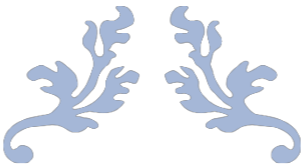
UNIVERSITÉ CÔTE D'AZUR | DIGITAL SYSTEMS FOR HUMANS GRADUATE SCHOOL AND RESEARCH

MIAGE RÉSEAU DES MIAGE DE FRANCE Nice



PROJET WEB

Rapport final

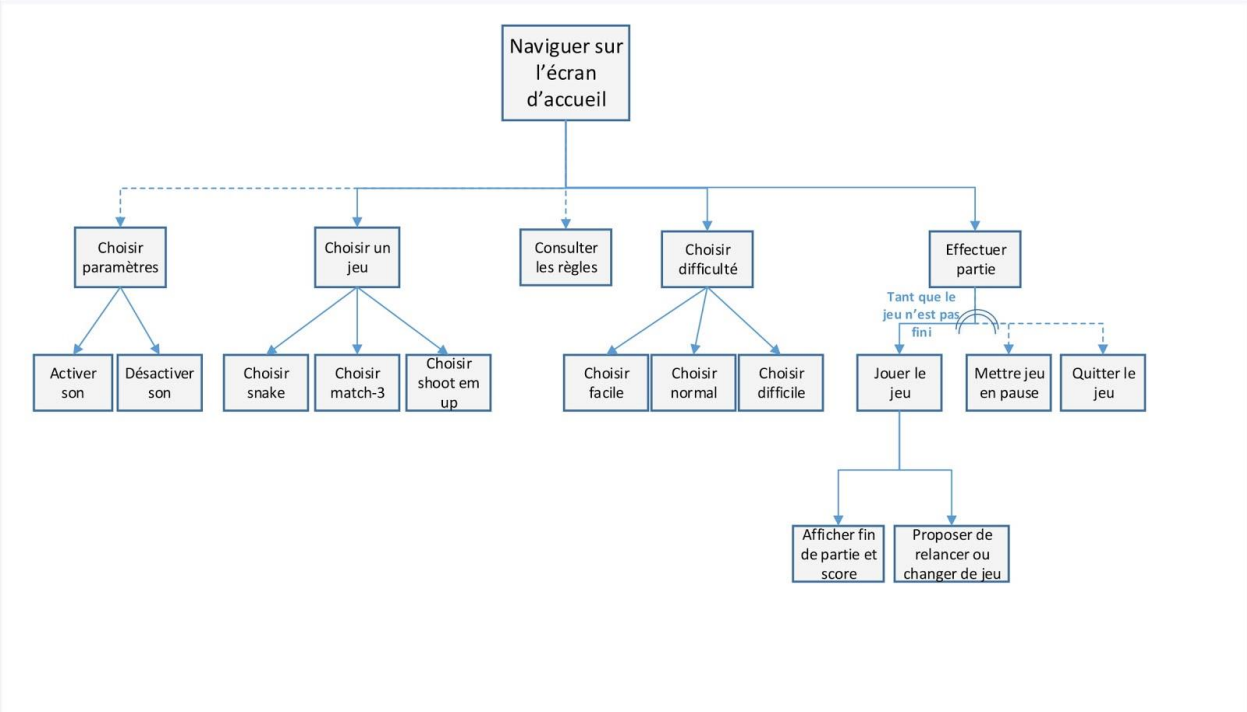
Mars 2021

Ecrit par : Saad El Din AHMED, Yessine BEN EL BEY, Hugo NORTIER

Enseignants : Michel BUFFA, Philippe RENEVIER GONIN

Table des matières

1. Modèle de tâches



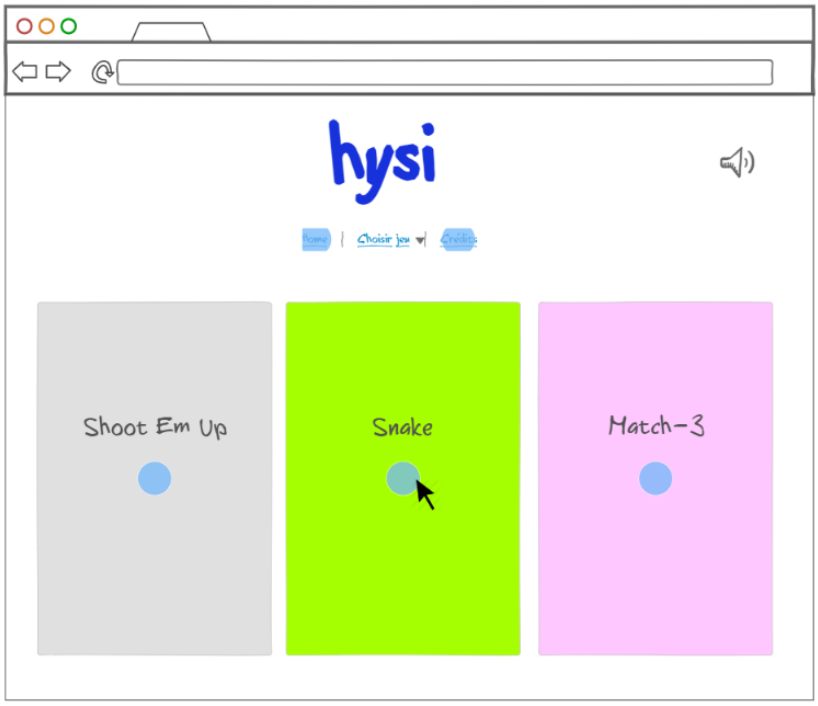
Ce modèle des tâches présente ce qui devait être réalisé lors de la conception du cahier des charges entre nous. Nous voulions un écran d'accueil sur lequel il est possible de choisir entre 3 jeux. Ces jeux sont le match3, le snake et le shoot em up.

