Rapport de Conception Technique

Projet : Games On Web 2025

MIAGE – LICENCE 3 MIASHS parcours MIAGE

Encadré par : M. Michel BUFFA

Groupe : CHELLALI Adam - GOURAR Yanis - ROUSSEAU Noah

Jeu DOM : *Aim Miaw*

1) ORGANISATION MODULAIRE

La structure modulaire du projet est implémentée à travers plusieurs fichiers clés :

main.js : Coordonne les transitions entre les différents états du jeu
game.js : Contient la logique métier principale et la boucle de jeu
target.js : Gère la création et le comportement des cibles
ui.js : Prend en charge les mises à jour de l'interface utilisateur
events.js : Traite les interactions utilisateur et les entrées
fps.js : Surveille les performances du système

2) FLUX DE CONTROLE

Le fichier main.js implémente le pattern State Machine qui gère les transitions entre :

L'état de menu (défini dans index.html)
L'état de jeu actif (coordonné par game.js)
L'état de fin de partie (géré dans game.js et main.js)

3) MECANISMES DE JEU

a) Système de Score Avancé

Le calcul du score est principalement implémenté dans target.js, qui :

Détermine la précision du clic via des calculs géométriques
Applique des bonus différenciés selon la zone touchée
Gère les pénalités pour les cibles spéciales (20% de chance)

L'affichage du score est mis à jour par ui.js avec des effets visuels temporaires.

b) Gestion Temporelle

La gestion du temps est répartie entre :

game.js : Contient le timer principal et gère les bonus/malus
target.js : Adapte dynamiquement la durée de vie des cibles
ui.js : Affiche le temps restant

4) OPTIMISATIONS TECHNIQUES

a) Gestion du DOM

Les optimisations DOM sont visibles dans :

target.js : Crée et supprime efficacement les éléments cibles
events.js : Gère les écouteurs d'événements de manière optimale
styles.css : Utilise des sélecteurs performants

b) Performances Graphiques

Les performances sont boostées par :

styles.css : Utilise transform/opacity pour les animations
target.js : Implémente le rétrécissement progressif des cibles
ui.js : Limite les mises à jour du DOM

5) SYSTEME DE RENDU

a) Couche Visuelle

La présentation visuelle est définie dans :

styles.css : Contient toutes les règles de style responsive
index.html : Structure les éléments d'interface
target.js : Gère l'apparence des différents types de cibles

b) Feedback Utilisateur

Le feedback est implémenté via :

target.js : Génère les effets d'impact visuel
ui.js : Affiche les messages temporaires
styles.css : Anime les éléments de feedback

6) INTEGRATION AUDIO

a) Gestion du Son

L'audio est configuré dans :

index.html : Déclare les éléments audio
target.js : Déclenche les sons lors des interactions

b) Synchronisation

La synchronisation est gérée par :

target.js : Réinitialise le curseur audio avant lecture
styles.css : Synchronise les animations avec les effets sonores

Jeu CANVA : *Ball 2 Goal*

1) ORGANISATION MODULAIRE

Le projet est structuré en modules spécialisés qui interagissent via des interfaces claires :

game.js : Module principal qui orchestre le jeu et coordonne les autres composants
engine.js : Moteur physique basé sur Matter.js avec gestion du rendu
players.js : Gestion des joueurs (mouvements, physique, contrôles)
ball.js : Implémentation du comportement de la balle
goals.js : Système de détection de buts et gestion des cages
score.js : Gestion du score, chronomètre et fin de partie
input.js : Gestion des entrées clavier et état de pause
ui.js : Interface utilisateur et menus

2) FLUX DE CONTROLE

Le flux principal est géré par la classe Game dans game.js qui :

Initialise tous les sous-systèmes au démarrage
Gère la boucle de jeu via Matter.js Runner
Coordonne les transitions entre états (menu, jeu actif, pause, fin de partie)
Écoute les événements de collision pour détecter les buts

3) MECANISMES DE JEU

a) Système Physique

Implémenté dans engine.js avec Matter.js, il gère :

Les corps rigides (joueurs, balle, murs)
Les collisions et leurs résolutions
Les forces appliquées (coups, sauts)
Les propriétés matérielles (restitution, friction)

b) Gestion Temporelle

Gérés par input.js avec :

Configuration des touches pour chaque joueur
Gestion de l'état des touches (pressée/relâchée)
Application des forces aux joueurs en fonction des entrées
Restrictions directionnelles pour les tirs

c) Système de Score et Chronomètre

Implémenté dans score.js avec :

Comptage des buts par équipe
Chronomètre dégressif avec pause
Affichage dynamique des scores
Gestion de la fin de partie (victoire/nul)
Effets sonores et visuels pour les buts

