Rapport de conception

Groupe Dreamz

Yohan Ménager - Clément Gorbatch - Mourichidatou Biokou

Déploiement

L'application utilise un backend Node.js minimal (Express) pour servir les fichiers du frontend. Les fichiers statiques sont organisés dans un dossier front, tandis que le serveur est dans back. Le tout est déployé sur Vercel, qui gère l'hébergement automatique à partir d'un dépôt Git. Le système repose sur un routage dynamique : chaque jeu ou section du site peut être chargé indépendamment, optimisant les performances et réduisant les temps de chargement. Ce découpage en pages légères permet aussi une meilleure maintenabilité, avec un préchargement ciblé selon le contexte utilisateur. La compatibilité Vercel avec Node.js permet de conserver un backend simple, tout en facilitant la gestion des routes et des ressources à livrer.

Jeu 1 : Cookie Crush

1- Description du jeu

Cookie Crush est un jeu développé en JavaScript en manipulant directement le DOM d'un document HTML. Il s'inspire du jeu mobile Candy Crush, en reprenant ses mécaniques principales : réaliser des alignements d'éléments, ici des cookies, pour marquer des points.

2- Objectif du jeu

L'objectif du jeu est de faire des alignements de cookies identiques, horizontalement ou verticalement. Lorsqu'un alignement est détecté, les cookies concernés disparaissent et le joueur marque des points. De nouveaux cookies tombent ensuite pour remplir les espaces vides, ce qui peut entraîner des combos.

3- Fonctionnalité

L'interface du jeu repose entièrement sur la manipulation du DOM. La grille de jeu est générée dynamiquement avec des cookies de différentes couleurs. Le joueur peut interagir en échangeant deux cookies adjacents, soit par glisser-déposer, soit par clics successifs. Après chaque mouvement, le jeu vérifie automatiquement s'il y

a un alignement valide. Si c'est le cas, les cookies disparaissent et le score augmente.

Le score est géré localement : il est stocké dans le localStorage, mais uniquement si le joueur dépasse son meilleur score précédent. Cela permet de conserver un historique et de faire un classement entre les différents joueurs, même après avoir fermé ou actualisé la page.

Sur le plan technique, le jeu repose sur une structure HTML pour l'affichage de la grille, stylisée via CSS. Le comportement dynamique est assuré par JavaScript, avec une gestion manuelle du DOM pour créer, modifier et supprimer les éléments de la grille. Le localStorage est utilisé pour enregistrer et comparer les scores.

4- Pistes d'améliorations

Plusieurs pistes d'amélioration sont envisageables. Il serait intéressant d'ajouter des animations lorsque les cookies explosent, ainsi que des cookies qui auraient des effets spéciaux lors de son explosion. Il serait également pertinent d'introduire des systèmes de combo ou de bonus spéciaux pour varier les stratégies. Enfin, une interface pour les appareils mobiles et l'ajout d'un classement permettraient d'élargir l'intérêt du jeu.

Jeu 2 : Petit monstre (Jeu canvas)

1- Description

Ce jeu exploite <canvas> pour les graphismes et diverses interactions. Il comprend la gestion des niveaux, des collisions, des écouteurs d'événements, des ennemis autonomes, des objets interactifs, des bonus et un système de particules. La logique du jeu est intégrée à Game.js, qui orchestre les règles de progression et d'interaction.

2- Objectif du jeu

Le jeu comporte 7 niveaux. Pour gagner, le joueur doit atteindre la sortie du niveau 7. Si le joueur réussit à passer tous les niveaux, il entend un son de victoire et voit un message de félicitations.

3 - Fonctionnalités

- Gestion complète du jeu (Game.js) :
  - Gestion des niveaux avec progression et nouveaux éléments à chaque étape.
  - Chronomètre limitant le temps par niveau.
  - Système de score évolutif.
  - Détection avancée des collisions entre objets.
  - Gestion du joueur : mouvements fluides et interactions.

- ❖ Gestion des ennemis : déplacement autonome et test de collision avec le joueur.
- ❖ Power-ups et objets spéciaux modifiant la taille du joueur et augmentant le score.
- ❖ Effets sonores dynamiques (collision, victoire, power-up, changement de niveau).
- ❖ Animation fluide avec requestAnimationFrame.