Ces trois jeux sont construits à l'aide d'un canvas. Une fois le jeu choisi, on peut visualiser les règles et choisir la difficulté. Une fois le jeu lancé, le son de l'écran peut ou non être retiré. Chaque jeu peut être mis en pause.
Une fois le jeu fini, on affiche la fin de partie ainsi que le score du joueur et on lui propose de rejouer.

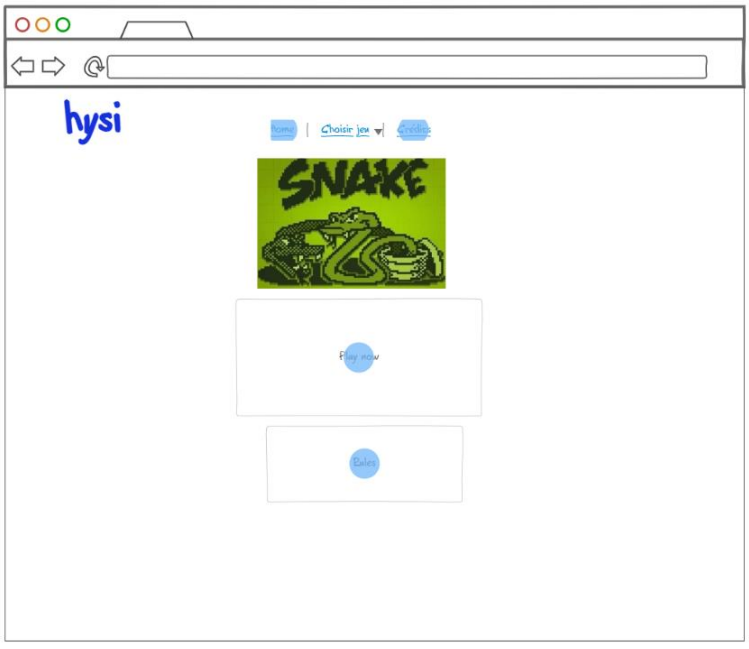
2. Maquettes réalisées

Voici les principales maquettes réalisées.
(Vous retrouverez l'intégralité des maquettes ici: https://ninjamock.com/s/GMWVHHx)