4) OPTIMISATIONS TECHNIQUES

a) Gestion des Événements

Collisions détectées via les événements Matter.js
Écouteurs d'entrée optimisés (un seul pour toutes les touches)
Gestion centralisée des événements UI dans UIManager

b) Performances Graphiques

Utilisation de sprites pour les joueurs et la balle
Rendu Canvas optimisé via Matter.Render
Éléments UI positionnés en CSS (pas dans le canvas)
Animations CSS pour les effets visuels

c) Gestion Mémoire

Réutilisation des corps physiques (pas de recréation)
Nettoyage des écouteurs d'événements
Gestion centralisée des ressources audio

5) SYSTEME DE RENDU

a) Couche Visuelle

Canvas : Pour les éléments physiques (joueurs, balle, terrain)
HTML/CSS : Pour l'interface utilisateur (scores, menus, messages)
Sprites : Images pour les joueurs et la balle avec scaling approprié
Effets Visuels : Ombres portées, animations pour les messages

b) Feedback Utilisateur

Messages de but avec effets visuels
Indicateurs de score mis à jour en temps réel
Boutons UI avec états visuels
Sons pour les actions importantes (buts, coups)

6) INTEGRATION AUDIO

a) Gestion du Son

Sons courts pour les impacts (kick.mp3)
Son distinctif pour les buts (goal.mp3)
Gestion via l'API Web Audio (éléments)

b) Synchronisation

Sons déclenchés lors des événements physiques
Synchronisation avec les effets visuels
Réinitialisation des pistes audio avant lecture

c) Points Forts de l'Architecture

Découpage Modulaire : Chaque composant a une responsabilité claire
Gestion Centralisée des États : Via la classe Game
Physique Réaliste : Grâce à Matter.js bien configuré
UI Flexible : Séparation claire entre éléments de jeu et interface
Système d'Entrée Robustes : Gestion des touches et de la pause efficace

d) Améliorations Possibles

Pool d'Objets : Pour les effets visuels temporaires
Système de Particules : Pour les impacts et buts
Compression Audio : Formats plus légers pour les sons
Optimisation Mobile : Contrôles tactiles alternatifs
État Replay : Pour revoir les buts marqués

Cette architecture modulaire et bien organisée permet des extensions futures tout en maintenant des performances optimales pour un jeu 2D physique en temps réel.

Jeu BABYLON JS : Starfall

1) ORGANISATION MODULAIRE

Le projet est structuré en modules répartis côté client et serveur :

client/ : Modules séparés pour les vaisseaux, planètes, interface utilisateur (UI), et gestion des contrôles.

server/ : Utilise Node.js avec WebSocket (ws) pour gérer la logique de jeu synchrone, les collisions et les scores.
worker.js + bullet.js : Gèrent les projectiles dans un thread parallèle via WebWorkers.

2) FLUX DE CONTROLE

Le client utilise Babylon.js pour le rendu 3D en temps réel.
Les interactions multijoueur sont gérées via WebSocket :
Chaque action (tir, déplacement) est transmise au serveur.
Le serveur envoie les mises à jour aux autres clients.
Le score est envoyé au serveur à chaque élimination.

3) MECANISMES DE JEU

Mouvement :
Inertie et accélération (touche E).
Rotation libre avec la souris.

Gravité Dynamique :
Les planètes et étoiles influencent les déplacements.
Une alerte visuelle s'active en zone dangereuse.

Combat :
Tirs de projectiles (touche Espace).
Effets visuels associés (glow, particules).

Système de Score et Interface :
Score envoyé au serveur à chaque élimination.

UI :
Vues dynamiques : cockpit ou troisième personne (touche V).
Radar : montre les vaisseaux ennemis à proximité.
Canvas d'info : FPS, coordonnées, nombre d'objets (touche X).
Santé : barre de vie avec cœur clippé et animation des dégâts récents.

Optimisations Techniques
WebWorkers : utilisés pour la gestion parallèle des projectiles (bullet.js + worker.js).
Level of Detail : réduit la fréquence de mise à jour pour les objets éloignés.
Effets visuels : particules optimisées, GlowLayer pour les projectiles.

4) SYSTEME DE RENDU

Babylon.js : moteur de rendu 3D WebGL.

Environnement :
Génération procédurale des planètes et étoiles avec textures PBR.
Skydome réaliste avec fond galactique.

Feedback Utilisateur
Effets visuels lors des tirs et impacts.
Barre de vie animée lors des dégâts récents.

Intégration Audio
Sons gérés via éléments audio dans index.html.

Sons déclenchés par les interactions (ex. tirs).
Réinitialisation du curseur audio avant lecture.
Synchronisation des effets sonores avec les animations via styles.css.

Points Forts de l'Architecture
Multijoueur en temps réel via WebSocket.
Mécaniques de gravité réalistes.
Interface immersive avec vues dynamiques.
Architecture modulaire et scalable.

Améliorations Possibles
Ajout de power-ups (boucliers, armes spéciales).
Mode équipe avec objectifs coopératifs.
Optimisation mobile (contrôles tactiles).