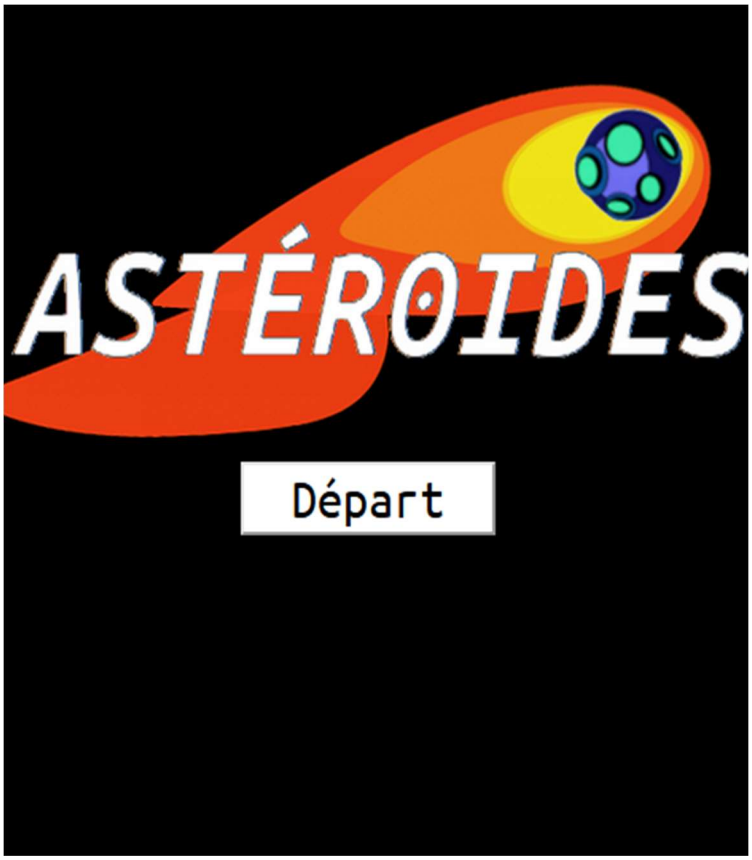


Astéroïdes

LE JEU BIEN AIMÉ PAR TOUS, RECRÉÉ EN JAVASCRIPT

Par: Youssef Rachad

Présenté à: M. Jean-François Bérubé

Dans le cadre du cours : ICS4U

Le vendredi, 30 avril 2021

École Secondaire Roméo-Dallaire

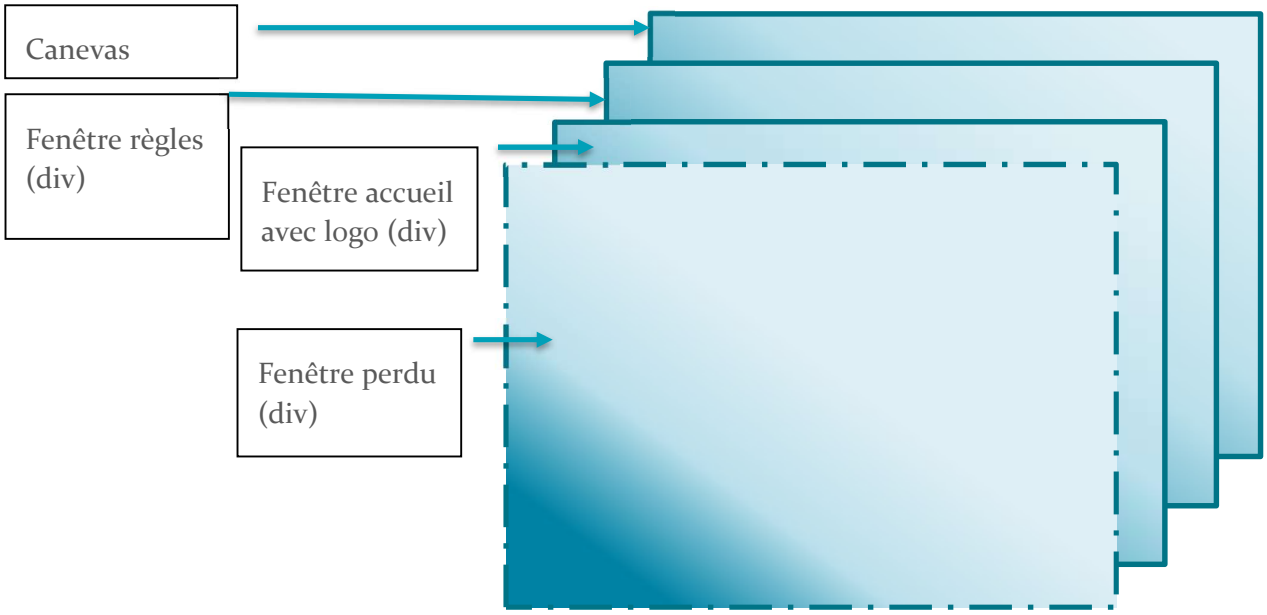
Objectif du jeu

Astéroïdes est un jeu d'habileté qui consiste à détruire des vagues d'astéroïdes en pleine trajectoire vers la Terre. Capitaine du vaisseau, le joueur a la tâche ardue de diriger le vaisseau et tirer des astéroïdes, qui rapportent des points.

