**Rapport de Conception Technique :**

***Neo-Clicker***

**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

**Flux de Contrôle**  
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)

**Mécanismes de Jeu**  
**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.

**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

**Optimisations Techniques**  
**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

**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

**Système de Rendu**  
**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

**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

**Intégration Audio**  
**Gestion du Son**  
L'audio est configuré dans :

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

**Synchronisation**  
La synchronisation est gérée par :

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

***HEADBALL***   
**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

**Flux de Contrôle**  
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

**Mécanismes de Jeu**  
**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)

**Contrôles des Joueurs**  
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

**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

**Optimisations Techniques**  
**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

**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

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

**Système de Rendu**  
**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

**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)

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

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

**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

**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.

**Starfall**

Starfall est un jeu spatial multijoueur en 3D temps réel, développé avec Babylon.js et WebSocket. Il combine des mécaniques de combat spatial, une gravité dynamique, et une interface immersive pour offrir une expérience de jeu fluide et compétitive.

**Technologies Utilisées**

**Frontend** : JavaScript (ES6+), Babylon.js (moteur 3D WebGL), HTML5/CSS3.

**Backend** : Node.js, Express.js, WebSocket (ws).

**Optimisation** : WebWorkers pour la gestion des projectiles et particules.

**Fonctionnalités Clés**

**Multijoueur en Temps Réel** :

Connexion via WebSocket pour des combats synchronisés entre joueurs.

Score envoyé au serveur à chaque élimination.

**Mécaniques de Jeu** :

**Mouvement** : Inertie, accélération (touche E), et rotation libre (souris).

**Gravité** : Planètes et étoiles influencent les déplacements (alerte visuelle activée en zone dangereuse).

**Combat** : Tirs de projectiles (touche Espace) avec effets visuels (glow, particules).

**Interface Utilisateur** :

**Vues Dynamiques** : Basculable entre cockpit et troisième personne (touche V).

**Radar** : Affiche les vaisseaux ennemis à proximité.

**Canvas d'Info** : FPS, coordonnées, nombre d'objets (touche X).

**Santé** : Barre de vie avec cœur clippé et dégâts récents animés.

**Environnement** :

**Planètes/Étoiles** : Génération procédurale avec textures PBR. Les étoiles émettent lumière et particules.

**Skydome** : Fond galactique réaliste.

**Optimisations**

**WebWorkers** : Gestion parallèle des projectiles (bullet.js + worker.js) pour des performances stables.

**Level of Detail** : Mises à jour moins fréquentes pour les objets éloignés.

**Effets Visuels** : GlowLayer pour les projectiles, particules optimisées.

**Structure du Projet**

**Client** : Modules séparés pour les vaisseaux, planètes, UI, et gestion des contrôles.

**Serveur** : Logique de jeu synchrone via WebSocket, avec gestion des collisions et scores.

**Points Forts**

Immersion grâce aux vues dynamiques et effets sonores.

Mécaniques de gravité réalistes.

Architecture modulaire et scalable.

**Améliorations Possibles**

Ajout de power-ups (boucliers, armes spéciales).

Mode équipe avec objectifs coopératifs.

Optimisation mobile (contrôles tactiles).