PROJET TRON - PLANIFICATION DÉTAILLÉE (VERSION RÉVISÉE)

Informations Générales			Jalons Clés		
Date de début	30 octobre 2025		**Sprint 1**	31 oct - 6 nov	Architecture + Schemas JSON + Base
Objectif MVP (v1.0)	1 décembre 2025		**Sprint 2**	7 nov - 13 nov	Backend + Persistance + Doc API
Date finale	À définir (soutenance)		**Sprint 3**	14 nov - 20 nov	Logique jeu + WebSocket + Doc
Durée MVP	32 jours		**Sprint 4**	21 nov - 27 nov	Interface client + Tests + Doc
Nombre de membre	3-4 personnes		**Sprint 5**	28 nov - 1 déc	Intégration + Tests + MVP
Type de projet	Application Web Mobile + Serveur		**Phase finale**	2 déc - Soutena	Rapport complet + Soutenance

Stack Technique	
IMPORTANT : Documentation continue	
Chaque sprint inclut des tâches de documentation. Les JSON schemas doivent être créés AVANT le code et maintenus à jour tout au long du projet.	
Base de données	MongoDB + Mongoose
Communication	WebSocket (Socket.io)
Format de données	JSON avec schemas de validation
Documentation	JSON Schemas + JSDoc continue

ARCHITECTURE TECHNIQUE (SANS EXPRESS)

ARCHITECTURE CLIENT (Cordova)

Composant	Technologie	Rôle	Fichiers
Interface UI	HTML5/CSS3	Écrans du jeu	index.html, login.html, game.html, lobby.html
Logique Client	JavaScript ES6+	Gestion d'état client	app.js, gameClient.js, stateManager.js
Communication	Socket.io Client	Temps réel avec serveur	socketClient.js
Rendu Canvas	HTML5 Canvas	Affichage grille de jeu	gameRenderer.js, drawEngine.js
Gestion inputs	Touch/Keyboard Events	Contrôles joueur	inputController.js
Stockage local	LocalStorage	Pseudo du joueur	storageService.js

ARCHITECTURE SERVEUR (Node.js - HTTP natif)

Composant	Technologie	Rôle	Fichiers
HTTP Server	http (natif Node.js)	Serveur HTTP simple	server.js
WebSocket Server	Socket.io	Communication temps réel	socketServer.js
Game Manager	JavaScript	Gestion parties multiples	gameManager.js
Game Logic	JavaScript	Moteur de jeu Tron	gameLogic.js, collisionDetector.js
Player Manager	JavaScript	Gestion joueurs/pseudos	playerManager.js
Database Layer	Mongoose	ORM MongoDB	models/Player.js, models/Game.js
Validation Layer	Ajv (JSON Schema)	Validation données entrantes	validators/schemas.js
Config Manager	JavaScript	Configuration serveur	config/gameConfig.js

ÉVÉNEMENTS WEBSOCKET

Événement	Direction	Payload (JSON)	Description
player:register	Client → Serveur	{pseudo: string}	Enregistrer un pseudo
player:registered	Serveur → Client	{playerId: string, pseudo: string}	Confirmation enregistrement
lobby:join	Client → Serveur	{playerId: string}	Rejoindre le lobby
lobby:status	Serveur → Client	{playersWaiting: number, maxPlayers: number}	État du lobby
game:start	Serveur → Tous	{gameId: string, players: Array, gridSize: Object}	Début de partie
player:direction	Client → Serveur	{playerId: string, direction: string}	Changement direction
game:state	Serveur → Tous	{tick: number, positions: Object, trails: Object}	État du jeu (60fps)

player:eliminated	Serveur → Tous	{playerId: string, reason: string}	Joueur éliminé	
game:end	Serveur → Tous	{winner: Object, scores: Array}	Fin de partie	
scores:request	Client → Serveur	{limit: number}	Demande high scores	
scores:response	Serveur → Client	{scores: Array}	Réponse high scores	

JSON SCHEMAS DÉTAILLÉS (À CRÉER EN SPRINT 1)

