Letoublon Thomas

Roustan Liam

Moyaerts Mathieu

**Rapport BDD**

**a) Analyse et justification des scénarios**

**i) Identification des cas de test**

**Démineur :**

Nous avons identifié plusieurs catégories de cas de test pour couvrir le comportement du Démineur :

* **Cas nominaux** : interactions classiques du joueur avec le jeu. Exemples :
  + Cliquer sur une case vide.
  + Poser ou retirer un drapeau.
  + Révéler toutes les cases non piégées.
* **Cas limites** : comportement en bordure de la grille ou avec des données particulières.
  + Cliquer sur une case au coin ou au bord.
  + Premier clic au centre ou à la périphérie.
* **Cas d'erreur / protection** :
  + Cliquer sur une case avec un drapeau.
  + Poser un drapeau sur une case déjà révélée.
  + Révélation de toutes les bombes en cas de défaite.

Notre stratégie vise une **couverture fonctionnelle totale**, en validant les règles de base, les limites, et la résilience aux mauvaises interactions.

**ii) Priorisation des scénarios**

**Démineur :**

L'ordre de développement a suivi une logique de dépendance fonctionnelle :

1. **Scénarios critiques / fondamentaux** :
   * Initialisation de la grille.
   * Premier clic sans bombe.
   * Placement des bombes.
   * Révélation de case.
2. **Scénarios interactifs** :
   * Poser / retirer un drapeau.
   * Empêcher une action sur case marquée.
3. **Scénarios de fin de partie** :
   * Cliquer sur une bombe.
   * Gagner la partie.

Cette priorité garantit une progression stable et testable du moteur du jeu.

**b) Architecture et représentation des données**

**i) Lisibilité des données de test**

Nous avons utilisé différentes structures dans les fichiers. feature :

* **Scenario Outline + Examples** : pour tester les cas paramétrés (dimensions de la grille, valeurs de numéro).
* **Background** : pour préparer un contexte commun (état de la partie en cours, grille initialisée).
* **Scénarios simples** : quand un cas est spécifique et autonome.

Ce choix améliore la lisibilité et réduit la duplication des contextes.

**ii) Extensibilité**

Notre architecture BDD et code est conçue pour être facilement étendue :

* Les StepDefinitions sont modulaires et réutilisables.
* Les Case, Grille, et GameManager sont isolés dans une bibliothèque (Game) utilisable par d'autres jeux si besoin.
* Il est facile d'ajouter d'autres jeux (ex. : Puissance 4, Sudoku) dans des dossiers Features/NouveauJeu/, avec leurs propres steps.
* Les règles peuvent être modifiées en adaptant seulement les méthodes liées dans la classe Grille.

**c) Stragégie BDD et bonnes pratiques**

**i) Langage ubiquitaire**

Nous avons utilisé un **vocabulaire métier** cohérent avec le jeu du Démineur :

* "case", "grille", "bombe", "drapeau", "révéler", "premier clic", "numéro", etc.

Chaque scénario est compréhensible par un utilisateur non technique.

**ii) Réutilisabilité des StepDefinitions**

Nous avons créé :

* des Given génériques ("une partie est en cours", "une case est marquée d'un drapeau")
* des Then réutilisables ("la case est révélée", "la partie est gagnée")

Et lorsque nécessaire, des Steps spécifiques pour les comportements propres ("toutes les bombes sont révélées").

**iii) Maintenance**

* **Structure en dossiers** : les fichiers .feature sont organisés par fonctionnalité (Initialisation, Drapeaux, Fin de partie, etc.)
* **Nommage clair** : des noms explicites pour chaque scénario et step.
* **Modularité** : chaque fonctionnalité a son propre fichier StepDefinitions.cs, ce qui facilite la lecture et la maintenance.

Cette organisation permet d'ajouter ou modifier les scénarios sans impact sur les autres modules du jeu.