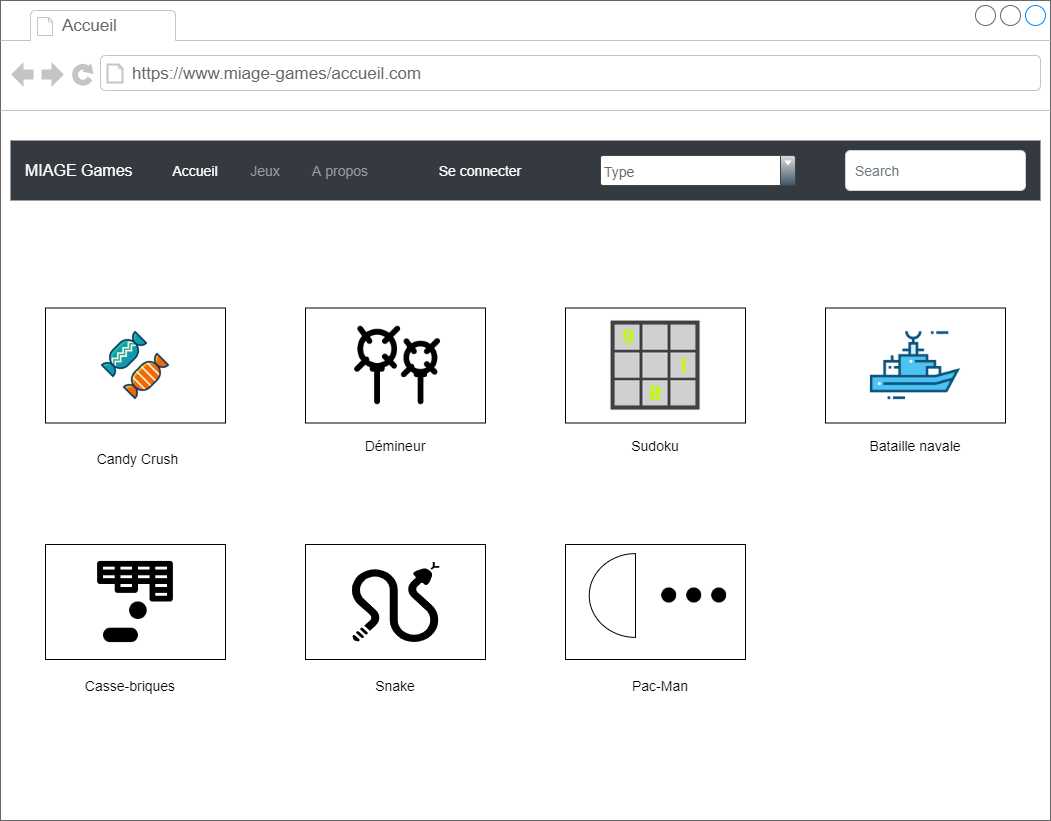
Modèle des tâches



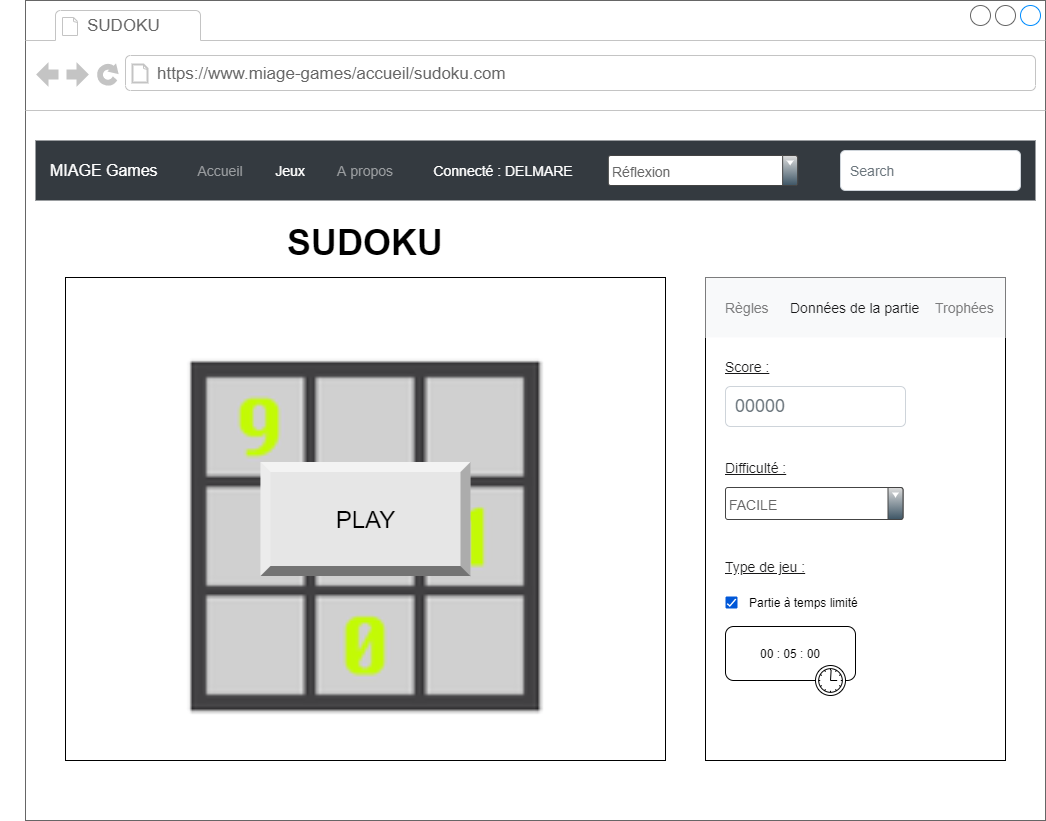
Maquettes prévues

Nous avons fait 4 maquettes de notre site lors du début de notre projet :

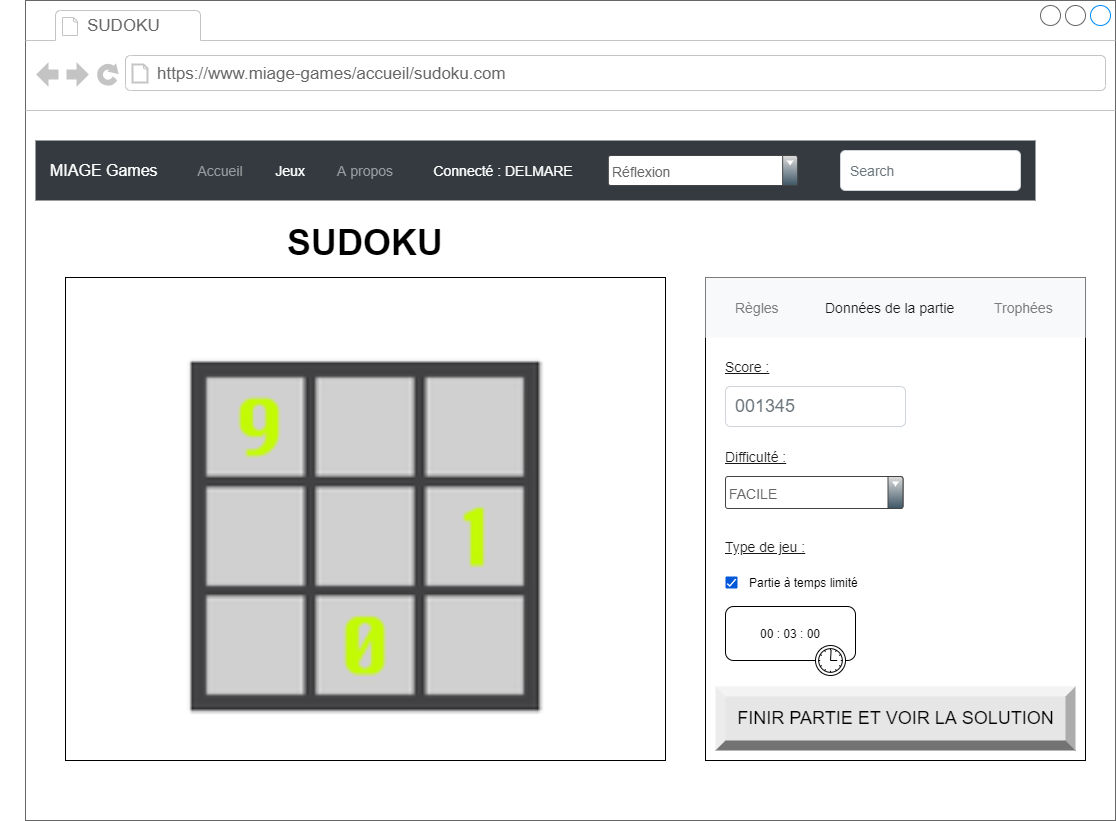
* Une maquette pour la page d’accueil du site