Écran d'accueil:

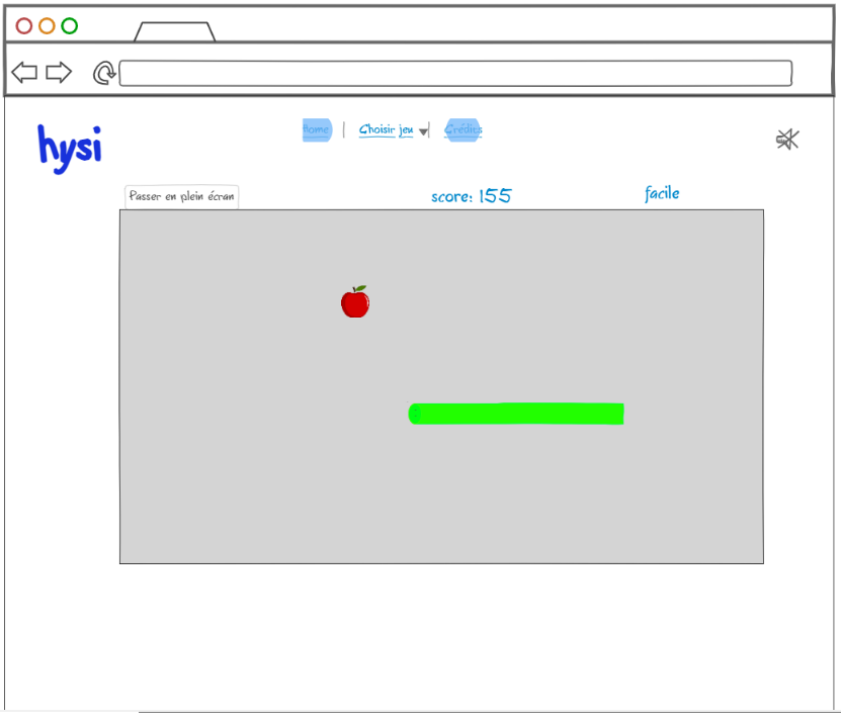


Écran d'accueil d'un jeu :



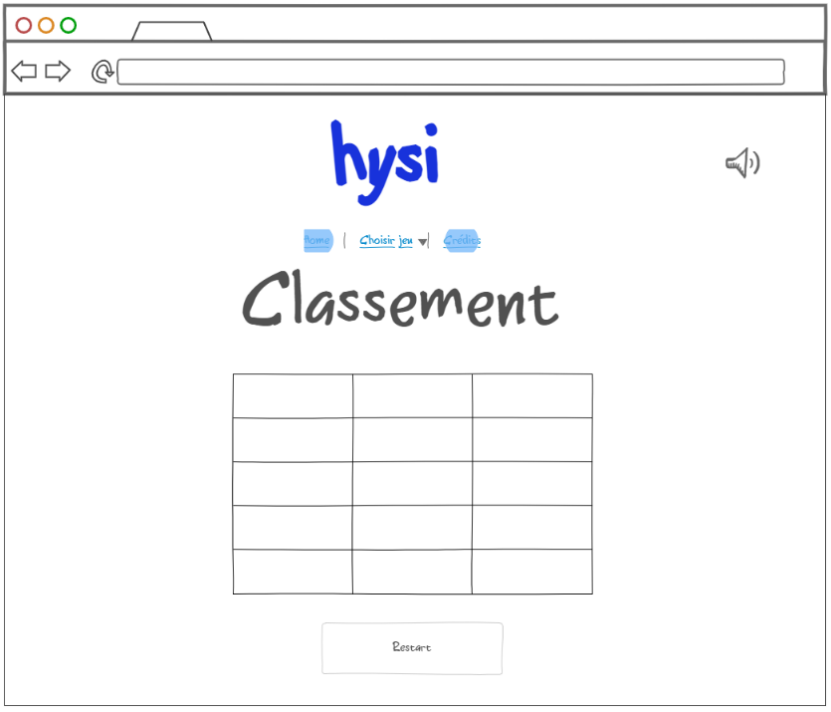
Écran des règles :

