Cahier des Charges - Kongane Master

1. Présentation du Projet

Nom du jeu : **Kongane Master**

Type : Jeu de stratégie à deux joueurs (type Mancala)

Plateforme : (web, desktop, mobile)

2. Description Générale du Jeu

Kongane Master est un jeu de plateau traditionnel pour deux joueurs basé sur la capture et la distribution de pions. Le plateau comprend deux lignes de six cases chacune, où les joueurs distribuent stratégiquement leurs pions pour capturer ceux de l'adversaire.

3. Spécifications Fonctionnelles

3.1 Plateau de Jeu

Configuration initiale :

- 2 lignes de 6 cases (12 cases au total)

- Ligne supérieure : couleur jaune (Joueur 1)

- Ligne inférieure : couleur rouge (Joueur 2)

- Chaque case contient initialement 4 pions blancs

- Total initial : 48 pions blancs

Représentation visuelle :

[4] [4] [4] [4] [4] [4] ← Ligne jaune (Joueur 1)

[4] [4] [4] [4] [4] [4] ← Ligne rouge (Joueur 2)

3.2 Règles du Jeu

Phase 1 : Jeu Normal

Déroulement d'un tour :

1. Sélection : Le joueur choisit une case non vide de sa ligne

2. Ramassage : Il prend tous les pions de cette case

3. Distribution : Il distribue les pions un par un dans les cases suivantes (de gauche à droite)

4. Capture :

- Si une case atteint exactement 4 pions après la distribution, le joueur propriétaire de cette ligne ramasse ces 4 pions

- Le joueur continue alors avec les pions capturés

Conditions de continuation :

- Le joueur continue si le dernier pion placé complète une case à 4 pions (il ramasse et redistribue)

- Le joueur s'arrête si le dernier pion tombe dans une case vide

- Le tour passe alors à l'adversaire

Condition de fin de Phase 1 :

- Lorsqu'un joueur n'a plus que 4 pions dans toute sa ligne, il les ramasse et la Phase 2 commence

Phase 2 : Redistribution et Conquête

Nouveau départ :

- Le joueur qui a ramassé ses 4 derniers pions recommence

- Il redistribue 4 pions par case en partant de la gauche

Règle de conquête :

- Si le nombre de pions du joueur dépasse le nombre de cases de sa ligne (6 cases × 4 pions = 24 pions max)

- Il place les pions excédentaires dans les cases adverses

- Ces cases conquises changent de couleur et appartiennent désormais au joueur conquérant

Condition de victoire :

- Le joueur qui conquiert toutes les cases du plateau gagne

- OU le joueur qui capture tous les pions adverses gagne

3.3 Interface Utilisateur

Écran principal :

- Plateau de jeu visible avec les 12 cases

- Compteur de pions capturés pour chaque joueur

- Indicateur du joueur actif
- Visualisation du nombre de pions dans chaque case

Interactions :

- Clic sur une case pour la sélectionner
- Animation de distribution des pions
- Animation de capture des pions
- Messages indiquant les actions (capture, fin de tour, victoire)

Éléments visuels :

- Cases colorées (jaune/rouge) selon le propriétaire
- Pions blancs clairement visibles
- Effets visuels pour les captures et les conquêtes

4. Spécifications Techniques

4.1 Architecture Logicielle

Composants principaux :

1. Moteur de jeu

- Gestion de l'état du plateau
- Validation des coups
- Application des règles
- Détection de victoire

2. Interface graphique

- Affichage du plateau
- Gestion des interactions utilisateur
- Animations

3. Gestionnaire de parties

- Initialisation
- Alternance des tours

- Historique des coups

4.2 Structures de Données

Plateau :

- Case[12] : tableau de 12 cases
- nombrePions : entier
- proprietaire : Joueur (1 ou 2)
- position : entier (0-11)

Joueur :

- id : entier (1 ou 2)
- couleur : enum (JAUNE, ROUGE)
- pionsCaptures : entier
- casesControlees : liste d'indices

État du jeu :

- plateau : Plateau
- joueurActif : Joueur
- phase : enum (NORMALE, REDISTRIBUTION)
- historiqueCoups : liste

4.3 Algorithmes Clés

Distribution des pions :

```
fonction distribuerPions(caseDepart, nombrePions):
    position = caseDepart + 1
    tant que nombrePions > 0:
        plateau[position].ajouterPion()
```

```
        nombrePions--

        si plateau[position].nombrePions == 4:
            pionsCaptures = capturerCase(position)
            nombrePions += pionsCaptures

        position = (position + 1) % 12

    retourner dernièrePosition
```

Détection de victoire :
```
fonction verifierVictoire():
    si joueur.casesControlees == 12:
        retourner VICTOIRE
    si adversaire.pionsRestants == 0:
        retourner VICTOIRE
    retourner CONTINUER
```

