Rapport de Conception : Jeu9fevCanva

1. Vue d'ensemble
   Ce projet constitue un moteur de jeu personnalisable conçu en JavaScript. Il repose sur une architecture modulaire, divisée en deux grands volets : moteur graphique/interaction (engine) et logique de jeu (game).
2. Classes principales

2.1 Moteur de Jeu (engine)

- Engine.js : Gère la boucle principale de mise à jour et de rendu.
- Scene.js : Classe de base pour les scènes (ex : GameScene).
- ObjectGraphique.js : Gère l'affichage des entités graphiques.
- GameObject.js : Sert de base pour les entités interactives (joueur, ennemis...).
- InputManager.js : Centralise les entrées clavier et souris.
- Collision.js : Fournit des outils de détection de collision.

2.2 Logique de Jeu (game)
Entities :

- Player.js : Définit les propriétés et comportements du joueur.
- Enemy.js : Gère les ennemis et leur comportement.
- Projectile.js : Représente les tirs du joueur.
- Item.js : Objets ramassables (bonus).
- ObjetSouris.js : Entité manipulable à la souris.

World :

- Dungeon.js : Génère dynamiquement les étages du donjon.
- Room.js : Gère les salles, obstacles, ennemis et progression.
- Obstacle.js : Représente les obstacles fixes.
- Sortie.js : Gère l'escalier vers l'étage suivant.
- GameScene.js : Scène principale orchestrant les entités du donjon.

3. Interactions entre classes

- Engine déclenche les mises à jour via Scene.
- GameScene contient le Dungeon, les Room, le Player et les Enemy.

- Room encapsule les éléments d'un étage (obstacles, ennemis, sortie).
- Projectile interagit avec Enemy via le module Collision.

4. **Points forts**

- Modularité : Chaque entité est indépendante.
- Réutilisabilité : Composants génériques facilement étendables.
- Organisation claire : Rôles bien définis entre moteur et logique.

5. **Améliorations futures**

- Sous-classes pour ennemis spécifiques ou boss.
- Étages thématiques ou dotés de mécaniques originales.
- Ajout de pièges dynamiques ou effets spéciaux complexes.

Rapport de Conception : CandyCrush

1. Vue d'ensemble
   Jeu de type Match-3 développé en JavaScript. L'interface est une grille dynamique de cookies avec interactions via clic et glisser-déposer. L'architecture suit un modèle modulaire (Grid, Cookie, ScoreManager, GameManager).
2. Classes principales

2.1 Classe Grid
Rôle : Gère la grille, les déplacements, les matchs et la génération de cookies.
Méthodes clés :

- showCookies(), detecterMatch3Lignes(), hasMatches()
- remplacerCookiesMarques() : Remplace les cookies supprimés.
- updateCookieTypes() : Adapte la complexité selon le niveau.

2.2 Classe Cookie
Rôle : Représente une cellule de la grille.
Attributs : position (ligne, colonne), type, image HTML.
Méthodes :

- marquer(), selectionnee(), deselectionnee()
- static swapCookies(c1, c2) : Permet d'échanger deux cookies.

2.3 Classe ScoreManager
Rôle : Gère le score, la progression et le niveau.
Méthodes :

- addScore(points), updateDisplay(), calculateThreshold()

2.4 Classe GameManager
Rôle : Supervise l'exécution globale du jeu.
Méthodes :

- init(), setupEventListeners(), faireDisparaitreCookies()

3. Interactions entre classes

- Grid contient les Cookie et utilise ScoreManager.
- GameManager orchestre la logique globale.
- Les événements personnalisés synchronisent les actions.

4. Points forts

- Architecture claire : Modules séparés et maintenables.
- Évolutivité : Ajout facile de nouveaux types de cookies ou niveaux.
- Responsabilités bien définies : Affichage, logique et score indépendants.

5. Améliorations futures

- Grilles garantissant au moins un match.
- Niveaux de difficulté progressifs.
- Animations et transitions plus fluides.

Rapport de Conception : Jeu PvP au Tour par Tour

1. Vue d'ensemble
   Ce jeu oppose deux joueurs sur une grille en tour par tour. Ils peuvent se déplacer, lancer des sorts et interagir avec des obstacles. L'architecture repose sur une séparation stricte des rôles par classe.
2. Classes principales

2.1 Classe Player
Fichier : Player.js
Rôle : Gère les caractéristiques et actions d'un joueur.
Attributs :

- pv, pa, pm avec leur version max
- position, name
  Méthodes :
- takeDamage(), heal(), resetTurn(), moveTo()

2.2 Classe Board
Fichier : Board.js
Rôle : Gère l'affichage, les obstacles et les cellules.
Méthodes :

- init(), renderPlayers(), renderObstacles()
- highlightMoveRange(), highlightSpellRange()
- clearHighlights(), addObstacle(), isObstacle()

2.3 Classe Spell
Fichier : Spell.js
Rôle : Représente un sort (attaque, soin, déplacement).
Attributs : name, cost, range, damage, targetType
Méthode principale :

- cast() : Applique les effets à une cible
  Sous-classe :
- JumpSpell : Permet de sauter d'une case à une autre.

2.4 Classe Obstacle
Fichier : Obstacle.js
Rôle : Représente une cellule bloquante.
Attributs : x, y, type, destructible, hp

3. Interactions entre classes

- Player utilise Board pour se déplacer.
- Spell.cast() interagit avec les Player et la grille.
- Obstacle est intégré au Board et peut interférer avec les mouvements.

4. Points forts

- Modularité : Chaque élément est une classe indépendante.
- Réutilisabilité : Sorts et plateau extensibles.
- Architecture claire : Contrôle fluide du gameplay et des règles.

5. Améliorations futures

- Création dynamique de sorts avec effets personnalisés.
- IA (classe AIPlayer) pour mode solo.
- Tests unitaires pour fiabiliser les mécaniques.

6. **Conclusion**
   Grâce à une structure claire et évolutive, ce jeu PvP offre un socle solide pour enrichissement futur : compétences avancées, IA, multi-joueurs ou mode campagne.