Ces schemas doivent être créés AVANT le code et maintenus dans /docs/schemas/

1. PLAYER REGISTRATION REQUEST

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "$id": "PlayerRegistration",
  "title": "Player Registration",
  "description": "Schema pour l'enregistrement d'un nouveau joueur",
  "type": "object",
  "properties": {
    "pseudo": {
      "type": "string",
      "minLength": 3,
      "maxLength": 20,
      "pattern": "^[a-zA-Z0-9_-]+$",
      "description": "Pseudo du joueur (alphanumerique, tirets, underscores)"
    }
  }
```

2. GAME STATE UPDATE

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "$id": "GameState",
  "title": "Game State",
  "description": "État complet du jeu à chaque tick",
  "type": "object",
  "properties": {
    "gameId": {
      "type": "string",
      "description": "ID unique de la partie"
    },
    "tick": {
      "type": "integer",
      "minimum": 0,
      "description": "Numéro du tick courant"
    },
    "timestamp": {
      "type": "integer",
      "description": "Timestamp Unix en millisecondes"
    },
    "players": {
      "type": "object",
      "patternProperties": {
        "^[a-f0-9]{24}$": {
          "type": "object",
          "properties": {
            "position": {
```

3. PLAYER DIRECTION CHANGE

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "$id": "DirectionChange",
  "title": "Direction Change",
  "description": "Changement de direction d'un joueur",
  "type": "object",
  "properties": {
    "playerId": {
      "type": "string",
      "pattern": "^[a-f0-9]{24}$",
      "description": "ID MongoDB du joueur"
    },
    "direction": {
      "type": "string",
      "enum": ["UP", "DOWN", "LEFT", "RIGHT"]
```

4. HIGH SCORES RESPONSE

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "$id": "HighScores",
  "title": "High Scores",
  "description": "Liste des meilleurs scores",
  "type": "object",
  "properties": {
    "scores": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "pseudo": {
            "type": "string",
            "minLength": 3,
            "maxLength": 20
          },
          "gamesWon": {
            "type": "integer"
```

INSTRUCTIONS POUR LES SCHEMAS

FICHIERS À CRÉER:

```
├── docs/
│   └── schemas/
│       ├── README.md (explication des schemas)
│       ├── player-registration.schema.json
│       ├── game-state.schema.json
│       ├── direction-change.schema.json
│       ├── high-scores.schema.json
│       └── server-config.schema.json
```

UTILISATION DANS LE CODE:

```
const Ajv = require('ajv');
const ajv = new Ajv();
const schema = require('./docs/schemas/player-registration.schema.json');
const validate = ajv.compile(schema);
```

MODÈLE DE DONNÉES MONGODB

Collection: players

Champ	Type	Index	Validation	Description
_id	ObjectId	Oui (PK)	Auto	ID auto-généré MongoDB
pseudo	String	Unique	required, minlength: 3, maxlength: 20, match: /^[a-zA-Z0-9_-]+$/	Pseudo unique du joueur
gamesPlayed	Number	Non	default: 0, min: 0	Nombre total de parties jouées
gamesWon	Number	Non	default: 0, min: 0	Nombre de victoires
createdAt	Date	Non	default: Date.now	Date de création du compte
lastSeen	Date	Non	default: Date.now	Dernière connexion

Collection: games

Champ	Type	Index	Validation	Description
_id	ObjectId	Oui (PK)	Auto	ID unique de la partie
players	Array[ObjectId]	Non	ref: 'Player', required	Références vers collection players
winner	ObjectId	Non	ref: 'Player'	Référence au joueur gagnant (null si aucun)
gridSize	Object	Non	required: {width: Number, height: Number}	Dimensions de la grille {width: 50, height: 50}
startTime	Date	Oui	required, default: Date.now	Timestamp de début de partie
endTime	Date	Non	default: null	Timestamp de fin (null si en cours)
status	String	Oui	enum: ['waiting', 'active', 'finished'], required	État de la partie

maxPlayers	Number	Non	default: 4, min: 2, max: 8	Nombre maximum de joueurs pour cette partie
gameSpeed	Number	Non	default: 100, min: 50, max: 500	Vitesse du jeu en ms par tick

Collection: serverConfig (singleton)

Champ	Type	Index	Validation	Description
_id	ObjectId	Oui	Singleton (1 seul doc)	ID de configuration (unique document)
gridWidth	Number	Non	default: 50, min: 20, max: 200	Largeur de la grille de jeu
gridHeight	Number	Non	default: 50, min: 20, max: 200	Hauteur de la grille de jeu
maxPlayersPerGame	Number	Non	default: 4, min: 2, max: 8	Nombre max de joueurs par partie
gameSpeed	Number	Non	default: 100, min: 50, max: 500	Vitesse du jeu (ms entre chaque tick)
matchmakingTimeout	Number	Non	default: 30000, min: 5000	Timeout du matchmaking en ms
updatedAt	Date	Non	default: Date.now	Dernière modification de la config