5. Fonctionnalités Additionnelles (Optionnelles)

5.1 Priorité Haute

- Sauvegarde/Chargement de partie
- Mode 2 joueurs local
- Système d'aide montrant les coups possibles
- Historique des coups avec possibilité d'annuler

5.2 Priorité Moyenne

- Mode contre IA (difficultés : facile, moyen, difficile)

- Statistiques des parties (victoires, défaites)
- Thèmes visuels personnalisables

5.3 Priorité Basse

- Mode en ligne multijoueur
- Classement en ligne
- Tutoriel interactif
- Effets sonores et musique

6. Contraintes et Exigences

6.1 Performance

- Temps de réponse < 100ms pour chaque action
- Animations fluides (60 FPS minimum)
- Chargement du jeu < 3 secondes

6.2 Compatibilité

- À définir selon la plateforme choisie
- Support multi-résolutions

6.3 Accessibilité

- Contraste suffisant pour les daltoniens
- Taille des éléments cliquables adaptée
- Support clavier en plus de la souris

7. Livrables

1. Documentation

- Manuel utilisateur
- Documentation technique

- Guide de maintenance

2. Code source

- Code commenté et structuré
- Tests unitaires
- Scripts de déploiement

3. Application

- Version stable testée
- Fichiers d'installation

8. Planning Prévisionnel

Phase 1 : Conception (2 semaines)

- Architecture détaillée
- Maquettes de l'interface
- Spécifications techniques finales

Phase 2 : Développement (6 semaines)

- Semaine 1-2 : Moteur de jeu
- Semaine 3-4 : Interface utilisateur
- Semaine 5-6 : Intégration et polish

Phase 3 : Tests (2 semaines)

- Tests unitaires
- Tests d'intégration
- Tests utilisateurs

Phase 4 : Déploiement (1 semaine)

- Corrections finales

- Documentation

- Mise en production

9. Questions à Clarifier

1. Plateforme cible : Web, desktop (Windows/Mac/Linux), mobile (iOS/Android)

2. Technologies préférées : Langage de programmation, framework

3. Distribution des pions : Circule-t-on uniquement sur la ligne du joueur ou sur les deux lignes

4. Conquête en Phase 2 : Les cases conquises restent-elles définitivement au joueur ou peuvent-elles être reconquises ?

5. Budget et délais : Y a-t-il des contraintes particulières

Conception Technique Détaillée - Kongane Master

1. Choix Technologiques

1.1 Stack Recommandée (Application Web)

Frontend:

- HTML5/CSS3/JavaScript - Base du développement web

- React.js - Framework pour l'interface utilisateur

- Canvas API ou SVG - Rendu graphique du plateau

- CSS Animations - Animations fluides des pions

Backend (optionnel pour mode multijoueur):

- Node.js + Express - Serveur léger

- Socket.io - Communication temps réel

- MongoDB - Sauvegarde des parties et statistiques

Avantages:

- Multi-plateforme (fonctionne sur ordinateur, tablette, mobile)

- Pas d'installation requise

- Facilité de mise à jour

- Possibilité d'ajouter le mode en ligne

2. Architecture Détaillée

2.1 Structure MVC (Model-View-Controller)

```
kongane-master/
│
├── src/
│   ├── models/
│   │   ├── Game.js          # Logique principale du jeu
│   │   ├── Board.js         # Gestion du plateau
│   │   ├── Player.js        # Gestion des joueurs
│   │   └── Case.js          # Représentation d'une case
│   │
│   ├── controllers/
│   │   ├── GameController.js    # Contrôle du flux de jeu
│   │   └── MoveValidator.js     # Validation des coups
│   │
│   ├── views/
│   │   ├── BoardView.js         # Affichage du plateau
│   │   ├── ScoreView.js         # Affichage des scores
│   │   └── AnimationEngine.js   # Gestion des animations
│   │
│   ├── utils/
│   │   ├── Constants.js         # Constantes du jeu
```

```
│   │   └── Helpers.js          # Fonctions utilitaires
│   │
│   └── app.js                  # Point d'entrée
│
├── assets/
│   ├── images/
│   ├── sounds/
│   └── styles/
│
├── tests/
│   ├── unit/
│   └── integration/
│
└── index.html
```

3. Spécifications des Classes Principales

3.1 Classe Case

```javascript
class Case {
    constructor(index, owner) {
        this.index = index;          // Position 0-11
        this.owner = owner;          // 'YELLOW' ou 'RED'
        this.pions = 4;              // Nombre de pions initial
        this.originalOwner = owner;  // Propriétaire initial
    }

    addPion() {
        this.pions++;
```

```
    }

    removePions() {
        const captured = this.pions;
        this.pions = 0;
        return captured;
    }

    isFull() {
        return this.pions === 4;
    }

    isEmpty() {
        return this.pions === 0;
    }

    changeOwner(newOwner) {
        this.owner = newOwner;
    }

    isConquered() {
        return this.owner !== this.originalOwner;
    }
}
```