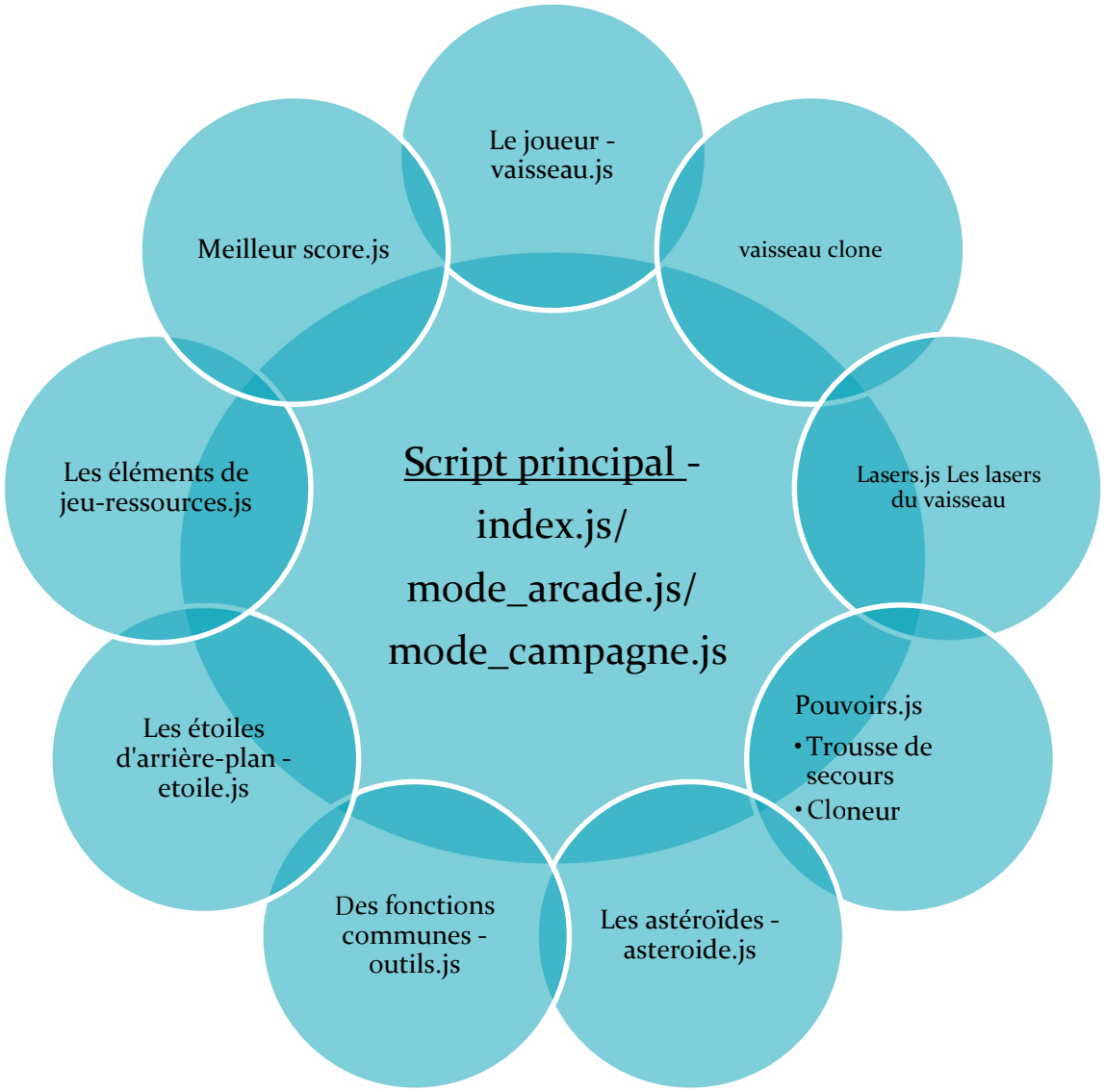
Structure du site web

Le site comporte 1 pages web qui est modifiée pour apporter une expérience immersive au joueur. Toutefois, les scripts sont divisés en 11 fichiers distincts pour mieux structurer le projet. Aussi, la page comporte 4 fenêtres : le canevas, la page de règles, la page d'accueil et la fenêtre « perdu ». On navigue d'une fenêtre à l'autre à l'aide de boutons et de la logique du jeu.

PAGE WEB → FENÊTRES



LES SCRIPTS



FONCTIONNALITÉ DES SCRIPTS

INDEX.JS – DÉBUTER LE JEU

Le début du jeu consiste à initialiser toutes les variables du canevas, de l'animation et des éléments du jeu. C'est ici que l'on créé les tableaux, les étoiles d'arrière-plan, l'arrière-plan noir. On traite aussi les touches du clavier grâce aux évènements « onKeyDown » et « onKeyUp ». On définit une fonction qui se charge d'appeler la fonction callback spécifique à chaque touche. Ces fonctions doivent être de nature fonction flèche et appelle la fonction de déplacement pour le vaisseau et son clone, si ce dernier existe. Il y a aussi les

écouteurs d'évènements pour traiter les boutons du menu, qui affichent la fenêtre appropriée, créer la première vague d'astéroïdes et déclencher un mode de jeu.