INDICES MONGODB ESSENTIELS

Collection	Champ(s)	Type	Commande MongoDB	
players	pseudo	Unique	db.players.createIndex({pseudo: 1}, {unique: true})	
games	status, startTime	Compound	db.games.createIndex({status: 1, startTime: -1})	
games	startTime	Simple (desc)	db.games.createIndex({startTime: -1})	

PLANNING DÉTAILLÉ DES TÂCHES (Documentation continue intégrée)							
ID	Tâche	Sprint	Responsable	Durée (j)	Début	Fin	Dépendances
1.1	Setup environnement Node.js + MongoDB	1	Backend Dev	1	31 oct	31 oct	-
1.2	Initialiser projet Cordova	1	Frontend Dev	1	31 oct	31 oct	-
1.3	Créer tous les JSON schemas (voir onglet)	1	Tous	2	31 oct	1 nov	-
1.4	Documenter architecture technique	1	Tous	1	1 nov	1 nov	1.3
1.5	Définir modèles Mongoose + validation	1	Backend Dev	2	2 nov	3 nov	1.1, 1.3
1.6	Créer serveur HTTP natif Node.js	1	Backend Dev	1	2 nov	2 nov	1.1
1.7	Setup Socket.io (client + serveur)	1	Full Stack	2	4 nov	5 nov	1.2, 1.6
1.8	Documenter API WebSocket (événements)	1	Full Stack	1	6 nov	6 nov	1.7
1.9	Structure de dossiers projet	1	Tous	0.5	31 oct	31 oct	-
2.1	Implémenter modèle Player (Mongoose)	2	Backend Dev	1	7 nov	7 nov	1.5
2.2	Implémenter modèle Game (Mongoose)	2	Backend Dev	1	7 nov	7 nov	1.5
2.3	Implémenter modèle ServerConfig	2	Backend Dev	1	8 nov	8 nov	1.5
2.4	PlayerManager: enregistrement pseudo	2	Backend Dev	2	9 nov	10 nov	2.1
2.5	Fonction getHighScores avec agrégation	2	Backend Dev	2	9 nov	10 nov	2.1
2.6	Tests persistance MongoDB	2	Backend Dev	1	11 nov	11 nov	2.4, 2.5
2.7	Documenter modèles de données (JSDoc)	2	Backend Dev	1	12 nov	12 nov	2.3
2.8	Guide de déploiement MongoDB	2	Backend Dev	1	13 nov	13 nov	2.6
2.9	Écran d'enregistrement pseudo (UI)	2	Frontend Dev	2	7 nov	8 nov	1.2
2.10	Intégration Socket.io enregistrement	2	Frontend Dev	2	9 nov	10 nov	1.7, 2.9
3.1	GameManager: création/gestion parties	3	Backend Dev	2	14 nov	15 nov	2.2
3.2	GameLogic: initialisation grille	3	Backend Dev	1	14 nov	14 nov	3.1
3.3	GameLogic: mouvement des motos	3	Backend Dev	2	15 nov	16 nov	3.2
3.4	CollisionDetector: détection collisions	3	Backend Dev	2	17 nov	18 nov	3.3
3.5	Gestion fin de partie & détermination gagnant	3	Backend Dev	1	19 nov	19 nov	3.4
3.6	Système de matchmaking automatique	3	Backend Dev	2	14 nov	15 nov	3.1
3.7	Implémentation tous événements WebSocket	3	Full Stack	3	16 nov	18 nov	1.8, 3.1
3.8	Game loop (60 ticks/sec)	3	Backend Dev	2	18 nov	19 nov	3.3, 3.7
3.9	Documenter logique de jeu	3	Backend Dev	1	20 nov	20 nov	3.5
3.10	Tests unitaires logique de jeu	3	Backend Dev	1	20 nov	20 nov	3.5
4.1	Écran lobby (attente de joueurs)	4	Frontend Dev	2	21 nov	22 nov	2.10
4.2	Canvas de jeu (grille)	4	Frontend Dev	2	21 nov	22 nov	2.10
4.3	DrawEngine: rendu motos et trails	4	Frontend Dev	2	23 nov	24 nov	4.2