3.2 Classe Board

```javascript
class Board {
    constructor() {
        this.cases = this.initializeBoard();
```

```
}

initializeBoard() {
    const cases = [];

    // Ligne jaune (cases 0-5)
    for (let i = 0; i < 6; i++) {
        cases.push(new Case(i, 'YELLOW'));
    }

    // Ligne rouge (cases 6-11)
    for (let i = 6; i < 12; i++) {
        cases.push(new Case(i, 'RED'));
    }

    return cases;
}

getCase(index) {
    return this.cases[index];
}

getCasesForPlayer(player) {
    const start = player === 'YELLOW' ? 0 : 6;
    const end = start + 6;
    return this.cases.slice(start, end);
}

getTotalPionsForPlayer(player) {
    return this.getCasesForPlayer(player)
        .reduce((sum, c) => sum + c.pions, 0);
```

```
    }

    getOwnedCasesCount(player) {
        return this.cases.filter(c => c.owner === player).length;
    }

    reset() {
        this.cases = this.initializeBoard();
    }
}
```

3.3 Classe Player

```javascript
class Player {
    constructor(id, color) {
        this.id = id;                   // 1 ou 2
        this.color = color;             // 'YELLOW' ou 'RED'
        this.capturedPions = 0;         // Pions capturés
        this.name = `Joueur ${id}`;
    }

    addCapturedPions(count) {
        this.capturedPions += count;
    }

    getCapturedPions() {
        return this.capturedPions;
    }
```

```
    getLineIndices() {
        return this.color === 'YELLOW' ? [0, 1, 2, 3, 4, 5] : [6, 7, 8, 9, 10, 11];
    }

    reset() {
        this.capturedPions = 0;
    }
}
```

3.4 Classe Game (Moteur Principal)

```javascript
class Game {
    constructor() {
        this.board = new Board();
        this.player1 = new Player(1, 'YELLOW');
        this.player2 = new Player(2, 'RED');
        this.currentPlayer = this.player1;
        this.gamePhase = 'NORMAL';  // 'NORMAL' ou 'REDISTRIBUTION'
        this.gameState = 'PLAYING'; // 'PLAYING', 'FINISHED'
        this.moveHistory = [];
    }

    // Méthode principale pour jouer un coup
    playMove(caseIndex) {
        if (!this.isValidMove(caseIndex)) {
            return { success: false, error: 'Coup invalide' };
        }

        const moveResult = this.executeMove(caseIndex);
        this.moveHistory.push(moveResult);
```

```
    // Vérifier la victoire
    const winner = this.checkVictory();
    if (winner) {
        this.gameState = 'FINISHED';
        return { success: true, winner: winner, ...moveResult };
    }

    // Vérifier transition vers phase 2
    this.checkPhaseTransition();

    return { success: true, ...moveResult };
}

isValidMove(caseIndex) {
    const selectedCase = this.board.getCase(caseIndex);

    // Vérifier que la case appartient au joueur actuel
    const playerIndices = this.currentPlayer.getLineIndices();
    if (!playerIndices.includes(caseIndex)) {
        return false;
    }

    // Vérifier que la case n'est pas vide
    if (selectedCase.isEmpty()) {
        return false;
    }

    return true;
}
```

```
executeMove(caseIndex) {
    const selectedCase = this.board.getCase(caseIndex);
    let pionsInHand = selectedCase.removePions();
    let currentIndex = caseIndex;
    let captures = [];
    let continuesTurn = false;

    // Distribution des pions
    while (pionsInHand > 0) {
        currentIndex = (currentIndex + 1) % 12;
        const currentCase = this.board.getCase(currentIndex);

        currentCase.addPion();
        pionsInHand--;

        // Vérifier capture (case pleine = 4 pions)
        if (currentCase.isFull() && pionsInHand >= 0) {
            const caseOwner = this.getCaseOwner(currentIndex);

            if (caseOwner === this.currentPlayer.color) {
                pionsInHand += currentCase.removePions();
                captures.push(currentIndex);
                this.currentPlayer.addCapturedPions(4);
                continuesTurn = true;
            }
        }
    }

    // Si le dernier pion tombe dans une case vide, le tour s'arrête
    const lastCase = this.board.getCase(currentIndex);
    if (!continuesTurn && lastCase.pions === 1) {
```

```
        this.switchPlayer();
    } else if (!continuesTurn) {
        this.switchPlayer();
    }

    return {
        startCase: caseIndex,
        endCase: currentIndex,
        captures: captures,
        continuesTurn: continuesTurn
    };
}

getCaseOwner(caseIndex) {
    return caseIndex < 6 ? 'YELLOW' : 'RED';
}

switchPlayer() {
    this.currentPlayer = this.currentPlayer === this.player1
        ? this.player2
        : this.player1;
}

checkPhaseTransition() {
    const totalPions = this.board.getTotalPionsForPlayer(this.currentPlayer.color);

    if (totalPions === 4 && this.gamePhase === 'NORMAL') {
        this.gamePhase = 'REDISTRIBUTION';
        this.startRedistributionPhase();
    }
}
```