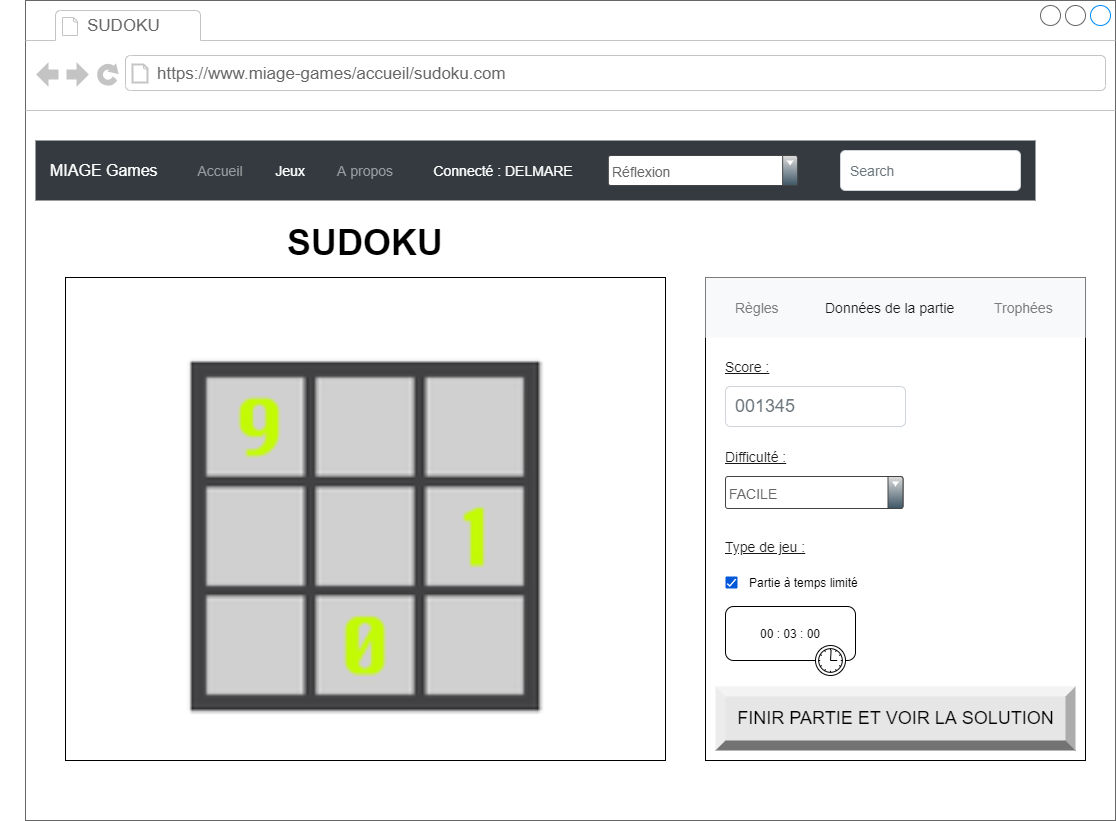
* Une maquette pour la page de jeu lorsque l’utilisateur arrive dessus



* Une maquette lorsque l’utilisateur est en cours de partie sur le jeu



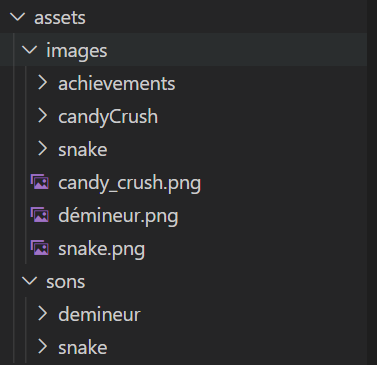
* Une maquette pour la fin de jeu



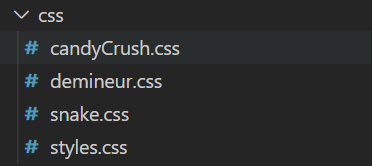
Architecture logicielle

L’organisation des fichiers dans notre projet est la suivante :

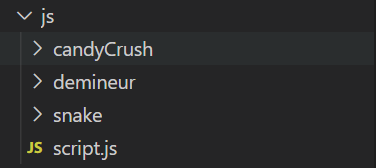
* Nous stockons tous les assets dans un dossier du même nom comportant les images, les sons et toutes autres ressources nécessaires au bon fonctionnement des jeux.



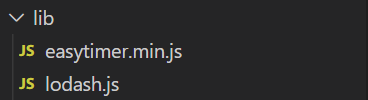
* Nous avons ensuite un dossier « css » comportant un fichier css par jeu ainsi qu’un fichier css général aux pages du site.



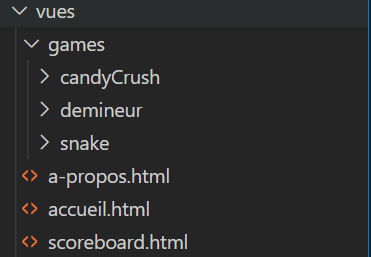
* Nous avons ensuite un dossier « js » comportant l’ensemble des scripts javascript utilisé par le site et les jeux.



* Nous avons un dossier « lib » contenant les librairies utilisées. Les librairies présentent dans ce dossier sont « lodash.jss », étant utilisé pour le début de développement du jeu Candy Crush et « easytimer.min.js » utilisé par le jeu du démineur pour faciliter l’intégration d’un timer.



* Nous avons un dossier « vues » comportant un dossier « games ». C’est ici que sera stocké les différents fichiers HTML utilisés par chaque jeu. Dans la racine de ce dossier, nous avons également 3 fichiers distincts ne concernant pas les jeux, « a-propos.html » concernant le code HTML de la page « à propos », « accueil.html » concernant l’accueil du site et « scoreboard.html » concernant le tableau de scores.



