UNIVERSITÉ CÔTE D'AZUR/DS4H

MIAGE – LICENCE 3 MIASHS parcours MIAGE

1645 Rte des Lucioles, 06410 Biot

Rapport de Conception

PROJET INFORMATIQUE - JEUX JAVASCRIPT

Réalisé par :

Axel GUILLOU, Alexander BORETTI, Nathan BITOUN

Encadrant :

Monsieur Michel BUFFA - Monsieur Othman Mekouar

Introduction du Projet

Objectifs

Ce projet a pour but de concevoir une plateforme web regroupant trois mini-jeux développés en JavaScript. Chaque jeu possède des mécaniques spécifiques et une interface utilisateur adaptée. Le tout est pensé pour offrir une expérience ludique, intuitive, et modulable.

Présentation des Jeux

Candy Game : Jeu inspiré de Candy Crush. Le joueur aligne des cookies identiques pour marquer des points.

Bounce : Jeu d'adresse basé sur la physique, dans lequel une balle doit rebondir et monter le plus haut possible en passant les obstacles.

Cube Invasion : Jeu de plateforme avec ombres et déplacements tactiques, où le joueur doit traverser des zones piégées et éviter des ennemis.

Répartition du Travail

Axel GUILLOU : Développement de Candy Game, et du site

Alexander BORETTI : Développement de Bounce

Nathan BITOUN : Développement de Cube Invasion

Technologies Utilisées

Frontend : HTML5, CSS3, JavaScript

Stockage local : local Storage pour les scores et la progression

Nous avons utilisé plusieurs façon de faire nos jeux en JavaScript :

- Bounce un jeu en canva
- Cube Invasion un jeu en canva
- Candy Game en jeu en DOM

Détail par Jeu

Candy Game

Introduction du Jeu

Candy Game est un jeu style puzzle développé en DOM. Le joueur doit aligner au moins trois cookies identiques horizontalement ou verticalement pour marquer des points. Le jeu se déroule sur une grille de 9x9 cases et propose un système de progression par niveaux avec un chronomètre.

L'objectif principal est d'atteindre le score cible dans le temps imparti. Chaque nouveau niveau augmente la difficulté en ajoutant un objectif de score.

Architecture du code

index.html : Structure de la page, conteneurs du jeu

BoucleJeu.js : Initialisation du jeu, gestion des écrans, démarrage

grille.js : Logique de jeu : grille, alignements, niveaux, affichage

cookie.js : Représentation d'un cookie (élément du jeu)

utils.js : Fonctions utilitaires (tableau 2D, etc.)

Structure du jeu

Modèle de données

Une Grille de dimensions 9x9 contient des objets Cookie.

Chaque Cookie est défini par :

un type (représenté par une image),

des coordonnées ligne et colonne,

une référence à son élément DOM (htmlImage),

un état de sélection ou d'alignement.

État du joueur

Un objet joueur contient :

nom : saisi au lancement,

score : mis à jour dynamiquement,

niveau : augmente si le score seuil est atteint.

Événements utilisateurs

Click : sélection de cookie.

Drag & Drop : déplacement et échange de cookies.

Boutons : démarrage, affichage des règles, rejouer.

Fonctionnalités principales

Affichage dynamique de la grille avec les images.

Sélection et échange de deux cookies voisins.

Détection des alignements horizontaux et verticaux.

Suppression en cascade avec effet de descente et remplissage.

Système de niveaux avec chrono et difficulté croissante.

Stockage local des meilleurs scores avec podium.

BOUNCE

Introduction du Jeu

BOUNCE est un jeu d'arcade 2D à défilement vertical pour deux joueurs en écran partagé. Chaque joueur contrôle une balle colorée progressant automatiquement vers le haut. L'objectif est de survivre le plus longtemps possible en franchissant des obstacles : la couleur de la balle doit correspondre à celle du segment d'obstacle touché. Le premier joueur à la ligne d'arrivée gagne.

Architecture du Projet et Scripts Clés

Le code source (src/) est structuré en modules pour la clarté et la maintenance.

Core/ : Noyau du Jeu

Logique fondamentale et moteur du jeu.

- game.js (Orchestrateur Principal): Gère la boucle de jeu (gameLoop via requestAnimationFrame), l'initialisation, les états du jeu (menu, jeu, pause, fin) et la logique de victoire. Il coordonne tous les autres modules.
- input.js: Traite les entrées clavier des joueurs (saut).
- physics.js: Implémente une physique simplifiée (gravité, saut, détection de collision basique).

Entities/ : Entités du Jeu

Objets interactifs du jeu.

- player.js (Logique du Joueur): Gère le mouvement (position, vélocité), la couleur, les collisions, le score et le dessin du joueur (et sa traînée). La couleur est cruciale pour l'interaction avec les obstacles.
- obstacles/: Types d'obstacles.
  - pulsar.js: Permet de changer de couleur en le traversant.
  - rotator.js: Obstacle rotatif avec segments colorés à traverser.

Managers/ : Gestionnaires de Systèmes

Classes pour aspects transversaux.

- colorManager.js: Gère la palette et les transitions de couleurs.
- gameState.js: suit l'état du jeu, les scores, et les conditions de victoire/défaite.
- obstacleManager.js (Gestion des Obstacles): Généré procéduralement les obstacles pour les deux joueurs et gère leur cycle de vie, assurant un défi progressif et une bonne rejouabilité.