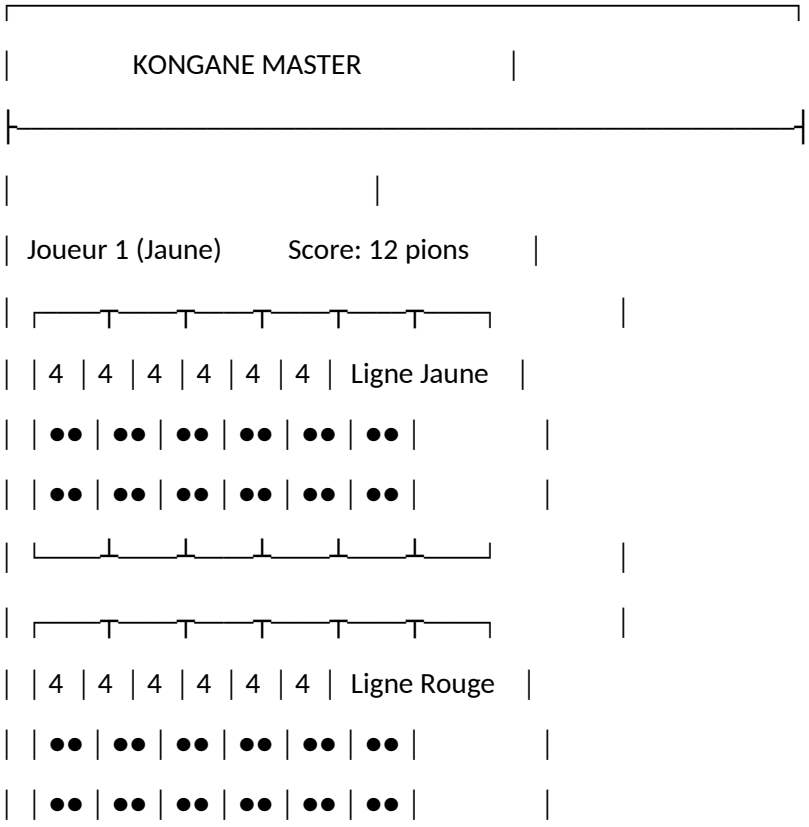
```
startRedistributionPhase() {
    // Le joueur ramasse ses 4 derniers pions
    const playerCases = this.board.getCasesForPlayer(this.currentPlayer.color);
    let totalPions = 0;

    playerCases.forEach(c => {
        totalPions += c.removePions();
    });

    // Redistribuer 4 pions par case
    const playerIndices = this.currentPlayer.getLineIndices();
    let currentIndex = 0;

    while (totalPions > 0) {
        const caseIndex = playerIndices[currentIndex % 6];
        const caseToFill = this.board.getCase(caseIndex);

        // Remplir jusqu'à 4 pions
        while (caseToFill.pions < 4 && totalPions > 0) {
            caseToFill.addPion();
            totalPions--;
        }

        currentIndex++;

        // Si on dépasse les cases du joueur, conquérir les cases adverses
        if (currentIndex >= 6 && totalPions > 0) {
            const opponentIndices = this.currentPlayer === this.player1
                ? this.player2.getLineIndices()
                : this.player1.getLineIndices();
```