- Interface utilisateur interactive avec boutons pour démarrer, rejouer et arrêter le jeu
- Capture des entrées utilisateur : Prise en compte du clavier (keydown, keyup) et de la souris (mousemove).
- Personnalisation du fond : Changement de l'image de fond et de la musique selon le niveau.
- Système de particules : Effets visuels lors de certains événements.

- Fonctions utilitaires :
  - ❖ Dessin rapide de cercles.
  - ❖ Création de grilles pour le positionnement des objets.
  - ❖ Rédimensionnement dynamique : Mise à jour automatique du canvas en cas de changement de taille de la fenêtre.

4- Dépendances

Le jeu repose sur plusieurs modules :

- ★ *Game.js* : Cœur du jeu (niveaux, collisions, musique, score).
- ★ *collisions.js* : Gestion des interactions entre objets.
- ★ *écouteurs.js* : Capture des entrées utilisateur.
- ★ *Ennemi.js* : Déplacement et rendu des ennemis.
- ★ *ObjectGraphique.js* : Classe de base pour les objets graphiques.
- ★ *Modules d'objets interactifs* : ObjetSouris.js, ObjetSpecial.js, Obstacle.js, PowerUp.js, Sortie.js.
- ★ *Particles.js* : Génération d'effets visuels dynamiques.
- ★ *Player.js* : Gestion du personnage joueur.
- ★ *utils.js* : Fonctions de rendu graphique.

5- Utilisation

- *Démarrer le jeu* : Cliquez sur Démarrer.
- *Rejouer* : Réinitialise la page.
- *Arrêter* : Redirige vers une page externe.

- Comment gagner des points ?

  Le joueur peut accumuler des points de différentes manières :
  - ❖ Atteindre la sortie : Chaque fois que le joueur atteint la sortie, il passe au niveau suivant et gagne 100 points.

- ❖ Ramasser un objet spécial : Ces objets bonus rapportent 50 points et permettent aussi de monter de niveau.
- ❖ Ramasser un power-up : Cela rapporte 5 points et réduit légèrement la taille du joueur, ce qui peut l'aider à éviter les obstacles.

- Comment perdre le jeu ?
  Le joueur peut perdre de différentes façons :
  - ❖ Temps écoulé : Chaque niveau est limité à 30 secondes. Si le joueur n'atteint pas la sortie avant la fin du temps, il perd et voit apparaître un écran de fin.
  - ❖ Collision avec un ennemi : Si le joueur entre en contact avec un ennemi, il perd instantanément et un message s'affiche indiquant la défaite.
  - ❖ Sortie du jeu volontaire : Si le joueur clique sur le bouton Arrêter, il est redirigé vers une page externe et la partie s'arrête.

6- Points d'améliorations

Plusieurs améliorations sont envisageables. Par exemple, le monstre pourrait être remplacé par un personnage contrôlable disposant de capacités de déplacement plus riches, comme la course ou le saut d'obstacles. Cela permettrait d'enrichir l'interaction avec le joueur en intégrant des mécaniques de gameplay plus dynamiques. De plus, il serait intéressant d'explorer des alternatives aux contrôles clavier classiques, telles que l'utilisation de la souris, d'un joystick virtuel ou de gestes tactiles, afin d'améliorer l'ergonomie et l'accessibilité du jeu.

Jeu 3 : Dreamz

1- Description

Le jeu repose sur un moteur Babylon.js initialisé dans une unique scène, évitant les rechargements entre niveaux pour optimiser les performances. Le système de points est personnalisé pour chaque utilisateur via la classe GestionPoints, qui stocke les meilleurs scores et la progression dans le localStorage. Les bonus rapportent des points, et atteindre la sortie valide le niveau si tous les objets requis ont été collectés.

2- Fonctionnalité

Les niveaux sont chargés dynamiquement via la classe ChargeurDreamz. Chaque niveau contient des entités structurées (joueur, ennemis, objets interactifs), gérées par un système unifié basé sur des métadonnées Babylon. Le HUD est mis à jour à chaque frame pour afficher le score, le temps et les objets collectés.