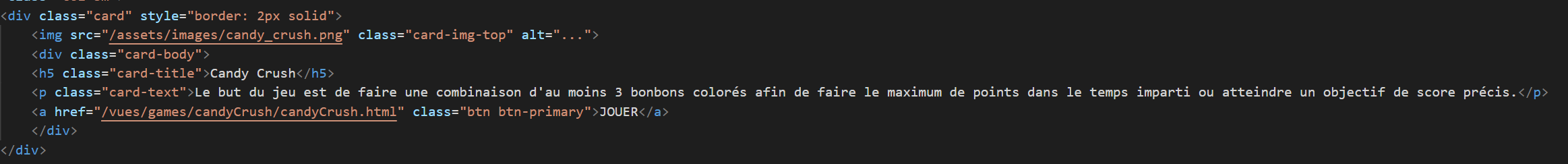
* Finalement, le fichier « index.html », se trouvant à la racine du projet et étant le fichier de démarrage du projet. Ce fichier correspond au formulaire de connexion du site.

Fonctionnement de notre site :

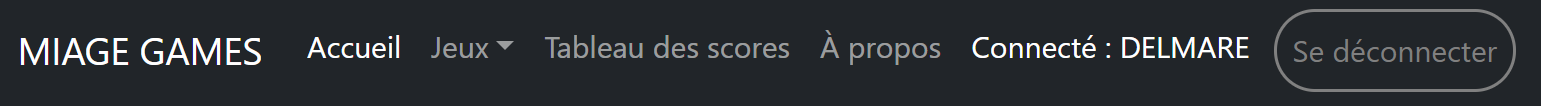
* Lorsque l’utilisateur arrive sur notre site, plusieurs choses se passent, la fonction « localStorageBuild() » est lancée et permet de créer deux utilisateurs de base (DELMARE et CHAINTREUIL) si ces derniers ne sont pas créées. Cette fonction affecte également la variable « activeUser » permettant de savoir quel compte est actuellement connecté. Cette variable permettra d’afficher et affecter d’autres variables sur l’intégralité du site. Lors du lancement de la page, si des utilisateurs sont créées, on leur affecte également des trophées grâce à la fonction « affecterTrophees() », chaque utilisateur se fera affecter 4 succès, en localStorage, 2 succès pour le jeu démineur et 2 pour le jeu snake. Les valeurs de ces clés seront à false. Lors de l’arrivée sur le site, si les variables de highscore ne sont pas implémentées, elles le sont, pour chaque difficulté de chaque jeu, une clé « nom » et une clé « score » sont ajoutées dans le localStorage.
* Lorsque la page est chargée, l’utilisateur peut alors choisir de se connecter ou de se créer un compte. Pour se faire, il doit remplir les deux cases lui étant proposés, un identifiant et un mot de passe. Lorsque l’utilisateur valide la connexion, la fonction « connexion() » est lancée, cette dernière récupèrera les valeurs entrées par l’utilisateur, vérifiera si les champs sont vides, si le formulaire envoyé permet de connecter l’utilisateur, si l’identifiant rentré est bon mais pas le mot de passe ou si l’identifiant n’est pas présent dans la base de données. Dans chacun des cas cités, une notification navigateur apparaitra, lui indiquant dans quel cas il se trouve.
* Si l’utilisateur veut se créer un compte, il pourra le créer grâce à un bouton, lançant la fonction « creerCompte() ». Cette fonction récupèrera les données entrées par l’utilisateur, vérifiera si le compte existe déjà, si l’utilisateur essaye de créer un compte avec des identifiants ou un mot de passe vide ou si le compte a été créé correctement. Une notification apparaîtra pour le prévenir dans quel cas il se trouve. Le formulaire est alors réinitialisé et l’utilisateur peut maintenant se connecter. Une fois connectée, la clé « activeUser » du localStorage est affectée avec l’identifiant entré.



* Lors de son arrivée sur l’accueil du site, il peut choisir de se diriger sur chacun des jeux du site, démineur, snake ou Candy Crush. Il sera redirigé grâce à une propriété href affecté à chaque bouton.

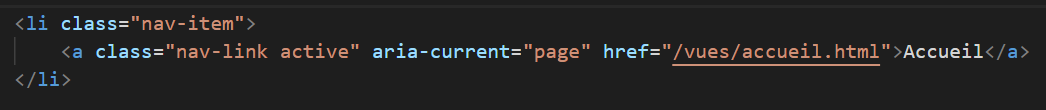


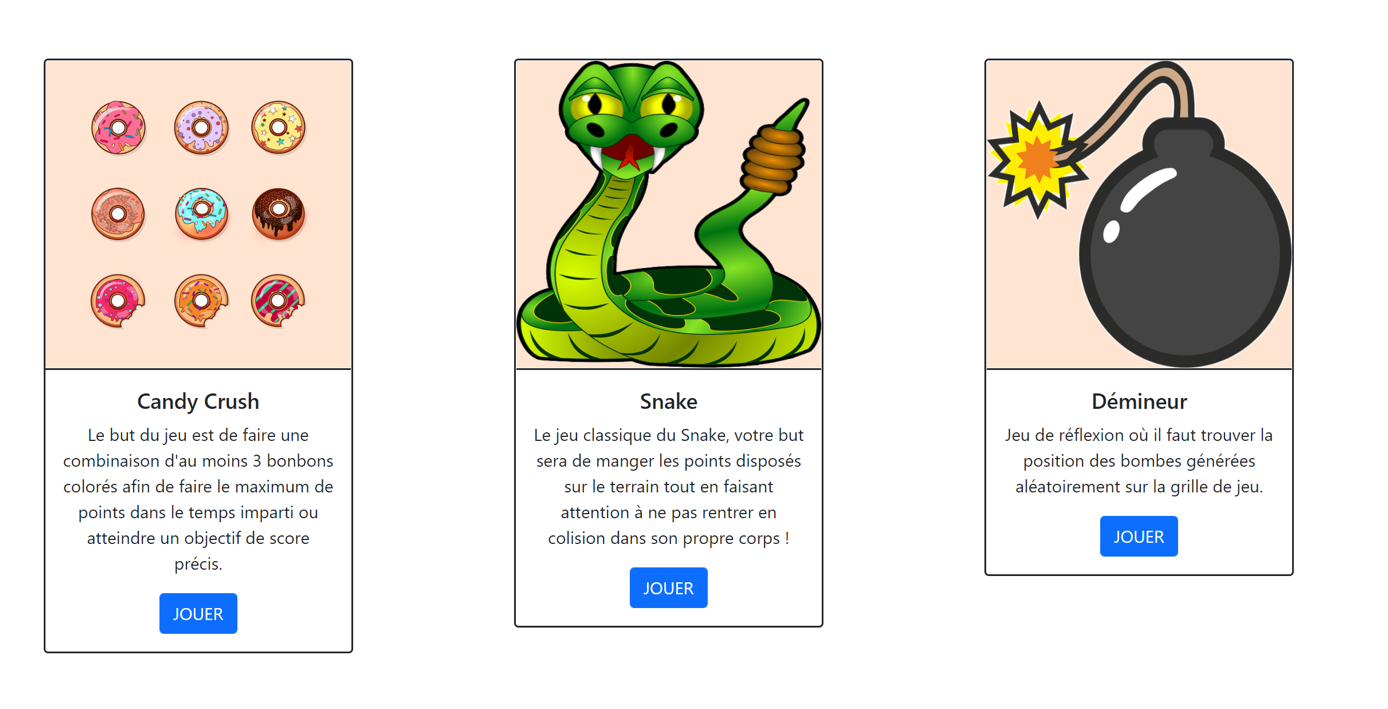
* Nous avons une barre de navigation, l’utilisateur pourra l’utiliser afin d’accéder aux différentes fonctionnalités du site.