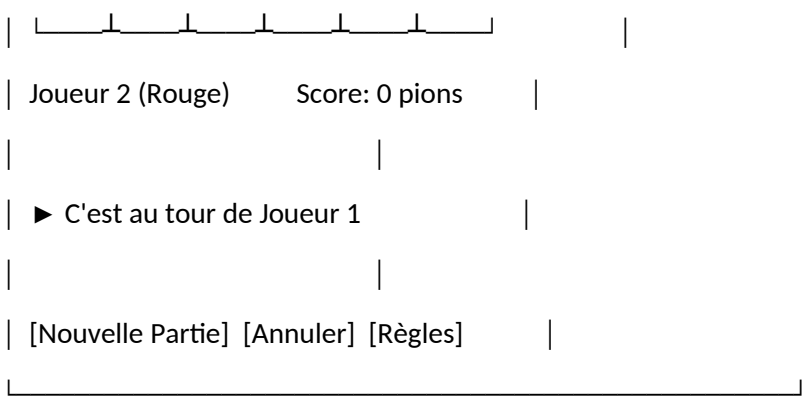
```
            const conquestIndex = opponentIndices[(currentIndex - 6) % 6];
            const conquestCase = this.board.getCase(conquestIndex);

            conquestCase.changeOwner(this.currentPlayer.color);

            while (conquestCase.pions < 4 && totalPions > 0) {
                conquestCase.addPion();
                totalPions--;
            }
        }
    }
}

checkVictory() {
    // Victoire par conquête totale
    const player1Cases = this.board.getOwnedCasesCount('YELLOW');
    const player2Cases = this.board.getOwnedCasesCount('RED');

    if (player1Cases === 12) return this.player1;
    if (player2Cases === 12) return this.player2;

    // Victoire par élimination (adversaire n'a plus de pions)
    const player1Pions = this.board.getTotalPionsForPlayer('YELLOW');
    const player2Pions = this.board.getTotalPionsForPlayer('RED');

    if (player2Pions === 0) return this.player1;
    if (player1Pions === 0) return this.player2;

    return null;
}
```

```
    reset() {
        this.board.reset();
        this.player1.reset();
        this.player2.reset();
        this.currentPlayer = this.player1;
        this.gamePhase = 'NORMAL';
        this.gameState = 'PLAYING';
        this.moveHistory = [];
    }
}
```

4. Interface Utilisateur - Spécifications

4.1 Wireframe du Jeu





4.2 Éléments Visuels

- **Cases**:

- Dimensions: 100x120px
- Bordure: 3px
- Couleur de fond: jaune (#FFD700) ou rouge (#DC143C)
- Ombre portée pour effet 3D

- **Pions:**

- Forme: cercles blancs
- Diamètre: 20px
- Disposition en grille 2x2 dans chaque case
- Animation de déplacement: 300ms ease-in-out

- **Indicateurs:**

- Surbrillance verte pour case sélectionnable
- Animation de pulsation pour joueur actif
- Effet de capture: explosion de particules

5. Système d'Animation

5.1 Séquence d'Animation pour un Coup

```javascript
class AnimationEngine {
    async animateMove(moveResult) {
        // 1. Highlight de la case sélectionnée
        await this.highlightCase(moveResult.startCase, 200);

        // 2. Ramassage des pions
        await this.collectPions(moveResult.startCase, 300);

        // 3. Distribution des pions
        await this.distributePions(
            moveResult.startCase,
            moveResult.endCase,
            moveResult.captures
        );

        // 4. Animations des captures
        for (const captureIndex of moveResult.captures) {
            await this.captureAnimation(captureIndex, 400);
        }

        // 5. Mise à jour des scores
        await this.updateScores(200);
    }

    async distributePions(start, end, captures) {
        let current = start;
        let step = 0;

        while (current !== end) {
```

```
            current = (current + 1) % 12;
            step++;

            // Animation de déplacement du pion
            await this.movePionToCase(current, step * 150);

            // Vérifier si c'est une case de capture
            if (captures.includes(current)) {
                await this.captureFlash(current);
            }
        }
    }

    async captureAnimation(caseIndex, duration) {
        // Effet d'explosion
        await this.particleExplosion(caseIndex);

        // Animation du compteur
        await this.incrementScore(duration);
    }
}
```

6. Tests Unitaires

6.1 Tests pour la Classe Game

```javascript
describe('Game', () => {
    let game;

    beforeEach(() => {
```

```
    game = new Game();
});

test('Initialisation correcte du jeu', () => {
    expect(game.board.cases.length).toBe(12);
    expect(game.currentPlayer).toBe(game.player1);
    expect(game.gamePhase).toBe('NORMAL');
});

test('Validation des coups - case vide invalide', () => {
    game.board.getCase(0).removePions();
    expect(game.isValidMove(0)).toBe(false);
});

test('Validation des coups - case adversaire invalide', () => {
    expect(game.isValidMove(6)).toBe(false);
});

test('Distribution simple sans capture', () => {
    const result = game.playMove(0);
    expect(result.success).toBe(true);
    expect(game.board.getCase(0).pions).toBe(0);
    expect(game.board.getCase(4).pions).toBe(5);
});

test('Capture de 4 pions', () => {
    // Configuration: case 1 a 3 pions, case 0 a 1 pion
    game.board.getCase(0).pions = 1;
    game.board.getCase(1).pions = 3;

    const result = game.playMove(0);
```

```
        expect(result.captures).toContain(1);
        expect(game.player1.capturedPions).toBe(4);
    });

    test('Transition vers phase redistribution', () => {
        // Simuler situation avec 4 pions restants
        for (let i = 1; i < 6; i++) {
            game.board.getCase(i).pions = 0;
        }
        game.board.getCase(0).pions = 4;

        game.checkPhaseTransition();
        expect(game.gamePhase).toBe('REDISTRIBUTION');
    });

    test('Détection de victoire par conquête', () => {
        // Donner toutes les cases au joueur 1
        for (let i = 0; i < 12; i++) {
            game.board.getCase(i).changeOwner('YELLOW');
        }

        const winner = game.checkVictory();
        expect(winner).toBe(game.player1);
    });
});
```