Écran d'une partie :

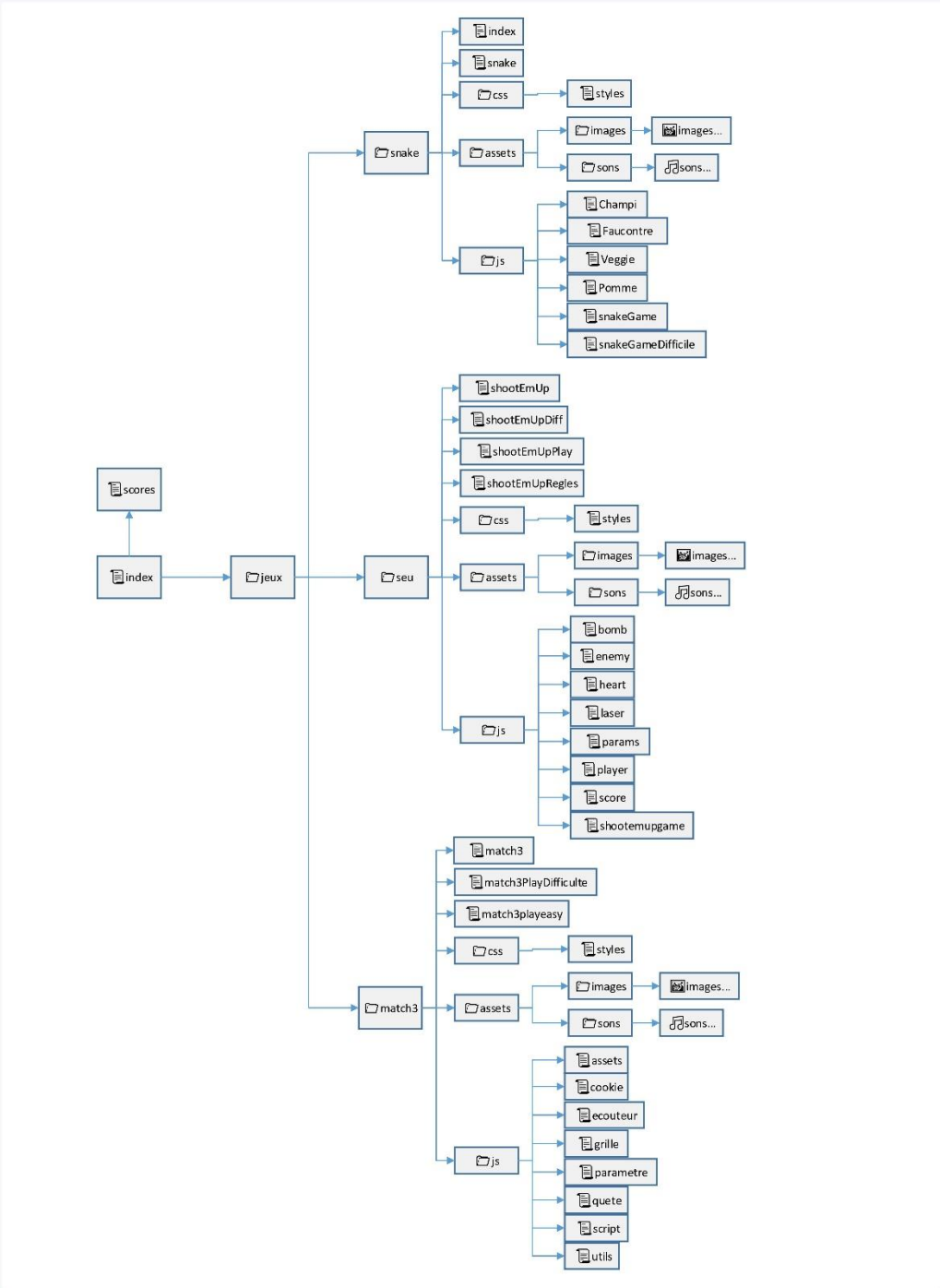


Écran de game over:

Classement :



3. Architecture logicielle



Notre écran d'accueil, là d'où tout part, est 'index.html' présent à la racine du projet 'HYSI'.
A la racine, on trouve aussi les classes javascript (js) utilisées par la totalité de l'application :

- sound.js qui correspond au bouton pour rendre muet les jeux.
- scoreglobal.js correspondant aux scores et classement de chaque jeu (sauvegardé en localstorage)

- achieve.js, achievements.js correspondent à la gestion des utilisateurs, de leurs scores et leurs trophées (sauvegardés en localstorage)
- achievementparser.js qui possède les méthodes manipulant les achievements (car on ne peut pas passer des méthodes en localstorage via json).

De index.html, on peut voyager vers scores.html qui se trouve lui aussi à la racine du projet, ce fichier affiche toutes les données correspondantes à achievement.js sauvegardés dans le localstorage.
Si l'on décide de cliquer, à l'accueil, sur l'une des icônes de jeu, on se retrouvera dans le dossier du jeu en question et du html correspondant à son menu.
Chaque jeu possède des fichiers css propres ainsi que des fichiers js. Ces dits fichiers sont propre à chaque jeu, seul styles.css reste commun afin d'avoir une page semblable de jeu en jeu.
Chaque jeu est donc sa propre entité et peut être déconnecté de l'ensemble de l'application sans l'endommager (il faut cependant faire en sorte que les icônes de l'écran d'accueil diminuent si le jeu est indisponible).
Il y a certaines différences entre l'architecture de chaque jeu niveau html :

- Le shootemup (seu) possède un html des règles de jeu contrairement aux autres qui ont une pop-up.
- Le seu et le match3 ont un html correspondant à l'écran de choix de difficultés alors que le snake fait ceci dans son index.html.

Pour ce qui est interaction entre les fichiers js, on va détailler cela par jeu :

- **Snake:**

  Initialisation (init())

  Lors du lancement du snake.html, script.js s'exécute et crée un canvas qui s'adapte à la page ouverte. Sinon, tout se passe dans la fonction init().

  Certaines fonctions diffèrent selon si le jeu a été lancé en mode facile ou difficile. Cette information est stockée en localStorage dans "isSnakeDifficile".
  Ensuite, on obtient la position de la souris en appelant getMousePos. On crée +/- d'instance des objets : pomme faucontre et champignon suivant la difficulté
  Aussi, on met un écouteur d'évènement sur la touche « p » qui servira à mettre en pause et à relancer le jeu.
  Enfin on demande une requestAnimationFrame avec en argument la fonction animate (ou animateDifficile si le jeu est en mode difficile) qui dessine le serpent (au joli nom de Veggie).

  Une fois en jeu

  On anime donc (fonction draw) chaque élément (serpent, pomme, faucon …) et on vérifie que le serpent ne s'est pas mordu la queue ni même qu'il se soit tapé contre un bord. (checkKnockout et checkCollisionTail). Auquel cas la fin de partie est affichée.
  Si le serpent interagit avec un élément (il le touche et on le sait avec la fonction checkSamePos) alors, certaines méthodes sont appelées. Par exemple, manger une pomme fait grandir() le serpent et appelle newPomme de la classe Pomme qui génère de nouvelles coordonnées pour la pomme. Aussi on incrémente le score. Toucher un faucon fait s'arrêter le jeu et toucher un champignon rend invincible aux faucons le serpent. Chaque "image" touchée génère un effet sonore sympathique.
  Chaque faucon régénère ses coordonnées (et réapparaît ailleurs) toutes les x secondes (3 en mode difficile et 5 en mode facile).