MODE_ARCADE.JS/MODE_CAMPAGNE.JS – LES JEUX

Ces deux scripts exécutent la boucle de jeu à l'aide de la fonction récursive dessiner() qui appelle la fonction requestAnimationFrame(). Les lignes 4,5,6 servent à contrôler la vitesse du jeu à 60 ips. Ensuite, on traite les touches du clavier avec la fonction toucheClavier(). Ensuite on sauvegarde et dessine un canevas noir. En servant des méthodes bouge() et dessine(), on affiche les étoiles et les lasers grâce à des boucles for. La planète est dessinée après les étoiles pour la placer en avant-plan.

Puis, on traverse le tableau des astéroïdes. Pour chaque astéroïde, on la déplace et la dessine avec ses méthodes. Puis on vérifie que l'astéroïde n'est pas en collision avec le vaisseau ni la planète. Si oui, on retranche un point de vie du vaisseau et du vaisseau clone et on joue un effet sonore, enlève l'astéroïde et met à jour l'image du vaisseau. Aussi on vérifie que l'on ait atteint ø. Si oui, on arrête le jeu. Le canevas est remis au noir et le score est affiché à l'écran «perdu» avec une animation. Si le score est un nouveau record, le «local storage» est mis à jour. Enfin, le tableau des astéroïdes est vidé, le vaisseau clone est désallouer de la mémoire et la position du vaisseau réinitialisée.