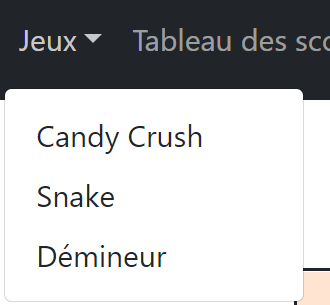
* En cliquant sur le logo « MIAGE GAMES » ou « Accueil » l’utilisateur sera redirigé sur la page d’accueil comportant tous les jeux.

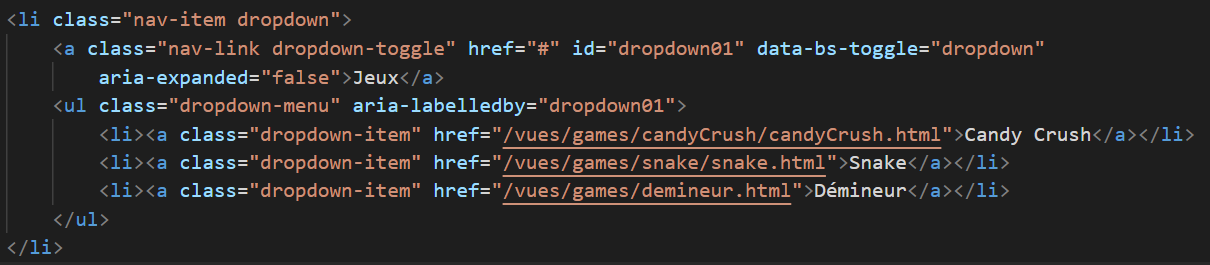




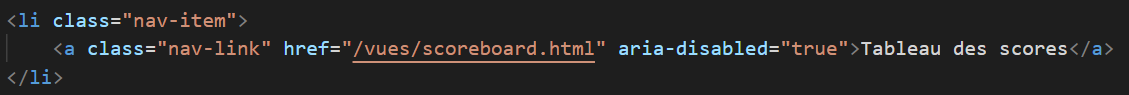


* Il peut également choisir de lancer le jeu grâce à la liste de jeux placée sur la barre de navigation en haut du site. Lors du click, on sera redirigé sur la page HTML du jeu.

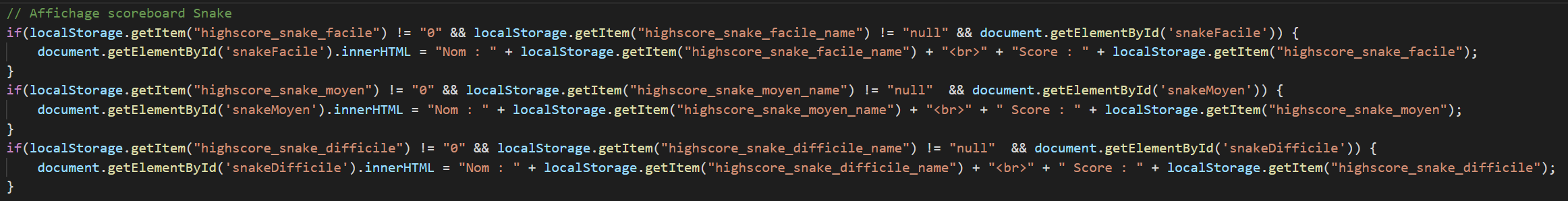




* Il peut également choisir de cliquer sur « Tableau des scores » grâce à la barre de navigation, en cliquant dessus, il accédera au tableau des scores.


* Si les variables instaurés dans le localStorage ont un nom et un score pour une difficulté, elles sont affichées grâce à la fonction « afficherScoreboard() ».

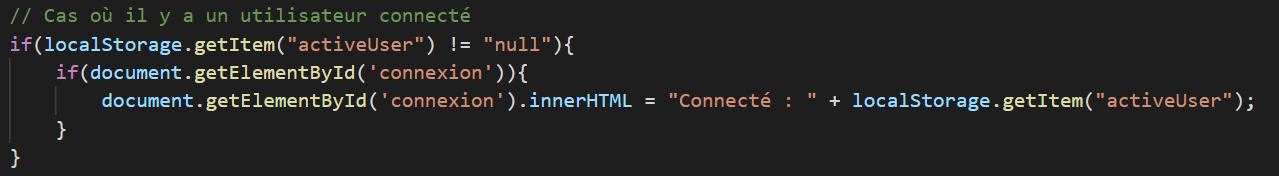


* En cliquant sur « A propos », il sera redirigé sur la page d’information de notre projet.

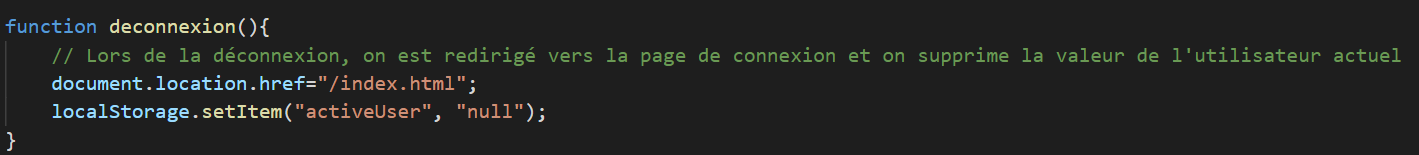




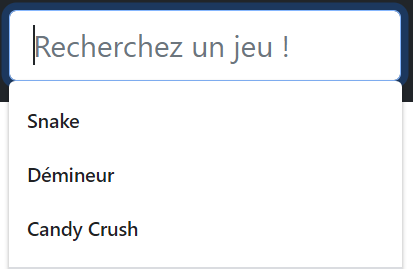
* En appuyant sur « ici », l’utilisateur sera redirigé vers le repository Github de notre projet.
* Un label est également affiché, « Connecté : » permettant de voir l’utilisateur qui est connecté, une partie de la fonction « localStorageBuild() » permet de l’afficher.

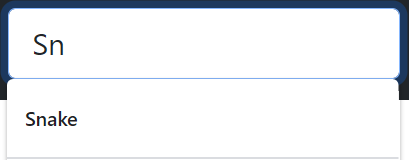


* Le bouton « Se déconnecter » lancera la fonction « deconnexion() » qui permettra d’être redirigé sur le formulaire de connexion et de réinitialiser la valeur de l’« activeUser ».