Fin du jeu

Le jeu est fini si le veggie a atteint sa taille maximale (donnée à 52 en facile et 72 en difficile) ou s'il a touché un mur, touché un faucon ou touché sa queue.

Une fois la partie finie, le popup de fin s'affiche avec le message de victoire ou de défaite, on affiche également le score et le classement et un bouton pour relancer le même mode de jeu avec la même difficulté.

- **Match3:**

Initialisation (init())

Lors du lancement du Match3Play.html alors le script.js se lance et exécute la fonction init qui initialise les paramètres et les quêtes par rapport à la difficulté stockée dans le local storage, on appelle également une fonction qui s'appelle checkFinPartie qui va être appelé 60 fois par seconde après son premier appel, cette fonction servira à savoir si la partie est fini ou pas, nous innitialisons également un interval (setInterval) permettant d'incrémenter le timer jusqu'à que le jeu ne soit pas fini.
Une fonction créeBoutonParamSon() est également appelé dès le lancement de cette page HTML permettant de créer le bouton de son en haut à droite dans le menu pour que l'utilisateur puisse mettre en sourdine le jeu s'il le souhaite.

Infos parties

Les paramètres du jeu seront visibles par l'utilisateur au-dessus du jeu et à côté de ses paramètres il y a un bouton "?" permettant de rappeler les règles aux joueurs durant la partie sans qu'il puisse la quitter. En cliquant sur ce bouton nous avons un lien href qui se lance qui se nomme popup4#, dans ce popup il y a également une croix qui va lancer le href de la page html pour qu'on puisse quitter le popup.

Blocage

Le jeu est terminé seulement si on gagne ou on perd, lors d'un blocage (C'est à dire quand on ne peut plus switch mais que le joueur a encore des coups à jouer) alors le jeu n'est pas fini, le tableau de cookies se re remplie pour que le joueur puisse continuer à jouer, lorsqu'il y a blocage alors on met l'overlay3 et la popup3 en display block pour qu'on puisse afficher la popup, dans cette popup on a un bouton "mix" qui permet de re remplir le tableau (grâce à la fonction remplirTabeauCookies()) et de remettre l'overlay et la popup à display 'none 'lorsqu'on clique dessus ce qui permet quitter la popup.

Switch et chute de cookies

On peut switch seulement si après ce switch il y a une chute.
Tout d'abord on a un événement lors d'un clic de souris, lorsqu'on clique sur un cookie on va

appeler la fonction onMouseDown qui va appeler faire un son si dans le localStorage on a la valeur de “son” à ‘sonore’ et ensuite si le cookie cliqué a un état cookieEnDrag alors on ne fait rien mais si le cookie a un état ‘rien’ alors on le met en cookieEnDrag permettant lors de l’animationLoop de le dessiner à xCookie/0.8 et yCookie/0.8 pour avoir un cookie dessiné en décalage lors d’un clique.
Lorsqu’on relâche la souris on fait la fonction onMouseUp qui va agir seulement si le cookie est dans l’état CookieEnDrag, cette fonction va regarder si un switch est possible pour pouvoir le faire et ensuite faire la chute, update le coup et update le score ainsi que les quêtes pour ensuite mettre le cookie à l’état ‘rien’ sinon si le cookie n’est pas CookieEnDrag on ne fera rien.

Victoire ou défaite

Lorsqu’il y a win ou défaite l’overlay correspondante sera en display ‘block’, le même procédé est utilisé que le mix mais contrairement au mix tous les paramètres sont refaits et il y a un classement dans ces popup permettant de classer le joueur par rapport au score qu’il a fait durant sa partie. Dans ces popup il y a un bouton tryagain permettant de relancer le jeu et c’est ce bouton qui reappelle la page HTML match3play.html.