Ensuite on vérifie si le vaisseau clone existe s'il est en collision avec l'astéroïde actuellement itéré. Si oui, le vaisseau clone se désintègre et l'astéroïde est enlevé du jeu.

Après les collisions avec vaisseaux, on vérifie si l'astéroïde est en collision avec un laser grâce à une boucle for. Si oui, on enlève l'astéroïde et le laser du jeu. On joue aussi un effet sonore et le score du joueur est incrémenté de 10 points. Ceci est fait pour chaque astéroïde.

La partie suivante du script dessine et déplace les vaisseaux (le vaisseau clone seulement s'il existe) et les pouvoirs de trousse de secours et de cloneur.

Le canevas est ensuite restauré et le temps zéro (t_ø) est mis à jour.

La dernière partie du script effectue la détection de collision entre les pouvoirs et le vaisseau. Si l'on attrape une trousse de secours, on gagne une vie, sauf s'il on a déjà 3 vies. Si l'on attrape un cloneur, on crée le vaisseau clone et on débute son compteur à zéro. Si le vaisseau clone existe déjà, on lui ajoute quelques centaines de millisecondes de temps. Puis on enlève du jeu tous les lasers qui dépasse la ligne décrite par y = 30. Ici, il faut vérifier que le tableau contient au moins un laser (laser[ø] renvoie «true» si l'élément existe). La logique de la méthode pop() provient du fait que le premier élément est toujours le plus près de la frontière puisqu'il est le premier à être tiré et que les lasers ont tous la même vitesse. Ensuite, on vérifie si le vaisseau clone existe et si son temps est écoulé. Si oui, il est enlevé du jeu.

Enfin on met à jour le score à l'écran et on appelle la prochaine image, soit requestAnimationFrame().

Les deux scripts décrivent deux modes de jeux et diffèrent selon leur régénération d'astéroïdes. Le mode arcade et un mode où les astéroïdes sont créés au hasard et à intervalle irrégulier. Pour ce mode, on créé un astéroïde dès qu'une est détruite soit par un laser ou par un vaisseau. Le mode campagne a une approche qui consiste à envoyer les astéroïdes par vagues. On vérifie donc à la fin de chaque boucle de jeu si le tableau des astéroïdes est vide afin d'appeler dessiner_campagne() et régénérer une nouvelle vague d'astéroïdes.

ÉTOILE.JS – LES ÉTOILES

Ce script définit une classe pour les étoiles de l'arrière-plan. Les étoiles ont une position (x, y), une dimension standard et une couleur au hasard, tirée de outils.js. Au début du jeu, on créé 100 étoiles que l'on ajoute à un tableau.

MEILLEUR_SCORE.JS – LES RECORDS

Ce script définit des fonctions pour obtenir et enregistrer les meilleurs scores obtenus par le joueur. Si les variables de score ne sont pas encore initialisées dans le « local storage » (un stockage non volatile, unique à chaque utilisateur et chaque navigateur), le script les créé. Au départ du jeu, ce script affiche les scores dans la boîte de score.

VAISSEAU.JS – LE JOUEUR

Le vaisseau est le joueur du jeu. Les images qui servent à le représenter ont un nom selon le format 'vaisseau' + nombre de vies + 'bouge' pour les images avec une propulsion. De cette manière, le script peut dessiner une image de forme rectangulaire contenant le vaisseau dans son état approprié.

Pour controller le vaisseau, il suffit de donner un écouteur d'évènement « onKeyDown » au document et extraire la touche de clavier appuyée pour ensuite déclencher une action selon le code de la touche (*e.keyCode*). Toutefois, mon approche naïve ne permettait pas le mouvement diagonal et les tirs en mouvement, c'est-à-dire le traitement de plusieurs touches simultanément. Selon *Handling Multiple Key Presses at Once in Vanilla JavaScript (for Game Controllers)* par Dover, il suffit d'implémenter un object contenant toutes les touches que l'on désire traiter et de leur attribuer une propriété « actif » deux écouteurs : « onKeyDown» et « onKeyUp » et de se souvenir de l'état « actif » d'une touche.