Rendering/ : Rendu Graphique

Logique d'affichage.

- renderer.js (Moteur de Rendu): Utilise l'API HTML5 Canvas pour dessiner tous les éléments (joueurs, obstacles, fond, grille, ligne d'arrivée). Il gère les caméras par joueur et la division de l'écran. C'est le cœur du système visuel.

- ui.js (Interface Utilisateur HTML): Gère les éléments HTML superposés (menus, scores, boutons "Jouer", "Pause", "Rejouer") et leurs interactions.

Gestion de l'Écran Partagé pour Deux Joueurs

BOUNCE est conçu pour une expérience à deux joueurs sur le même écran.

- Division de l'Écran: renderer.js divise le canvas en deux sections verticales, une pour chaque joueur.
- Caméras Indépendantes: Chaque joueur a une caméra virtuelle (this.cameras[0/1]) qui suit sa progression verticale.
- Rendu Séparé: Dans renderer.render():
  1. Le contexte du canvas est sauvegardé (ctx.save()).
  2. Une région de découpage (clipping region via ctx.clip()) est définie pour la vue d'un joueur.
  3. Les transformations de caméra (ctx.translate()) sont appliquées.
  4. Les éléments (obstacles, joueur) sont dessinés dans cette région.
  5. Le contexte est restauré (ctx.restore()).
  6. Le processus est répété pour l'autre joueur sur l'autre moitié de l'écran.
- Entrées Utilisateur Distinctes: input.js associe des touches différentes à chaque joueur.

Cube Invasion

Introduction et Vision du Jeu

Cube Invasion est un jeu d'arcade 2D où le joueur incarne un héros devant survivre à des vagues infinies d'ennemis. Le gameplay repose sur des mécaniques de combat, de progression et de gestion des ressources. L'objectif est de survivre le plus longtemps possible tout en éliminant des ennemis pour accumuler de l'expérience et améliorer ses compétences. Ce document détaille l'architecture technique et les mécaniques principales du jeu.

Architecture du Projet et Scripts Clés

Le projet est structuré en modules JavaScript pour assurer une séparation claire des responsabilités et faciliter la maintenance.

Core/ : Noyau du Jeu

- gameManager.js : Gère la boucle de jeu via requestAnimationFrame, les entrées utilisateur, le spawn des ennemis, les collisions et la progression du joueur. Il coordonne les modules et adapte le jeu au framerate grâce à un delta time.

- game.js : Contient les données principales (joueur, ennemis, projectiles, cristaux d'expérience) et gère les interactions entre entités.

Entities/ : Entités du Jeu

- player.js : Gère les mouvements, attaques, collecte d'expérience et améliorations. Le joueur peut choisir des bonus à chaque montée de niveau.
- enemy.js : Définit les types d'ennemis (corps à corps, distance, guérisseur) avec des comportements spécifiques.
- projectile.js : Gère les projectiles avec trajectoire et collisions.
- xp.js et fix.js : Représentent les cristaux d'expérience et les objets de soin laissés par les ennemis.

Managers/ : Gestionnaires de Systèmes

- collision.js : Gère les collisions entre entités et applique les effets (dégâts, collecte d'XP, etc.).
- logic.js : Fournit des fonctions utilitaires pour le spawn, les comportements ennemis et autres logiques de gameplay.

Interface/ : Interface Utilisateur

- ui.js : Affiche les barres de vie et d'expérience, ainsi que les menus (démarrage, pause, fin).
- menu.js : Gère les boutons du menu principal (démarrer, scores, aide).

Technologies Mises en Œuvre

- JavaScript : Logique du jeu et interactions.
- HTML5 Canvas API : Rendu graphique temps réel.
- CSS3 : Style de l'interface utilisateur.

- LocalStorage : Sauvegarde des scores.

Adaptation au Frame Rate

Cube Invasion utilise un delta time pour une expérience fluide sur tous les appareils.

- Le delta est calculé à chaque frame et passé aux entités.
- Les vitesses sont multipliées par le delta pour des mouvements cohérents.

Mécaniques Clés et Progression

Système de Progression

Le joueur gagne de l'expérience en éliminant des ennemis. À chaque niveau, il peut choisir une amélioration :

- Santé maximale
- Dégâts
- Vitesse
- Temps de recharge réduit
- Portée d’attaque

IA des Ennemis

- Mêlée : Attaque au corps à corps
- Range : Tire à distance
- Healer : Soigne ses alliés

Gestion des Ressources

Les ennemis vaincus laissent tomber :

- Cristaux d'expérience (80 %)

- Objets de soin (5 %)

Le joueur doit les collecter pour survivre et progresser.

Possibilités d'Amélioration

- Optimisation :
    - Utiliser un système de partition spatiale (grille ou quad-tree)
    - Réduire l'usage de Math.hypot
- IA améliorée :
    - Ajouter des formations et des boss
- Sauvegarde :
    - Système de sauvegarde/chargement de progression
- Multijoueur :
    - Mode coopératif ou compétitif
- Effets visuels et sonores :
    - Animations et sons pour les actions
- Nouveau contenu :
    - Ennemis, armes, compétences, environnements variés

Avec son architecture modulaire et ses mécaniques engageantes, *Cube Invasion* constitue une base solide pour évoluer vers un jeu toujours plus complet et captivant.