Quêtes

Les quêtes sont créées dès l’initialisation, elles possèdent un width à atteindre, un id, un état et un txt, elles sont initialisées par rapport à leur type (Type 1, 2 ou 3) car elles ont des attributs différents (Numéro de ligne, type de cookie...).
Elles vont être mise à jour après une chute, on appellera donc updateQuest qui va vérifier les cookies qui ont un état de disparition et suivant ça on rajoute des points ou pas, lorsqu’on doit rajouter des points alors on va devoir faire progresser la barre on va augmenter le width de la quête par rapport à l’objectif finale, c’est à dire si l’objectif est d’avoir 3 cookies explosé à la ligne 1 alors lorsqu’on on explose un à la ligne 1 on aura ⅓ de la barre qui sera en vert.
Lorsqu’on finit une quête le score s’incrémente de 100 pts (En faisant getElementById du score et on ajoute 100 après avoir faire un parseInt).
Lorsque les quêtes sont dans l’état “completed” alors elles ne feront plus d’update.

- **Shoot em up**

Quand on lance ShootEmUpPlay.html, le fichier shootemupgame.js se met en marche, il fait un init de toutes les statistiques du jeu par rapport à la difficulté (retrouvée par le bouton de difficulté sélectionné par le joueur) et initialise aussi les vies, les bombes et le vaisseau du joueur. Tous ces éléments correspondent à une classe, chacune dans un fichier js différent. Les bombes dans bomb.js, les vies dans heart.js et le vaisseau dans player.js.

Une fois ces éléments initialisés, la boucle main du jeu se met en route tant que le joueur possède des vies (son objectif est d'avoir le meilleur score possible et pas finir un niveau défini). Cette boucle tourne en parallèle avec une autre boucle que toutes les x secondes crée un ennemi aléatoire d’un set prédéfini dans Enemy.js (on distingue 4 types). Les ennemis et le joueur peuvent lancer des projectiles (laser.js) qui peuvent blesser l’opposant.

La boucle main vérifiera s'il y a eu un input du joueur (pour bouger ou tirer un projectile), s'il y a eu collision, si le joueur a encore des vies, il mettra à jour la position de chaque élément puis les redessine sur le canvas.

Si le joueur n'a plus de vie après vérification, alors la boucle s'arrête et le popup de game over sera affichée avec les meilleurs scores récupérés en local storage après appel de la méthode de mise à jour des scores dans scoreglobal.js à la racine du projet.

4. Planning de réalisation

Nous avons fait le début (tout ce qui est conception et écran de menu principal) ensemble, ensuite nous nous sommes reparti les jeux, un par personne. Yessine s'est occupé du Match-3, Hugo du snake et Saad du Shoot em up.

Chaque semaine on faisait une réunion où on discutait de nos avancements, nos problèmes afin de mettre en commun et pouvoir s'entraider.

Voici le déroulement du projet par date :

27 janviers au 3 février : Réalisation du modèle des tâches et réflexion sur l'IHM.
7 février : Retour du professeur et modification du modèle des tâches, réalisation des scenarii.

11 février : Finalisation de la maquette après les retours du professeur et répartition des tâches.

16 février : Création du menu principal et des menus de chaque jeu.

Snake :

16 février : Création de l'écran d'accueil à la racine, qui mène aux 3 jeux.

17 février : Création de l'écran d'accueil du jeu snake

18 février : Création d'un bouton pour rendre muet la page

21 février : Mise à jour du css de la page d'accueil du snake et création d'une fenêtre modale proposant de jouer au jeu ou voir les règles.

25 février : Mise à jour de la susdite fenêtre modale et utilisation du localstorage pour le stockage et la gestion des pseudos

26 et 27 février : Mise à jour du css et du pied de page de la page d'accueil du snake, et début de snake.

1er mars : Création des pommes pour snake

8 mars : Création des classes pour les différents objets (pomme, faucontre)

9 mars : Création du fichier javascript : Veggie.js (le fichier du snake)

16 mars : Mise à jour du snake (collisions avec bord, avec faucontre et pomme et gestion de la fin de partie)

17 mars : Gestion des problèmes d'apparition des objets, popup de fin différente si victoire ou défaite et qui affiche le classement et le score.