Ensuite, l'effet de propulsion (l'image du feu qui apparaît sous le vaisseau) n'est plus contrôlé par une condition « if » qui est plus coûteuse. À la place, les méthodes de déplacement (monte, descend, droite et gauche) et de tire indique l'image ayant le feu dans la source de l'image du vaisseau. Pour « éteindre » le vaisseau, l'événement « onKeyUp » remodifie la source pour indiquer une image n'ayant pas de feu. D'ailleurs, les noms d'images ont été préformatés pour correspondre à un état spécifique et la méthode «

split() » permet d'extraire le préfixe approprié de l'URL, tandis que la variable « vies » indique quelle image choisir. Ceci a pour effet secondaire de rendre l'alternance de l'image du vaisseau plus stable à l'entrée de plusieurs touches.

Lorsque le vaisseau tire un laser, ce dernier s'ajoute au tableau (à portée globale) à la position du vaisseau, après un petit décalage. Aussi, pour chaque 1000 points, le joueur a droit a un projectile par tire. Par contre, le joueur ne peut tirer qu'une fois toutes les 30 secondes grâce à un compteur et un drapeau (valeur booléenne).

VAISSEAU_CLONE – La sous classe pour le clone

Le vaisseau clone utilise le même constructeur que sa classe parent, ajoute une variable compteur pour sa durée de vie et s'assure d'initialiser avec la bonne image. D'autant plus que le nom de la classe ajout un aspect sémantique à la logique de jeu.

LASER.JS – LES LASERS

Les lasers sont des rectangles ayant une paire de coordonnées, une couleur et des dimensions standards. La méthode *bouge()* fait avancer un laser de 5 pixels vers le haut de l'écran, tandis que la méthode *dessine()* trace un rectangle sur l'écran selon la position, dimensions et couleur d'un laser donné.

En gardant les lasers dans un tableau, il est plus facile de gérer leur dessin et leur déplacement. En utilisant un seul tableau pour les lasers du vaisseau et de son clone, on passe à travers un seul tableau. Ceci a contribué une économie de 50 lignes de codes dans deux fichiers (100 lignes en total) pour améliorer la lisibilité des scripts et réduire leur taille au moment de charge par le navigateur.

ASTEROIDE.JS – LES ASTÉROÏDES

Les astéroïdes sont dessinés à partir de cercle dans le canevas. Pour permettre une variété de styles et de couleurs, on utilise des tableaux pour les jeux de couleurs et pour les fonctions qui décrivent la forme.

Dans le mode campagne, les astéroïdes surviennent par vagues. Les vagues sont prédéfinies dans un tableau, où un astéroïde est un 1 et un ø est un espace vide. La fonction «dessiner_campagne» vérifie si un élément du tableau de formation est 1. Si oui, on interprète la valeur booléenne «True» et on peut dessiner un astéroïde. Dans la valeur x, on exécute un modulo pour former 6 colonnes et, dans la valeur y, on exécute un arrondissement pour former 4 rangées (en divisant par 6, on obtient un multiple de 6, soit la position y que l'on décale par 1).

POUVOIR.JS – LES BONIS

Ce script définit une classe et deux sous classes pour chaque pouvoir boni que l'on peut attraper dans le jeu. La classe parent définit la position, l'angle utilisé dans les calculs et le

rayon. Elle signale aussi la méthode « dessin » que chaque sous classe doit implémenter et un fonction bouge(), où le déplacement vertical est linéaire, tandis que le déplacement horizontal est sinusoïdale pour donner l'illusion de flotter. À noter que la méthode context.beginPath() est nécessaire, sinon la couleur du pouvoir persiste et modifie la couleur des astéroïdes.