4.4	InputController: contrôles tactiles	4	Frontend Dev	2	23 nov	24 nov	4.2
4.5	InputController: contrôles clavier	4	Frontend Dev	1	24 nov	24 nov	4.4
4.6	Intégration WebSocket client (tous événements)	4	Frontend Dev	2	25 nov	26 nov	3.7, 4.3
4.7	StateManager: gestion état local	4	Frontend Dev	2	25 nov	26 nov	4.6
4.8	Écran high scores (UI)	4	Frontend Dev	1	27 nov	27 nov	2.5
4.9	📄 Documenter composants frontend	4	Frontend Dev	1	27 nov	27 nov	4.6
4.10	Tests intégration client-serveur	4	Tous	1	27 nov	27 nov	4.6
5.1	Tests multi-joueurs (2-4 joueurs)	5	Tous	1	28 nov	28 nov	4.10
5.2	Optimisation performance serveur	5	Backend Dev	1	29 nov	29 nov	5.1
5.3	Optimisation rendu client (60fps)	5	Frontend Dev	1	29 nov	29 nov	5.1
5.4	Correction bugs critiques	5	Tous	1	30 nov	30 nov	5.1
5.5	Tests sur différents smartphones	5	Frontend Dev	1	30 nov	30 nov	5.3
5.6	Gestion déconnexions/reconnexions	5	Full Stack	1	30 nov	30 nov	5.4
5.7	Packaging APK Cordova	5	Frontend Dev	1	1 déc	1 déc	5.5
5.8	📄 Compléter documentation déploiement	5	Backend Dev	1	1 déc	1 déc	5.2
5.9	📄 README complet (install + run)	5	Tous	0.5	1 déc	1 déc	5.8
5.10	Validation MVP complète	5	Tous	0.5	1 déc	1 déc	5.7, 5.9
6.1	📄 Rédaction rapport technique complet	6	Tous	3	2 déc	À définir	5.10
6.2	📄 Diagrammes d'architecture finaux	6	Backend Dev	1	2 déc	À définir	5.10
6.3	📄 Documentation API complète	6	Full Stack	1	2 déc	À définir	5.10
6.4	📄 Section découpage des tâches (rapport)	6	Tous	1	2 déc	À définir	6.1
6.5	📄 Justification des choix techniques	6	Tous	1	2 déc	À définir	6.1
6.6	Préparation présentation soutenance	6	Tous	2	À définir	À définir	6.1
6.7	Répétition soutenance + démo	6	Tous	1	À définir	À définir	6.6
6.8	Préparation archive ZIP finale	6	Tous	1	À définir	À définir	6.1

ORGANISATION EN SPRINTS (avec documentation continue)				
Sprint	**Dates**	**Objectif principal**	**Livrables techniques**	**Livrables documentation**
Sprint 1	31 oct - 6 nov	Fondations + Schemas	• Serveur HTTP Node.js • Socket.io opérationnel • Modèles Mongoose définis	JSON schemas complets Architecture documentée API WebSocket spécifiée
Sprint 2	7 nov - 13 nov	Backend & Persistance	• MongoDB opérationnel • Enregistrement pseudo • High scores fonctionnel	Modèles documentés (JSDoc) Guide déploiement MongoDB Tests de persistance
Sprint 3	14 nov - 20 nov	Logique du jeu	• Moteur de jeu Tron • Détection collisions • Matchmaking • Game loop 60fps	Logique de jeu documentée Tests unitaires Algorithmes expliqués
Sprint 4	21 nov - 27 nov	Interface & intégration	• Interface complète • Contrôles fonctionnels • Client-serveur intégré	Composants frontend doc Tests d'intégration Guide utilisation

Sprint 5	28 nov - 1 déc	MVP & optimisation	• Tests multi-joueurs • Optimisation perf • APK généré	Documentation déploiement README complet Guide d'installation
Sprint 6	2 déc - Soutenance	Rapport & soutenance	• Archive ZIP finale • Démonstration préparée	Rapport technique Diagrammes finaux Justifications choix Découpage tâches

PLAN DE DOCUMENTATION CONTINUE

La documentation doit être créée en parallèle du code, pas après !

Dossier/Fichier	Contenu	Créé dans Sprint	Maintenu dans
/docs/schemas/	Tous les JSON schemas	Sprint 1	Tous
/docs/architecture/	Diagrammes d'architecture	Sprint 1	Sprint 6
/docs/api/	Documentation WebSocket API	Sprint 1	Sprint 3-4
/docs/deployment/	Guides de déploiement	Sprint 2	Sprint 5
/docs/database/	Modèles MongoDB détaillés	Sprint 2	Sprint 2
README.md	Installation et utilisation	Sprint 1	Sprint 5
CHANGELOG.md	Historique des modifications	Sprint 1	Tous
Code comments	JSDoc sur toutes les fonctions	Inline	Tous
/docs/game-logic/	Explication algorithmes	Sprint 3	Sprint 3
/docs/testing/	Stratégie et résultats tests	Sprint 4	Sprint 5
/docs/report/	Rapport technique final	Sprint 6	Sprint 6