Les ennemis sont contrôlés par une IA qui utilise Recast.js pour naviguer sur une grille de navigation. Ils patrouillent ou poursuivent le joueur selon sa position. Leur

comportement est mis à jour en temps réel avec gestion de l'animation et de la direction.

Les collisions sont activées globalement dans la scène. Des observateurs détectent les interactions entre le joueur et les objets : bonus, ennemis, ou sortie. Les bonus sont supprimés à la collecte et déclenchent une mise à jour des points. Le joueur est téléporté ou éliminé s'il tombe ou touche un ennemi.

L'interface est gérée par MenuDreamz, qui lance le jeu et les niveaux. Une skybox personnalisée et une caméra arcrotate assurent l'ambiance visuelle et le confort de navigation. La lumière et les matériaux sont simples pour garantir la lisibilité et la fluidité du rendu.

Le jeu repose sur un système de progression personnalisé, associé à chaque utilisateur via son identifiant. Les niveaux débloqués et les scores sont enregistrés dans le localStorage sous une clé unique par utilisateur. Seules les meilleures performances par niveau sont conservées, et le score total est calculé dynamiquement. La classe GestionPoints centralise cette logique.

Les niveaux sont construits dynamiquement à partir d'une matrice de 1s et de 0s, chaque case indiquant s'il y a du sol, ou non. Un système d'entités génériques gère les objets du jeu : joueur, ennemis, obstacles et sorties. Chaque entité possède des propriétés de position, de mouvement et de comportement spécifiques. Les niveaux débloqués sont gérés par le menu et stockés dans le localStorage également.

L'IA des ennemis repose sur Recast.js pour la navigation, avec une logique de patrouille entre des points définis et une détection du joueur déclenchant la poursuite. Ces entités utilisent un système de foule pour optimiser les déplacements et les interactions dans l'espace 3D. Chaque ennemi a des points de patrouille propres.

Les collisions sont gérées avec moveWithCollision : le joueur ne passe pas à travers les éléments du décor. De plus, chaque entité voit ses collisions gérées à l'aide d'une hitbox et d'un écouteur de collisions.

Il y a un HUD et un menu, gérés avec BABYLON.GUI. Le HUD affiche le niveau actuel, le nombre de points récupérés sur le niveau actuel, le temps passé et le nombre de clés à récupérer.

3- Choix techniques

Le système repose sur des modules ES6, facilitant la structure du code et la séparation des responsabilités (chargement, menu, HUD, entités, IA). La gestion du temps et du score est centralisée, tandis que les entités du niveau (joueur, ennemis, objets) utilisent une architecture orientée objet avec métadonnées intégrées à leurs

meshes Babylon. L'utilisation d'une seule scène évite des temps de chargement inutiles et permet de conserver des éléments communs entre niveaux.

4- Pistes d'amélioration

La première amélioration qu'il faudrait mettre mais pour laquelle nous avons manqué de temps, c'est de mettre de la musique et des effets sonores pour le jeu, et qui y soient réactifs.

Le jeu pourrait être enrichi avec des niveaux multi-étages intégrant des plateformes mobiles, des ascenseurs ou des mécanismes de téléportation. Cela offrirait une dimension verticale, plus de diversité dans les mécaniques et des situations de navigation plus complexes pour l'IA. Une refonte partielle du système de pathfinding serait nécessaire pour prendre en compte la verticalité. D'autres types d'ennemis ou d'objets interactifs (pièges, interrupteurs, portes temporisées) pourraient également être envisagés.

Difficultés rencontrées

Les difficultés rencontrées lors de la mise en place de notre projet vont du niveau global au niveau individuel. De façon générale, notre équipe a été confrontée à plusieurs obstacles, notamment liés à :

- Conflits de fusions (merge conflits)
- Mauvaise synchronisation des dépôts : oublie de faire push ou commit
- Mauvaise coordination par endroit
- Adapter les conceptions communes à chaque jeu.

Personnellement les difficultés rencontrées sont les suivantes :

- Jeu 1 : animation des cookies qui tombent
- Jeu 2 : concevoir une boucle de jeu fluide pour gérer les mouvements, animations et mise à jour en temps réel ; optimiser les performances du jeu
- Jeu 3 : mettre les textures sur le décor, faire le déploiement, pour la gestion de la mémoire aussi la création des entités.