RESSOURCES.JS – LES ÉLÉMENTS EXTERNE

Ce script contient l'initialisation des images utilisés et des fichiers audio. Aussi, il contient les écouteurs d'évènements requis pour régler le volume dans le jeu.

CONSOMMATION DE MÉMOIRE

Lors des tests du prototype du jeu, j'ai remarqué qu'il y avait des instants de chute dans les ips, allant jusqu'à 16 ips, tel indiqué dans la figure XX. Ceci est inacceptable du point de vue du joueur et signale des fléaux dans la gestion de la mémoire du jeu.

Une inspection des performances du la page web indique un processus appelé « Non-inremental G[arbage]C[ollection] ». Selon les ressources du MDN, le NIGC se met en marche pour désallouer d'importante quantité de mémoire lorsque le collecteur juge qu'il manquera de mémoire bientôt. Il faut donc éviter de trop charger la mémoire du site web et la vider d'un trait. On remarque que ce procédé s'effectue après avoir complété une joute, soit au moment où les astéroïdes de la joute précédente sont retirés du tableau « astéroïdes » et où l'on ajoute une nouvelle vague.

Afin d'éviter d'allouer et de désallouer d'importante quantité de mémoire à chaque fois que le jouer recommence une nouvelle partie, il faut désallouer la mémoire petit à petit. Pour signaler la désallocation d'une variable ou d'un objet en JavaScript, il est courant de lui assigner la valeur *null*, soit lui retirer sa déclaration et augmenter sa priorité de désallocation pour le collecteur.

L'opération d'allocation de mémoire est abordable au niveau de la mémoire et on peut demeurer à un minimum de 40 ips grâce à cette amélioration et aux deux prochaines :

D'abord, lors de l'arrêt du jeu, la méthode window.cancelanimationFrame() est appelée pour permettre de garder en mémoire le tableau des étoiles en arrière-plan au lieu de le désallouer et le réallouer.

Au début, on peut voir le NIGC prenant 300ms et faisant chuter les ips jusqu'à moins de 10.

Après les optimisations, on ne descend plus en-dessous des 40 ips.

ESTHÉTISME

L'esthétisme du jeu exhibe un thème à la fois jeu arcade rétro et aventure spatial. L'arrière-plan du site web est un fond spatial qui n'attire pas d'attention grâce aux couleurs froides, maximisant ainsi la concentration du joueur sur le jeu. Quant au jeu, les couleurs sont limités tandis que les formes sont plutôt pixelisés pour simuler l'apparence d'ancien jeux arcade qui avaient des résolutions d'écran inférieures à celle couramment trouvées de nos jours. L'utilisations de certaines images plus lisses indiquent le progrès que l'on retrouve dans l'utilisation de technologies modernes, tel JavaScript. Ce mélange de styles donne une apparence plaisante au joueur, tout en lui rappelant (ou exposant) les origines des jeux vidéo.

Quant au fichier index.css, il contient les styles de la page. Les dimensions des minis fenêtres, l'arrière-plan et la position des éléments y sont spécifiés.

EFFETS SONORES

Les sons utilisés dans ce jeu visent le style rétro-arcade, étant courts avec beaucoup de bruit de fond. Quant à la musique, elle évoque le sentiment de s'aventurer dans l'espace.

POINTS À AMÉLIORER

Bien qu'une mise à jour ne soit pas prescrite pour le moment, il y a bien d'aspects que l'on peut pousser plus loin. Par exemple, des algorithmes peuvent être développées pour générer des vagues d'astéroïdes dans le mode campagne et offrir une meilleure expérience de jeu. On peut faire augmenter un niveau de difficulté. Aussi, les astéroïdes peuvent avoir

plus de styles. On peut même avoir un astéroïde en or qui apparaît rarement et accorde 50 points. Quant au vaisseau, un système de vente peut être mis en place de sorte que le joueur peut s'acheter de nouveaux vaisseaux, soit d'apparence différentes ou de fonctionnalités amélioré.