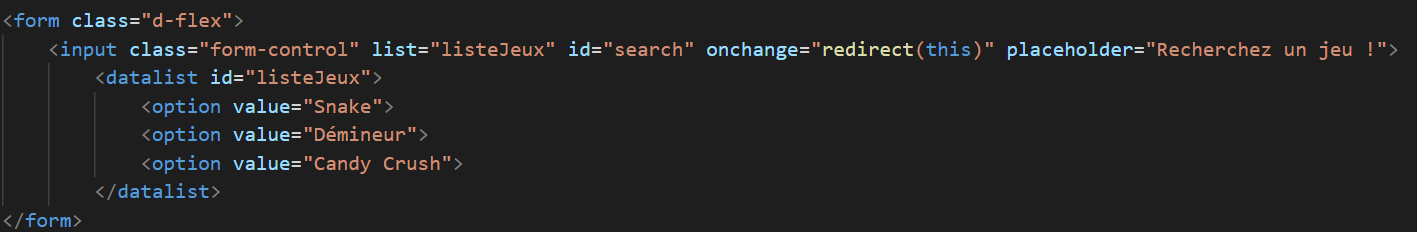


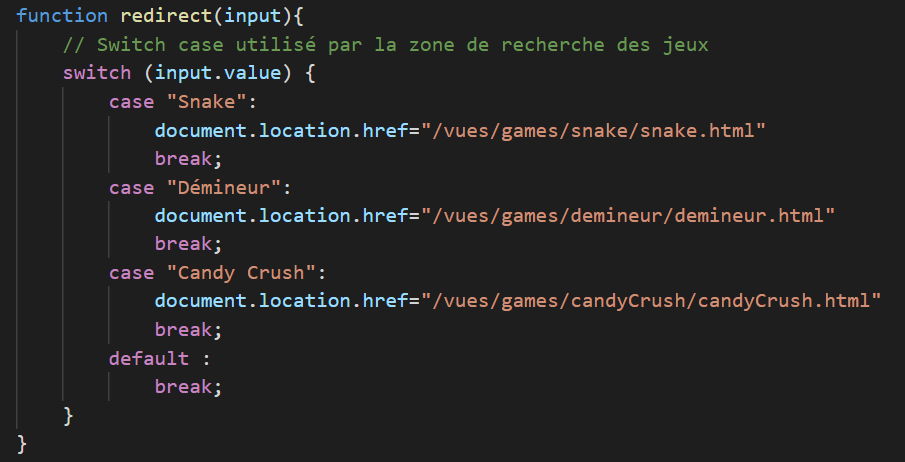
* Enfin, l’utilisateur pourra faire une recherche grâce à une zone de recherche placée à la droite de la barre de navigation.





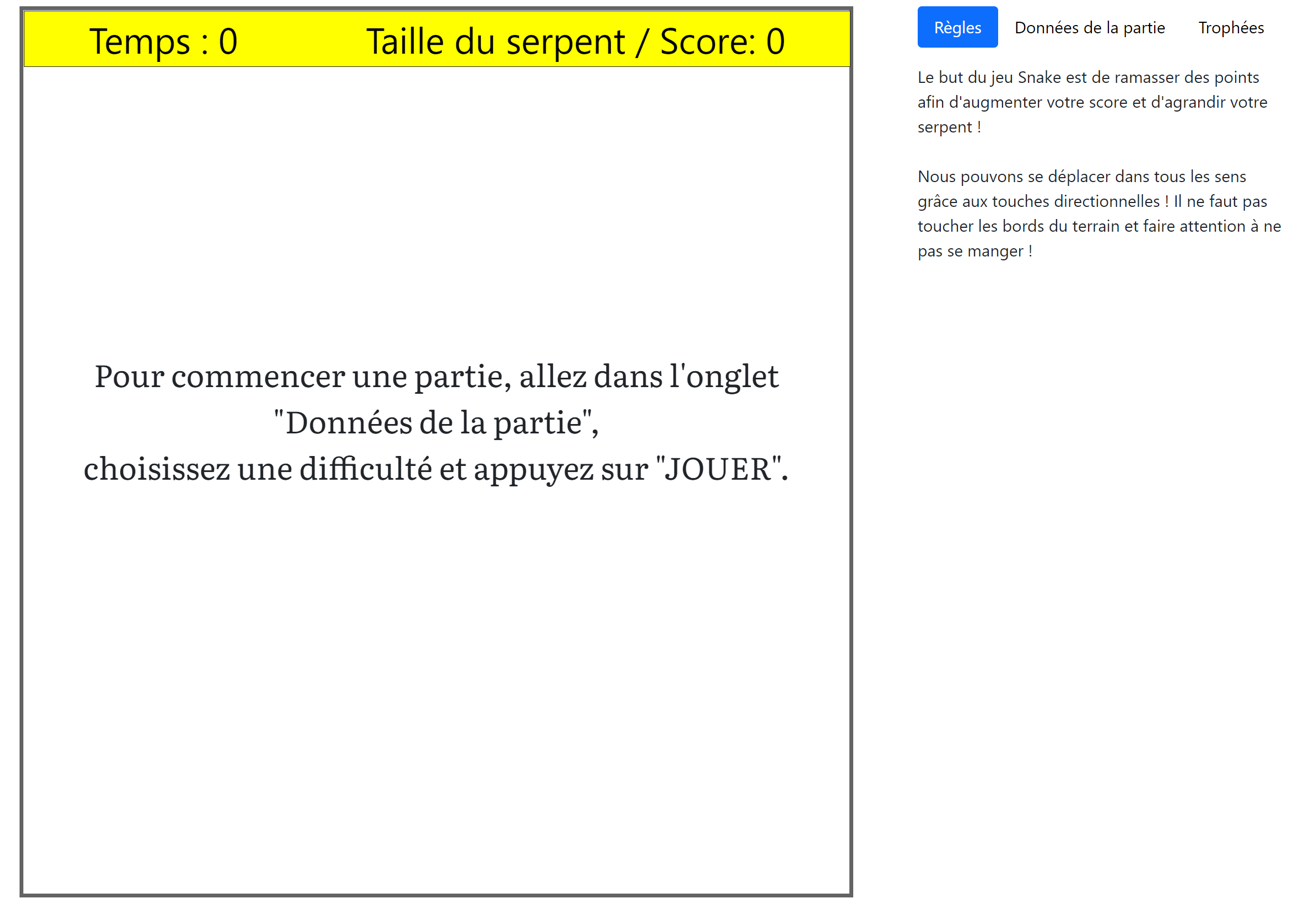
* Si l’utilisateur clique sur l’un des choix, la fonction « redirect(this) » permettra de rediriger l’utilisateur sur la bonne page. « this » comporte l’élément form control ainsi que la value que l’utilisateur entre.



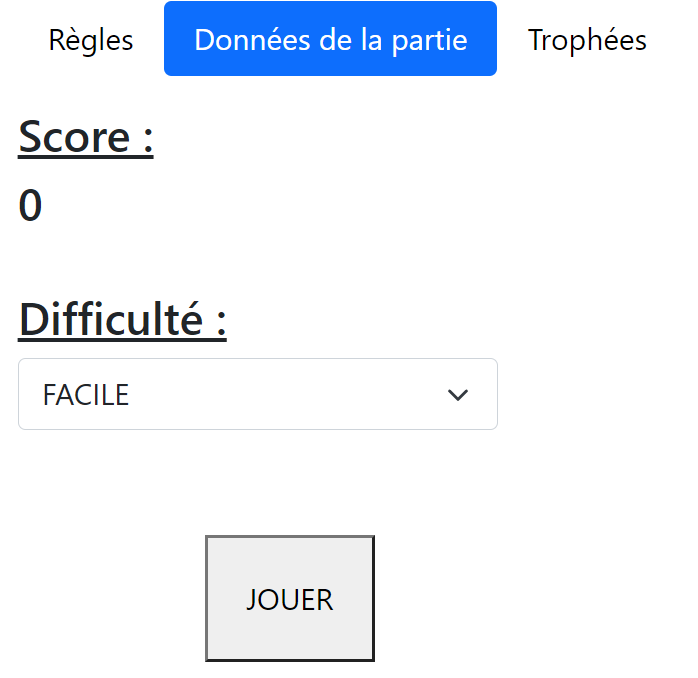
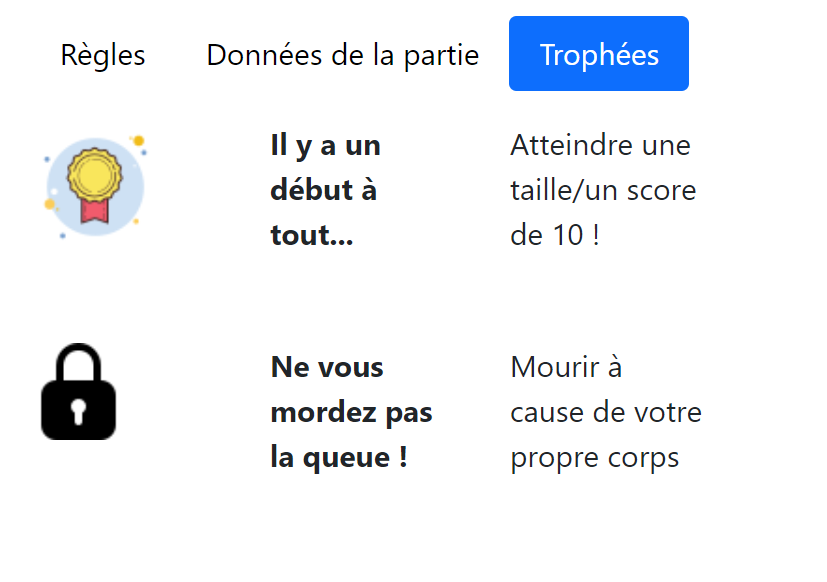