18 mars : Création d'une classe "champignon" pour un objets champi qui rend invincible Veggie (le snake) au faucontre (le faucon ennemi) + refactoring.

Match 3 :

17 février : Création de l'écran de difficulté ainsi que l'écran du jeu, ajout d'un popup gameOver

18 février : Faire la chute des cookies, switch que pour casser

19 février : Implémentation du GameOver et affichage du popup lorsque ça arrive
20 février : Finalisation de la popup GameOver pour un meilleur affichage, Ajout du son et implémentation du vibration cookies

21 février : Implémentation de la hiérarchie état du cookie, mettre un score à atteindre et l'incrémenter quand on casse des cookies pour qu'on puisse gagner

23 février : Ajout des coups, lorsque coup est à 0 on a perdu, ajout d'une classe paramètre qui va contenir le score, le nombre de coups initial du jeu et du nombre de cookies.

24 février : Gérer les chutes dans les chutes

25 février : Implémentation de la popup win et de la popup blocage qui survient lorsqu'on ne peut plus switch mais que le joueur a des coups restants

26 février : Ajout du logo mute/unmute dans le menu et transmettre si c'est muté ou pas aux autres jeux pour savoir comment initialiser les jeux. Ajout d'animation lors du switch et de l'apparition d'un nouveau cookie.

27 février : Ecran de règle

28 février : Finalisation de l'écran règle

1 mars : Difficulté moyen et difficile commencé et finalisé

2 mars : Ajout de quêtes dans le jeu

3 mars : Animation des chutes différemment pour que l'utilisateur ne soit pas perdu lors des chutes dans des chutes.

10 mars : finalisation des quêtes et sons

14 mars : Modifications de tous les scores et coup pour que le jeu soit plus difficile

16 mars : Mise en place du classement et modification des popup pour que l'affichage soit correct

17 mars : Commentaires, mettre le jeu en anglais, modifications de certains bugs

Shoot em up :

16 février : Création de l'écran de titre du jeu.

18 février : Création d'écran de règles, écran de choix de difficulté et première version du jeu

19 février : Modification déplacements du vaisseau et ajout des sons

25 février : Création de bombes, image d'arrière-plan

2 mars : Mise en place du système de pause, de plusieurs types de jeu et spawn d'ennemis random

3 mars : Création de barre latérale de scores affichant vies, scores, difficulté et message de pause

15 mars : Création d'écran de gameover, compteur d'ennemis tués

17 mars : Nettoyage de code et commentaires

17 mars : Création de page de scores, mise en place de système de trophées et implémentation dans chaque jeu

5. Techniques utilisées

Nous avons utilisé html et css pour le fond et la forme des pages du site web.

Nous avons utilisé JavaScript pour tout ce qui est logique des jeux.

Nous avons codé en utilisant des objets (chaque jeu possède des classes, par exemple match 3 possède des cookies et une grille qui sont des instances d'objets "Grille.js" et "Cookie.js").

Chaque jeu est construit dans un canvas et fonctionne uniquement en javascript vanilla.

6. Comparaison et explication des différences

Ce qui était prévu	**Ce qui a été réalisé**
✅ un écran d'accueil	
✅ un écran par jeu (seu, snake, match-3)	
✅ différentes difficultés par jeu	
✅ désactiver son du jeu	
✅ consulter les règles de chaque jeu	
✅ jouer chaque jeu	
✅ écrans de game over, possibilité de relancer une partie	
✅ affichage du classement à la fin d'une partie	
un écran de crédits	→ un pied de page de crédits
mettre en pause chaque jeu	→ mettre en pause chaque jeu sauf match3 qui n'en a pas l'utilité
	✅ écran d'affichage des trophées et scores de chaques joueurs
	✅ gestion des pseudos
	✅ gestion des trophées
	✅ gestion des classements
	✅ stockage des données dans le localstorage