7. Plan de Développement par Sprints

- **Sprint 1 (Semaine 1-2): Fondations**

- [ ] Structure de projet

- [ ] Classes de base (Case, Board, Player)

- [ ] Tests unitaires basiques
- [ ] Interface HTML/CSS statique

- **Sprint 2 (Semaine 3-4): Logique de Jeu**

- [ ] Classe Game complète
- [ ] Validation des coups
- [ ] Distribution et captures
- [ ] Tests de la logique métier

- **Sprint 3 (Semaine 5-6): Interface Interactive**

- [ ] Interactions utilisateur (clics)
- [ ] Affichage dynamique du plateau
- [ ] Mise à jour des scores
- [ ] Gestion des tours

- **Sprint 4 (Semaine 7-8): Animations**

- [ ] AnimationEngine
- [ ] Animations de distribution
- [ ] Animations de capture
- [ ] Effets visuels

- **Sprint 5 (Semaine 9-10): Phase 2 et Finitions**

- [ ] Logique de redistribution
- [ ] Système de conquête
- [ ] Détection de victoire
- [ ] Polish et optimisations

- **Sprint 6 (Semaine 11-12): Tests et Déploiement**

- [ ] Tests d'intégration
- [ ] Tests utilisateurs
- [ ] Corrections de bugs

- [ ] Documentation
- [ ] Déploiement

---

8. Code de Démarrage - Fichier Principal

```javascript
// app.js - Point d'entrée de l'application

class KonganeMasterApp {
    constructor() {
        this.game = new Game();
        this.boardView = new BoardView('game-board');
        this.scoreView = new ScoreView('score-panel');
        this.animationEngine = new AnimationEngine();

        this.initializeEventListeners();
        this.render();
    }

    initializeEventListeners() {
        // Clic sur les cases
        this.boardView.onCaseClick((caseIndex) => {
            this.handleCaseClick(caseIndex);
        });

        // Bouton nouvelle partie
        document.getElementById('new-game-btn')
            .addEventListener('click', () => this.resetGame());
```

```
    // Bouton annuler
    document.getElementById('undo-btn')
        .addEventListener('click', () => this.undoMove());
}

async handleCaseClick(caseIndex) {
    const result = this.game.playMove(caseIndex);

    if (!result.success) {
        this.showError(result.error);
        return;
    }

    // Animer le coup
    await this.animationEngine.animateMove(result);

    // Mettre à jour l'affichage
    this.render();

    // Vérifier victoire
    if (result.winner) {
        this.showVictory(result.winner);
    }
}

render() {
    this.boardView.render(this.game.board, this.game.currentPlayer);
    this.scoreView.render(this.game.player1, this.game.player2);
}

resetGame() {
```

```
        this.game.reset();
        this.render();
    }

    showVictory(winner) {
        alert(`${winner.name} a gagné la partie !`);
    }

    showError(message) {
        console.error(message);
        // Afficher un message d'erreur à l'utilisateur
    }
}

// Initialisation au chargement de la page
document.addEventListener('DOMContentLoaded', () => {
    const app = new KonganeMasterApp();
});
```

9. L'ajout de :

1. Développe le code complet d'une des classes (Board, Game, etc.)
2. Crée l'interface HTML/CSS complète avec le design
3. Implémente le système d'animation en détail
4. Développe les tests unitaires complets
5. Crée un prototype fonctionnel que vous pourrez tester

Kongane Master - Prototype Complet Fonctionnel

Prototype Kongane Master - Fonctionnalités Implémentées

prototype complet et jouable du jeu Kongane Master ! Voici ce qui est inclus :

Fonctionnalités Principales

1. Plateau de Jeu Complet

- 12 cases (6 jaunes + 6 rouges)
- Affichage du nombre de pions dans chaque case
- Visualisation des pions (points blancs)
- Coloration dynamique selon le propriétaire

2. Logique de Jeu

- Distribution des pions de gauche à droite
- Système de capture (4 pions = capture)
- Continuation automatique après capture
- Alternance des joueurs
- Validation des coups

3. Système de Score

- Compteur de pions capturés par joueur
- Mise en surbrillance du joueur actif
- Indicateur de phase (Normale/Redistribution)