## Fonctionnement du jeu Snake

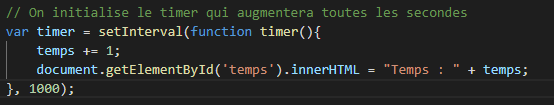
Dans le cas où l’utilisateur veut accéder au jeu Snake, il arrive sur cette vue. Tant que l’utilisateur n’est pas allé dans les données de la partie et a appuyé sur le bouton « JOUER », le jeu ne se lance pas.



Les données de la partie et les trophées sont également accessibles grâce aux onglets à droite du jeu :

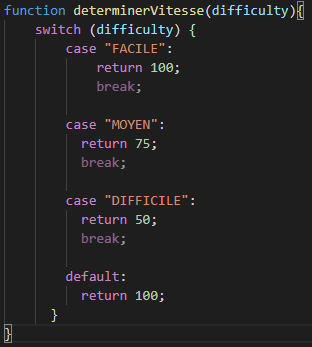


Lorsque l’utilisateur appuie sur le bouton « JOUER », la fonction « initSnakeGame() » est lancée. Cette dernière enlèvera l’affichage d’attente de lancement de partie (en sélectionnant l’élément html et en utilisant sa fonction « remove() », affichera le canvas de jeu (en modifiant la propriété display du canvas en « initial »), créera le snake (grâce à la fonction « createSnake() » qui créera un array de 3 données à deux dimensions). Ensuite, elle réinitialisera le score à 0 ainsi que le timer grâce aux propriétés « innerHTML » des id concernant le score et le timer. La fonction récupèrera, dans une valeur, le canvas (querySelector(#snakeCanvas)) et dans une autre son context (getContext(‘2d’)). Elle initialise le timer et rafraichit son affichage grâce à un interval se rafraichissant toutes les secondes.

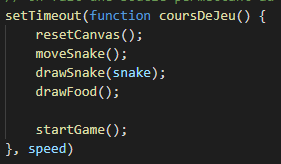


On récupère au préalable l’id de ce timer afin de pouvoir arrêter ce timer plus tard.

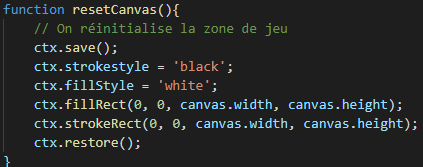
La fonction « startGame() » est alors lancée. Cette dernière récupère la difficulté sélectionnée par l’utilisateur grâce à l’Id du sélecteur de difficulté et la fonction « déterminerVitesse(difficulty) » permettra de déterminer la vitesse du snake.



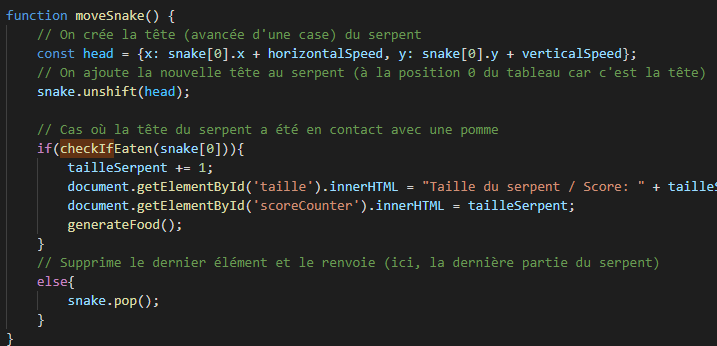
Ensuite, un timeout est paramétré avec la vitesse désirée :



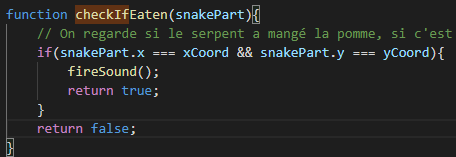
La fonction « resetCanvas() » permettra de pouvoir réinitialiser le dessin fait par la fonction « moveSnake() » utilisée juste après. Sans l’utilisation de cette fonction, les parties déplacées du snake resteront derrière elles, nous aurons une impression que le snake grandit alors que non.



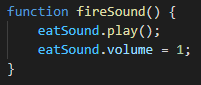
La fonction « moveSnake() » créera une nouvelle tête au serpent, à une case au-dessus de l’actuel. Si la tête du serpent touche une pomme (on le vérifie grâce à la function « checkIfEaten(snakePart) »), la taille et le score sont augmentés et le dernier élément est gardé. Un nouveau fruit est généré grâce à la méthode « generateFood() ». Si le snake n’a pas mangé de pomme, ce dernier aura une nouvelle tête mais la dernière partie sera enlevée (grâce à la fonction « pop() », enlevant le dernier élément d’un tableau, ici le tableau généré par le snake et sa fonction createSnake()).



La fonction « checkIfEaten(snakePart) » comparera la position actuelle de la tête du serpent et jouera un audio grâce à la fonction « fireSound() » si les coordonnées de la tête du serpent et du fruit sont les mêmes. Si le serpent a mangé, la fonction retourne « true » sinon « false ».

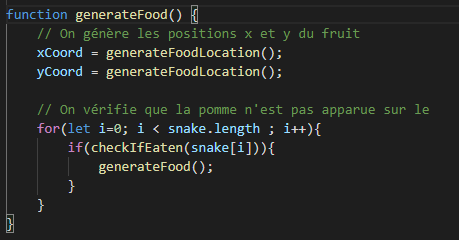


« fireSound() » jouera un objet audio préalablement crée dans le script du jeu.

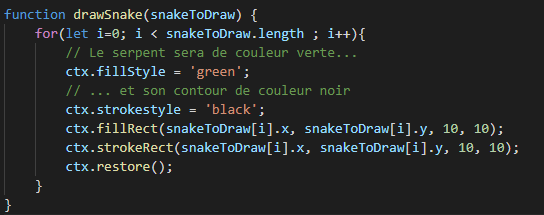




La fonction « generateFood() » affectera des coordonnées x et y (elles-mêmes générées par la fonction « generateFoodLocation() » qui généra des coordonnées comprises dans la taille du Canvas). On vérifie après grâce à la fonction « checkIfEaten(snakePart) » dont on a parlé plus tôt)



La fonction « startGame() » continue alors et lance la fonction « drawSnake(snakeToDraw) » qui fera une boucle sur le tableau stockant les coordonnées du snake et les dessinera sur la grille grâce aux différentes fonctions liées au contexte.



