Rapport Technique – 2048 Nature

Architecture Flutter et Choix d'Implémentation

Alexis Bouchendhomme FISA 5A

Lien Github: https://github.com/AlexisBCD/deux_mille_quarante_huit.git

1. Originalité et Innovations du Projet

1.1 Concept et Thématisation

2048 Nature transforme le jeu classique en une expérience de jardinage numérique, remplaçant les valeurs numériques par une progression végétale cohérente (🌱 → 🌿 → ☘️ → 🌾 → 🌸 → 🌺 → 🌳 → 🍀 → 🌻 → 🌷 → 🌹). Cette thématisation apporte une dimension immersive et relaxante au gameplay traditionnel.

1.2 Fonctionnalités Innovantes

Système de modes de jeu avancé :

- Classique 4×4 : Jeu traditionnel sans mécaniques spéciales
- Bonus/Malus 4×4 : Intégration de tuiles spéciales (5 % bonus ×2, 3 % gel temporaire)
- Extension 5×5 : Plateau agrandi pour gameplay prolongé et complexité accrue

Mécaniques de jeu étendues :

- **Tuiles bonus :** Multiplication ×2 du score lors de la fusion avec effet visuel doré
- **Tuiles gel :** Immobilisation temporaire des tuiles adjacentes pendant 3 tours avec bordure bleue
- Système de probabilités configurable par mode de jeu dans les GameSettings

Interface et expérience utilisateur :

- Thème visuel: Palette de verts naturels
- Audio contextuel : Musique d'ambiance (attention à l'originalité) en boucle + effets sonores (mouvement, fusion, game over)

2. Architecture Logicielle et Choix Techniques

2.1 Clean Architecture

La structure respecte une organisation modulaire stricte, garantissant la séparation des responsabilités selon les principes SOLID.

Arborescence :

```
None

lib/
├── core/services/ → Services transversaux (AudioService)
├── features/
| ├── home/ → Module sélection de mode
| |     ├── domain/ → GameMode enum, entités métier
| |     ├── application/ → HomeCubit, HomeState
| |     └── presentation/ → Pages, vues, widgets UI
| └── game/ → Module plateau de jeu
|       ├── domain/ → Entités (GameBoard, Tile, Direction)
|       ├── data/ → Repositories, data sources
|       ├── application/ → GameBoardCubit, états complexes
|       └── presentation/ → Interface utilisateur
```

Justification du choix :

- Maintenabilité : séparation claire entre logique métier et présentation
- Testabilité : use cases isolés
- Extensibilité : ajout de nouveaux modes sans impact sur l'existant

2.2 Pattern de Gestion d'État : BLoC/Cubit

Le choix s'est porté sur le pattern BLoC/Cubit.

Alternatives rejetées :

- Provider : trop simple pour la complexité des états de jeu
- setState : synchronisation impossible entre widgets (scores, plateau, audio)

Avantages du BLoC/Cubit :

- États immutables : prévention des effets de bord avec copyWith
- Flux unidirectionnel : traçabilité des changements d'état
- Réactivité : mise à jour automatique de l'interface via BlocBuilder

2.3 Flux de Gestion d'État Détaillé

Exemple concret : mouvement d'une tuile

1. GestureDetector détecte le swipe et en extrait une direction
2. Appel à context.read<GameBoardCubit>().move(direction)
3. Le cubit vérifie la validité de l'état, émet un état "loading", exécute le use case MoveTilesUseCase, déclenche les effets audio, puis publie un nouvel état mis à jour
4. BlocBuilder reconstruit automatiquement les widgets concernés
5. Les TileWidget s'animent vers leurs nouvelles positions avec AnimatedContainer

Cycle de vie des états :

- Idle : état stable, interactions utilisateur possibles
- Loading : traitement en cours, UI temporairement bloquée
- Updated : nouvel état propagé, animations déclenchées
- Error : message utilisateur pouvant être affiché en cas d'erreur

2.4 États Complexes et Synchronisation

Exemple de GameBoardState : plateau de jeu, configuration, état de traitement, messages d'erreur.

Avantages :

- Getters pratiques évitant la duplication de logique dans les widgets
- Synchronisation garanties

3. Widgets et Composants UI – Choix et Justifications

3.1 Module Home – Sélection de Mode

- HomePage : injection du HomeCubit via BlocProvider
- HomeView : StatelessWidget + BlocListener et BlocBuilder pour navigation réactive
- GameModeCard : AnimatedContainer pour retour visuel immédiat lors de la sélection

3.2 Module Game – Plateau de Jeu

- GameBoardPage : injection des use cases et du GameBoardCubit
- GameBoardView : WillPopScope pour confirmation et sauvegarde à la sortie
- GameBoardWidget : GridView.builder adaptatif (4×4 ou 5×5)
- GestureDetectorWidget : détection des swipes avec seuil calibré (500 px/s)
- TileWidget : AnimatedContainer affichant couleur, emoji et états (bonus, gel, normal)

3.3 Hiérarchie de Composition des Widgets

```
None
GameBoardView
 ├── GameHeaderWidget (scores, audio)
 ├── FlowerBackgroundWidget
 ├── GestureDetectorWidget
 │ └── GameBoardWidget
 │ └── TileWidget[]
 └── GameOverWidget (affiché uniquement si nécessaire)
```

3.4 Widgets Utilitaires et Services

- AudioControlWidget : PopupMenuButton contrôlant musique et effets via AudioService singleton

4. Services Techniques et Patterns

4.1 AudioService (Singleton Pattern)

Service centralisé gérant musique et effets sonores avec deux AudioPlayer distincts.

Avantages :

- Instance unique, évite la surcharge mémoire
- Séparation claire entre musique et effets
- Volume et préférences utilisateur persistants

4.2 Factory Pattern (GameSettings)

Chaque mode de jeu est configuré automatiquement via une factory, garantissant cohérence et sécurité.

4.3 Repository Pattern et Services Domaine

Services spécialisés :

- TileGenerationService : génération probabiliste des tuiles
- MovementService : déplacements et fusions
- FreezeEffectService : gestion des effets gel
- BoardUtilsService : utilitaires de manipulation du plateau

5. Optimisations de Performance

5.1 Reconstruction Sélective Widgets

BlocBuilder configurés avec buildWhen pour ne reconstruire que les widgets nécessaires (scores, plateau, fin de partie).

5.2 Gestion Mémoire et Ressources

- Libération des ressources audio dans dispose()
- BLoC automatiquement libéré par flutter_bloc

5.3 Animations Performantes

- AnimatedContainer avec durée optimisée (200 ms)
- GPU-accelerated pour fluidité

6. Analyse Comparative et Alternatives

6.1 Choix Architecturaux Justifiés

- BLoC vs Provider : gestion d'états complexes et testabilité accrue
- GridView vs Custom Painter : maintenance et animations facilitées

6.2 Limitations et Améliorations Possibles

Limitations : plateau limité (théoriquement extensible à 8×8), animations basiques, persistance simple avec SharedPreferences.

Un ralentissement est causé par le service audio lors du lancement d'une partie.

Améliorations envisagées : mode multijoueur via WebSocket, thèmes multiples, analytics intégrés.