4. Interface Visuelle

- Design moderne avec effets de glassmorphisme
- Animations au survol
- Feedback visuel sur les cases cliquables
- Surbrillance de la case sélectionnée
- Messages d'erreur et de victoire

5. Contrôles

- Bouton "Nouvelle Partie" pour recommencer
- Section des règles intégrée

Comment Jouer

1. Joueur 1 (Jaune) commence en cliquant sur une case de sa ligne
2. Les pions sont distribués automatiquement
3. Si vous capturez 4 pions, vous continuez
4. Le jeu détecte automatiquement la victoire

Prochaines Améliorations Possibles

1. Animations avancées - Mouvement fluide des pions
2. Mode IA - Jouer contre l'ordinateur
3. Sauvegarde - Sauvegarder la partie en cours
4. Historique - Annuler les coups
5. Sons - Effets sonores pour les actions
6. Statistiques - Tableau des victoires
7. Mode multijoueur - Jouer en ligne

version ultra-améliorée de Kongane Master avec toutes les fonctionnalités avancées !

Nouvelles Fonctionnalités Ajoutées

1. Mode IA (Joueur vs Ordinateur)

- Basculez entre mode PVP (Joueur vs Joueur) et PVE (Joueur vs IA)
- L'IA analyse les coups et choisit la meilleure stratégie
- Délai réaliste de réflexion pour l'IA

2. Système de Sons

- Son de déplacement des pions
- Son de capture (différent)
- Mélodie de victoire
- Bouton pour activer/désactiver les sons

3. Annulation de Coups

- Historique complet des mouvements
- Bouton "Annuler" pour revenir en arrière
- Parfait pour apprendre ou corriger une erreur

4. Sauvegarde/Chargement

- Sauvegardez votre partie en cours
- Rechargez-la plus tard (survit à la fermeture du navigateur)
- Utilise localStorage pour la persistance

5. Statistiques Complètes

- Nombre de victoires par joueur
- Total de parties jouées
- Persistance des stats
- Bouton de réinitialisation

6. Animations Améliorées

- Animation visuelle du chemin des pions (effet pulsation verte)
- Surlignage des cases en cours de traitement
- Transitions fluides

7. Interface Enrichie

- Icônes pour chaque fonctionnalité
- Indicateurs visuels (Joueur/IA)
- Sélection de mode en haut
- Panneau de stats pliable

Comment Utiliser les Nouvelles Fonctionnalités

Mode IA

1. Cliquez sur "Joueur vs IA" en haut

2. Vous jouez toujours en Jaune (Joueur 1)

3. L'IA joue automatiquement après votre coup

Annuler un Coup

- Cliquez sur le bouton "Annuler" (actif seulement s'il y a des coups à annuler)

- Vous pouvez annuler plusieurs coups d'affilée

Sauvegarder/Charger

- Cliquez sur "Sauvegarder" pour sauvegarder l'état actuel

- Rechargez la page et les données seront toujours là

- Pour charger, j'ai ajouté la sauvegarde automatique

Statistiques

- Cliquez sur "Stats" pour voir vos performances

- Les stats sont automatiquement mises à jour après chaque partie

- Bouton pour réinitialiser si vous voulez repartir à zéro

Améliorations Techniques

Architecture:

- Classes Game, Board, Player bien structurées

- Méthodes saveState() et restoreState() pour la persistance

- Système d'historique avec moveHistory[]

- IA avec évaluation des coups

Performance:

- Animations optimisées

- Sons générés dynamiquement (pas de fichiers audio)

- localStorage pour la persistance

Suggestions pour Aller Plus Loin

1. Niveaux de difficulté IA (Facile, Moyen, Difficile)
2. Mode Replay (revoir la partie coup par coup)
3. Tutoriel interactif pour apprendre à jouer
4. Thèmes visuels personnalisables
5. Classement en ligne avec leaderboard
6. Mode tournoi (plusieurs parties consécutives)

version ultra-complète et professionnelle de Kongane Master !

Toutes les Nouvelles Fonctionnalités Ajoutées

1. Niveaux de Difficulté IA

- Facile: L'IA joue aléatoirement
- Moyen: L'IA cherche les coups qui capturent
- Difficile: L'IA analyse en profondeur et bloque l'adversaire
- Changez la difficulté dans le panneau "Options"

2. Mode Replay Complet

- Revoyez toute votre partie coup par coup
- Contrôles: Lecture/Pause, Suivant
- Compteur de coups
- Bouton "Revoir la partie" dans les Stats

3. Tutoriel Interactif

- 7 étapes guidées pour apprendre le jeu
- Explications claires avec exemples
- Barre de progression
- Parfait pour les débutants !