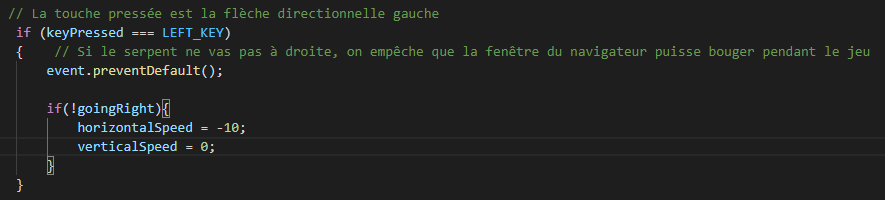
La fonction « drawFood() » dessinera elle le fruit généré avec les coordonnées générées au préalable.

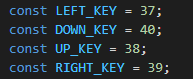


Dans le cas où l’utilisateur perd, la fonction « gameEnd() » mettra fin à la partie. La fin de partie se passe lorsque le serpent touche une partie de son corps ou les bords du canvas. Nous vérifions ça grâce aux coordonnées de la tête du serpent. Si l’utilisateur a perdu, la fonction « showGameOverScreen() » est lancée. On vérifie si le succès 1 est obtenu/si il a été obtenu auparavant et si le score lui permet, d’afficher le succès. Si le score de partie est plus grand que le highscore actuel, on l’affecte grâce aux fonctions « setItem(key, value) » et « getItem(key) » du localStorage. On arrête le timer grâce à la fonction clearInterval(int). On enlève le canvas de jeu en modifiant sa propriété display et on ajoute sur le canvas un élément HTML pour annoncer la fin de partie dont on lui donne un Id afin de pouvoir l’enlever plus tard. On change le libellé du bouton dans « données de la partie » pour ‘rejouer’.

Lorsque la partie sera relancée, l’Id crée pour l’écran de « Game Over » sera récupérée afin de pouvoir supprimer ce dernier de l’affichage.

A l’initialisation du code, un listener a été ajouté sur la page afin de pouvoir gérer les moments où l’utilisateur voudra changer la direction du serpent. L’utilisateur ne pourra pas aller au sens opposé où il va avec une seule touche et l’action par défaut des flèches directionnelles sont annulés (afin d’éviter que la page bouge pendant le jeu par exemple, « event.preventDefault() »). Des constantes avec le code des touches directionnelles ont été créé au préalable lors de l’initialisation du script.





## Fonctionnement du jeu démineur :

**Une image contenant table

Description générée automatiquement**

Le code du jeu s’organise autour de deux classes principales :

* Une classe Minesweeper : représentant le jeu dans son ensemble, avec les méthodes qui y sont associées
* Une classe Cell : représentant une cellule du jeu

Je vais détailler l’ensemble des fonctions de la classe Minesweeper :

🡪 Lorsque le jeu se lance, une instance de ce dernier est créée et assignée à une variable globale nommée « game »

🡪 Le constructeur de la classe se lance et initialise les variables de l’instance du jeu

🡪 Le constructeur appelle la fonction « init() » chargé de l’initialisation du jeu, cette dernière va notamment charger l’ensemble des assets audio utilisés par ce dernier

🡪 Trois fonctions principales sont alors appelées :

🡪 « create2DArray() » Fonction retournant un 2D-array de taille (row, col) contenant les objets Cell composant la grille du jeu

🡪 « populateGrid() » Fonction s’occupant de la génération des Cell et de la population aléatoire des mines

🡪 « draw() » Fonction dessinant la grille

Planning de réalisation des scénarios

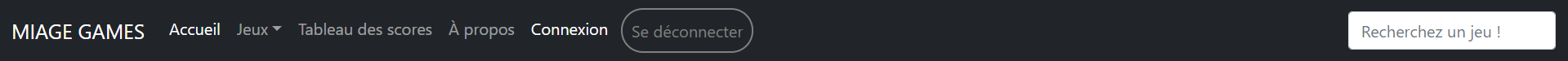
Nous avons détaillé notre projet en 6 itérations distinctes :

* La première itération où on a fait les templates de chaque partie du site en fidélisant au mieux aux maquettes prévues. On avait donc deux templates, un template pour les pages de jeux et un template de page du site basique.

Le template pour la page de jeu ressemblera à ça :



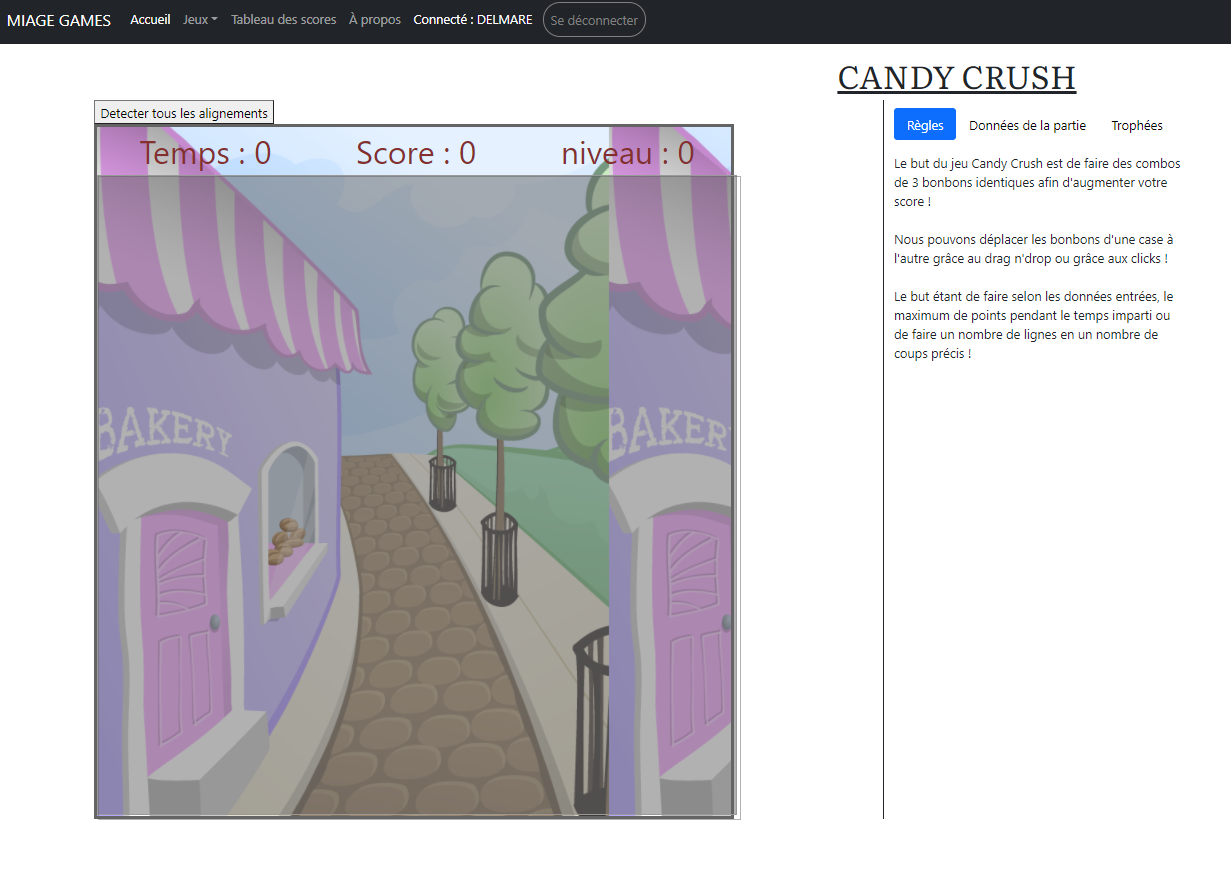
Le template pour la page générale de site comprendra uniquement la navbar qui se trouve tout en haut du site.

