README - Projet UBGarden

Date: 5 Mai 2025
Auteur(s): Thien An Truong, Paraschos Lucas Koumasonas
Projet A1-12

1. Introduction

Ce document détaille les modifications apportées au projet UBGarden. L'objectif principal était d'enrichir l'expérience de jeu en introduisant de nouvelles mécaniques, entités, et en améliorant la flexibilité de la configuration et du chargement des niveaux.

2. Modifications Architecturales Majeures

L'architecture initiale a été étendue pour supporter de nouvelles fonctionnalités. Les changements clés incluent :

- **Gestionnaire de Jeu (`GameEngine`) :**
  - **Logique de transition de niveaux :** Le `GameEngine` gère désormais le changement de niveau, incluant le nettoyage des sprites du niveau précédent et l'initialisation du nouveau (`initializeSpritesForLevel`, `checkAndHandleSwitchLevel`).
  - **Gestion dynamique des sprites :** Ajout et suppression des sprites en cours de jeu (ex: apparition d'abeilles/frelons, bonus).
  - **Création de la barre de statut :** La création des composants de la barre de statut a été déplacée dans `GameEngine` pour un meilleur contrôle, tandis que la classe `StatusBar` se concentre sur la mise à jour des valeurs.
- **Logique de Jeu (`Game`) :**
  - **Gestion des entités dynamiques :** `Game` maintient des listes d'abeilles (`Wasp`) et de frelons (`Hornet`) actifs, et gère leur cycle de vie (ajout, suppression, mise à jour via `updateDynamicEntities`).
  - **Mécaniques de jeu avancées :** Implémentation de la logique de complétion des carottes pour ouvrir les portes (`checkCarrotCompletion`), et de l'apparition aléatoire de bombes insecticides (`spawnRandomBomb`).
- **Héritage et Nouvelles Entités :**
  - **`Decor` et ses sous-classes :**
    - `Door` : Devient une sous-classe de `Decor` avec un état (ouverte/fermée) et une direction (vers niveau suivant/précédent). Sa logique de transition est gérée par le `Gardener`.
    - `NestWasp` / `NestHornet` : Sous-classes de `Decor` responsables de faire apparaître périodiquement des `Wasp` / `Hornet` et des bombes. Elles utilisent un `Timer` et une référence à `Game` pour interagir avec le système.
    - `Flowers`, `Hedgehog`, `Dirt` (anciennement `Land`) : Nouveaux types de `Decor`.
    - **Justification :** Factoriser le comportement commun des éléments de décor et permettre une spécialisation. L'accès à l'instance `Game` pour les nids est crucial pour le spawning.
  - **`Bonus` et ses sous-classes :**