BONNES PRATIQUES DE DOCUMENTATION

✅ FAIRE:
• Écrire les JSON schemas AVANT le code
• Documenter chaque fonction avec JSDoc
• Commenter les parties complexes
• Mettre à jour CHANGELOG.md à chaque modification
• Créer des diagrammes pour l'architecture
• Expliquer les algorithmes de collision
• Documenter les événements WebSocket
• Versionner la documentation avec Git

❌ NE PAS FAIRE:
• Documenter à la fin du projet
• Copier-coller sans adapter
• Oublier de mettre à jour après modifs
• Documenter l'évident
• Utiliser un jargon incompréhensible

📝 OUTILS RECOMMANDÉS:
• JSDoc pour JavaScript
• Markdown pour README
• draw.io ou Lucidchart pour diagrammes
• JSON Schema pour validation

RÉPARTITION DES RÔLES & RESPONSABILITÉS

Suggestion de répartition pour 3-4 personnes

Rôle	Responsabilités	Tâches principales	Compétences requises
Backend Developer (1 personne)	• Architecture serveur • Base de données • Logique de jeu • JSON schemas	• Setup Node.js/MongoDB • Serveur HTTP natif • Modèles Mongoose • GameManager & logique • Détection collisions • Validation schemas	• Node.js natif (http) • MongoDB/Mongoose • Socket.io • JSON Schema • Algorithmique
Frontend Developer (1 personne)	• Interface utilisateur • Application Cordova • Rendu du jeu • UX/UI	• Setup Cordova • Écrans (pseudo, lobby, jeu) • Canvas & rendu • Contrôles tactiles/clavier • Packaging APK	• HTML5/CSS3/JavaScript • Cordova • Canvas API • Responsive design • Touch events
Full Stack Developer (1 personne)	• Communication • WebSocket • Intégration • Tests • Documentation	• Socket.io (client & serveur) • Événements temps réel • Tests d'intégration • Documentation API • Optimisation perf • README et guides	• JavaScript fullstack • WebSocket/Socket.io • Debugging • Tests • Rédaction technique

Chef de projet (optionnel, 4ème personne)	• Coordination • Documentation • Rapport • Planning	• Suivi avancement • Rapport technique • JSON schemas • Documentation continue • Préparation soutenance • Support développeurs	• Rédaction • Organisation • Communication • JSON Schema • Connaissances tech

Note importante

TOUS les membres doivent contribuer à la documentation continue ! Les JSON schemas sont créés collectivement dans le Sprint 1. Cette répartition est flexible - le travail collaboratif est essentiel.

CHECKLIST AVANT RENDU FINAL		
Catégorie	**Élément à vérifier**	**Status**
Code	Code source complet dans archive .zip	☐
Code	Pas de node_modules dans l'archive	☐
Code	Pas de dossier .git dans l'archive	☐
Code	Fichiers .env exclus	☐
Code	Code commenté (JSDoc sur fonctions)	☐
Code	Serveur HTTP natif Node.js (pas Express)	☐
Documentation	README.md complet avec instructions	☐
Documentation	Tous les JSON schemas créés et versionnés	☐
Documentation	Guide de déploiement serveur	☐
Documentation	Documentation API WebSocket	☐
Documentation	Schémas MongoDB documentés	☐
Documentation	CHANGELOG.md maintenu	☐
Documentation	Diagrammes d'architecture inclus	☐
Documentation	Documentation de la logique de jeu	☐
Rapport	Architecture présentée et expliquée	☐
Rapport	Choix techniques justifiés (pourquoi pas Express)	☐
Rapport	Découpe des tâches entre étudiants	☐
Rapport	JSON schemas référencés et expliqués	☐
Rapport	Taille de l'archive < limite Moodle	☐
Fonctionnel	Serveur HTTP natif démarre sans erreur	☐
Fonctionnel	Client Cordova compile	☐
Fonctionnel	Enregistrement avec pseudo fonctionne	☐
Fonctionnel	Partie multijoueur (2-4 joueurs) OK	☐
Fonctionnel	High scores affichés correctement	☐
Fonctionnel	Jeu jouable sur smartphone	☐
Fonctionnel	Jeu jouable sur navigateur web	☐
Fonctionnel	Validation JSON schemas côté serveur	☐

Tests	Tests manuels effectués	☐
Tests	Scénarios multijoueurs testés	☐
Tests	Gestion déconnexions testée	☐
Tests	Tests de charge (plusieurs parties)	☐
Soutenance	Présentation préparée	☐
Soutenance	Démonstration prête	☐
Soutenance	Questions anticipées	☐