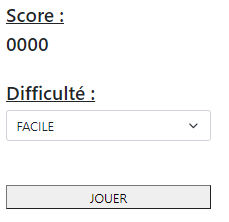
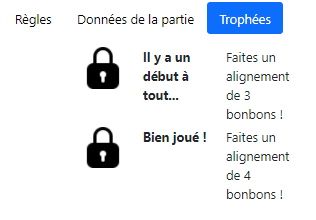


* La deuxième itération est l’ajout du premier jeu, le Candy Crush (jeu type match-3) que l’on a récupéré du projet que l’on avait commencé avec monsieur BUFFA.

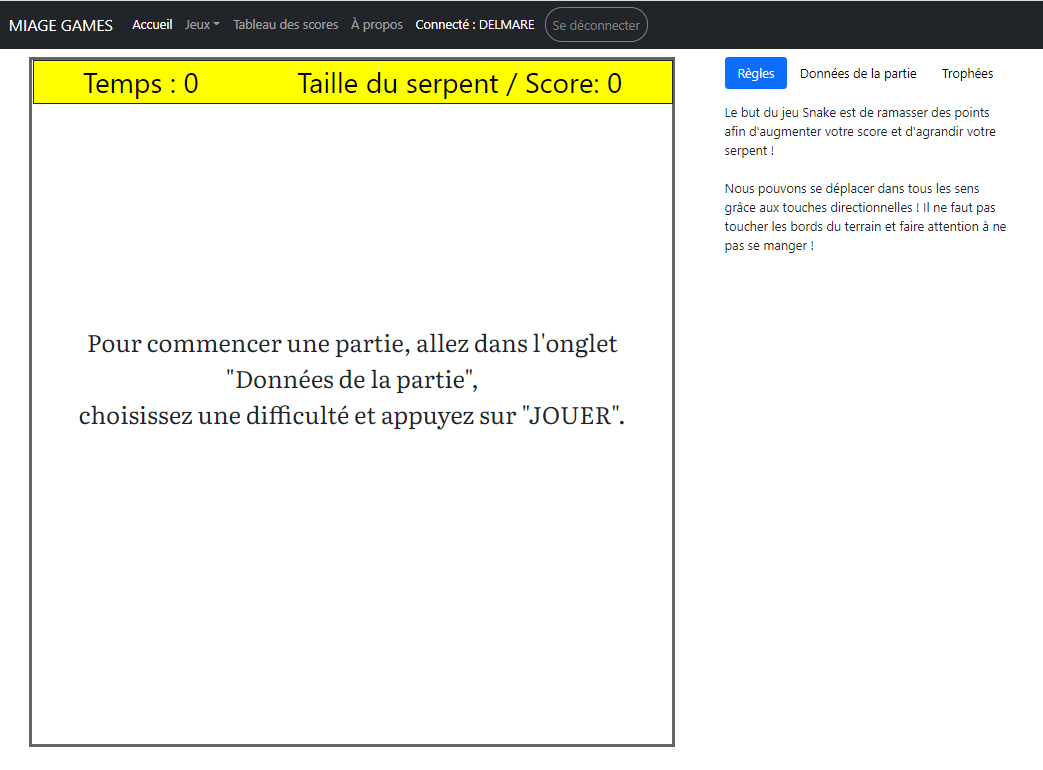
Nous l’avons donc intégré au site ainsi que toutes les données liées au jeu (données de la partie, règles et trophées, uniquement graphiquement).

La finalisation et l’implémentation des succès et données de la partie n’a finalement pas été fait, nos objectifs étaient trop ambitieux.

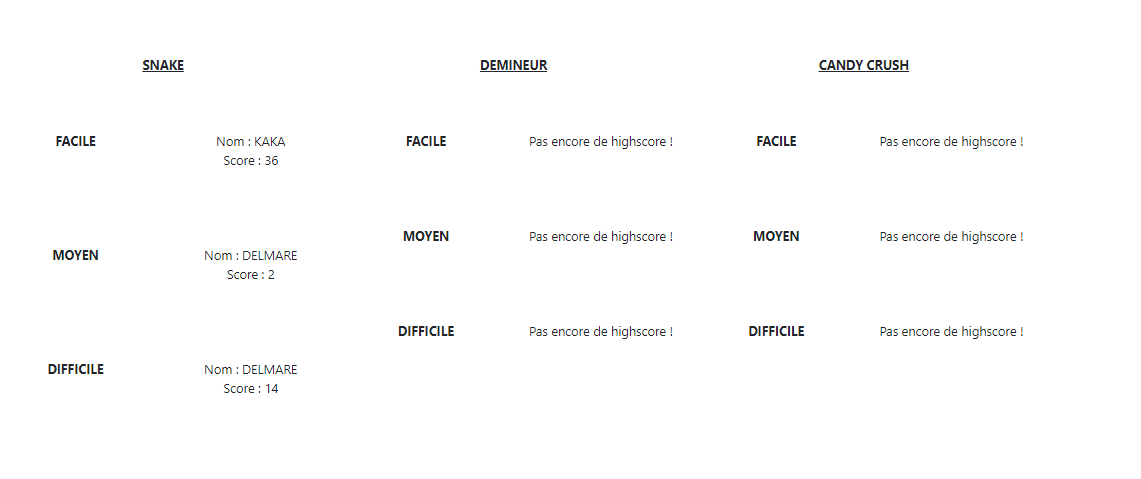




* La troisième itération concerne le développement de deux jeux, le Snake et le démineur. On a donc choisi de développer un jeu chacun, je m’occupe du Snake et Benjamin du démineur.
* La quatrième itération concerne l’intégration des jeux dans les templates du site crées lors de la première itération.



* La cinquième itération concernera l’ajout de la page de connexion au site (et donc de l’implémentation de comptes) ainsi que l’implémentation des succès dans les jeux Snake et démineur. Nous avons également ajouté le scoreboard.



* La sixième et dernière itération a été consacré au nettoyage du code, à le commenter et à harmoniser les différentes pages (amélioration du CSS).

Techniques utilisées

Nous avons développé le jeu en Javascript pour la partie back-end de notre site et concernant la partie front-end du site, nous avons utilisé HTML et CSS.  
  
Afin de nous aider au développement de notre projet, nous avons utilisé les librairies Bootstrap et JQuery.

Comparaison et explication des différences entre   
ce qui était prévu et ce qui a été fait

Au vu des maquettes proposées en début de projet et de nos objectifs, plusieurs choses n’ont pas pu être fait.   
  
Nous voulions intégrer un scoreboard au niveau du jeu, en fin de partie. Le scoreboard a finalement été intégré dans un onglet à part.

L’onglet solutions lorsque l’on est en jeu n’a pas non plus, au vu des jeux que l’on propose finalement (le snake et le démineur), une proposition de solution n’a pas d’intérêt pour le snake. Concernant le démineur, la solution sera montrée en fin de partie.

Le type de jeu n’a pas été implémenté, on voulait implémenter des parties en temps limité mais nous n’avons pas pu.

Le filtrage des jeux par leur type n’a pas été implémenté, nous avons pensé que cela n’était pas pertinent au vu du nombre de jeux proposés par le projet.

Finalement, nous voulions proposer plus de jeux (bataille navale, finalisation du Candy Crush, Casse-briques et Pac-Man) mais par manque de temps, nous n’avons pas pu les développer et les intégrer à notre site.