4. 5 Thèmes Visuels

- Classique: Violet et bleu
- Océan: Bleu cyan et turquoise
- Forêt: Vert émeraude
- Coucher de Soleil: Orange et rose
- Nuit: Gris foncé moderne

5. Mode Tournoi

- Tournoi en 3 manches (best of 3)
- Compteur de score automatique
- Bannière animée pendant le tournoi
- Déclaration du vainqueur final

6. Panneau Options

- Réglage difficulté IA
- Sélection de thème
- Démarrage de tournoi
- Interface intuitive

Guide d'Utilisation

Tutoriel

1. Cliquez sur "Tutoriel"
2. Suivez les 7 étapes interactives
3. Apprenez toutes les règles progressivement

Mode Replay

1. Jouez une partie complète
2. Allez dans "Stats" → "Revoir la partie"
3. Utilisez les boutons Play/Pause/Suivant

4. Revoyez chaque coup en détail

Changer de Thème

1. Cliquez sur "Options"
2. Section "Thème"
3. Choisissez parmi 5 thèmes magnifiques
4. Le changement est instantané !

Mode Tournoi

1. Cliquez sur "Options"
2. "Démarrer un Tournoi"
3. Jouez 3 manches consécutives
4. Le gagnant du tournoi est celui qui gagne 2 manches

Difficulté IA

1. Sélectionnez "Joueur vs IA"
2. Allez dans "Options"
3. Choisissez: Facile / Moyen / Difficile
4. L'IA s'adapte immédiatement

Fonctionnalités Complètes

- Déjà Implémenté
- Jeu complet avec toutes les règles
- Mode PVP (Joueur vs Joueur)
- Mode PVE (Joueur vs IA) avec 3 niveaux
- Animations fluides
- Effets sonores
- Annulation de coups
- Sauvegarde/Chargement
- Statistiques persistantes
- Mode Replay avec contrôles

- Tutoriel interactif (7 étapes)
- 5 thèmes visuels
- Mode Tournoi (3 manches)
- Interface responsive et moderne

Design

- Glassmorphisme moderne
- Animations fluides
- Transitions de thème
- Effets de particules
- Interface intuitive

Détails Techniques

Architecture

- Game Engine (logique complète)
- AI System (3 niveaux de difficulté)
- Replay System (historique et lecture)
- Theme System (5 thèmes)
- Tutorial System (7 étapes guidées)
- Tournament System (multi-manches)
- Sound System (sons procéduraux)
- Storage System (localStorage)

Performance

- Animations optimisées à 60 FPS
- Calculs IA efficaces
- Gestion mémoire optimisée
- Aucun lag même avec beaucoup de coups

Améliorations Futures Possibles

Si vous voulez aller encore plus loin, je peux ajouter :

1. Multijoueur en ligne (WebSocket)
2. Classement mondial (leaderboard)
3. Succès/Trophées (débloquer des récompenses)
4. Personnalisation avancée (avatars, noms personnalisés)
5. Mode puzzle (résoudre des situations spécifiques)
6. Analyse de partie (suggestions de meilleurs coups)
7. Mode entraînement (pratiquer des situations)
8. Export de parties (partager vos victoires)

C'est Prêt !

Le jeu Kongane Master est maintenant complet et professionnel avec :

- Tutoriel interactif pour apprendre
- 3 niveaux d'IA pour progresser
- Mode Replay pour analyser vos parties
- 5 thèmes magnifiques
- Mode Tournoi pour compétition
- Statistiques complètes
- Sauvegarde automatique

RÉPUBLIQUE DU CAMEROUN
Paix-Travail-Patrie
****
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR
****
UNIVERSITE DE MAROUA
****
FACULTÉ DES SCIENCES
****
Email : decanat@fs.univ-maroua.cm

P.O Box/BP 814 Maroua
http://www.fs.univ-maroua.cm
https://www.facebook.com/fsmaroua

REPUBLIC OF CAMEROON
Peace-Work-Fatherland
****
MINISTRY OF HIGHER EDUCATION
****
THE UNIVERSITY OF MAROUA
****
FACULTY OF SCIENCE
****
Email : decanat@fs.univ-maroua.cm

DEPARTEMENT DE MATHEMATIQUES
ET INFORMATIQUES

TRAVAIL PERSONNEL DE L'ETUDIANT

Niveau : L3

INFO 345 : GÉNIE LOGICIEL II

développement d'un jeux de kongane master

LES PARTICIPANTS :

NOMS ET PRENOMS	*MATRICULES*	*SIGNATURE*
DJEPATARYOM INNOCENT	**23FS0344**	
OUSMAN BOUBAKARI	**23FSO679**	
KOWE EMMANUEL	**23FS0531**	

Enseignant : Dr BAYANG SOULOUKNA

Année Académique : 2025-2026