    - `Carrot`, `InsecticideBomb`, `PoisonedApple` : Nouvelles sous-classes de `Bonus`, chacune avec un effet spécifique déclenché lors du ramassage par le `Gardener`.
    - **Justification :** Permet de définir des objets ramassables distincts avec des logiques d'effet encapsulées.
  - `GameObject` **et ses sous-classes (Mobiles) :**
    - `Wasp` / `Hornet` : Nouvelles sous-classes de `GameObject` implémentant `Movable`. Elles possèdent une logique de déplacement (avec tentatives de "smart move" pour éviter les blocages simples), d'attaque (`sting`), et de cycle de vie (points de vie pour `Hornet`, mort après piqûre pour `Wasp`).
    - **Justification :** Isoler la logique complexe des ennemis dans leurs propres classes.
  - `Sprite` **et ses sous-classes :**
    - `SpriteDoor`, `SpriteWasp`, `SpriteHornet` : Sous-classes de `Sprite` permettant une mise à jour dynamique de l'apparence en fonction de l'état de l'objet de jeu associé (ex: porte ouverte/fermée, direction de l'insecte).
    - **Justification :** Simplifie la logique de rendu et permet aux sprites d'avoir un comportement visuel spécifique sans surcharger `SpriteFactory`.
- `Gardener` **(Joueur) :**
  - **Mécaniques étendues :**
    - Gestion de l'énergie avec récupération passive et coûts variables selon le terrain et l'état de maladie.
    - Système de maladie (pommes empoisonnées) avec effets cumulatifs et drainage d'énergie.
    - Inventaire de bombes insecticides.
    - Logique de demande de changement de niveau (`requestSwitchLevel`, `isSwitchLevelRequested`).
  - **Implémentation de** `WalkVisitor` **:** Permet de définir finement sur quels types de `Decor` le `Gardener` peut marcher (ex: portes ouvertes, pas sur les arbres ou les nids).
    - **Justification :** Centralise la logique de déplacement et d'interaction du joueur, rendant le `Gardener` plus autonome.
- **Configuration (`Configuration`) :**
  - Fortement étendue pour inclure de nombreux nouveaux paramètres (fréquence de déplacement/apparition des insectes, dégâts, cooldowns, capacité de bombes du joueur, etc.).
  - **Justification :** Augmente la flexibilité et facilite l'équilibrage du jeu sans recompiler.
- **Chargement des Niveaux (`GameLauncher`) :**
  - Support du chargement de plusieurs niveaux à partir de fichiers `.properties`.
  - Implémentation d'un système de décompression simple pour les chaînes de caractères des cartes.
  - Détection et placement initial du `Gardener` au niveau 1.
  - Gestion des erreurs améliorée lors du chargement.
  - **Justification :** Permet une création de niveaux plus complexe et découplée du code source.
- **Enumérations :**
  - `MapEntity` : Étendue avec de nouvelles entités.

- `Direction` : Ajout des méthodes `opposite()`, `turnLeft()`, `turnRight()` pour faciliter la logique de déplacement des IA.

3. Fonctionnalités Implémentées (Ce qui fonctionne)

Toutes les fonctionnalités décrites par l'analyse du code fourni sont considérées comme fonctionnelles :

- **Joueur (`Gardener`) :**
  - Déplacement (haut, bas, gauche, droite) avec gestion des collisions et des coûts d'énergie.
  - Ramassage de bonus :
    - Pomme (`EnergyBoost`): Restaure l'énergie, guérit les maladies.
    - Pomme Empoisonnée (`PoisonedApple`): Applique un effet de maladie temporaire et cumulatif.
    - Carotte (`Carrot`): Ramasser toutes les carottes d'un niveau ouvre la porte de sortie.
    - Bombe Insecticide (`InsecticideBomb`): Ajoute une bombe à l'inventaire du joueur (capacité limitée).
  - Perte d'énergie due au déplacement, aux maladies, et aux piqûres d'insectes.
  - Récupération passive d'énergie si immobile.
  - Utilisation de bombes pour se défendre (automatique lors d'une collision avec un insecte).
  - Mort si l'énergie atteint 0.
  - Victoire en atteignant le `Hedgehog`.
- **Insectes :**
  - **Abeilles (`Wasp`) :**
    - Apparition depuis `NestWasp`.
    - Déplacement semi-aléatoire (tente d'éviter de se bloquer).
    - Pique le joueur au contact (inflige des dégâts, l'abeille meurt).
    - Meurt si elle entre en collision avec une bombe insecticide au sol ou si le joueur utilise une bombe lors d'une collision.
  - **Frelons (`Hornet`) :**
    - Apparition depuis `NestHornet`.
    - Déplacement similaire aux abeilles.
    - Possède plusieurs points de vie (2) et peut piquer plusieurs fois (2) avec un cooldown entre les piqûres.
    - Meurt après avoir utilisé toutes ses piqûres ou perdu tous ses points de vie (par des bombes).
    - Est blessé par les bombes (au sol ou utilisées par le joueur).
- **Nids (`NestWasp`, `NestHornet`) :**
  - `NestWasp` : Fait apparaître une `Wasp` et une bombe insecticide aléatoire sur la carte périodiquement.
  - `NestHornet` : Fait apparaître un `Hornet` et deux bombes insecticides aléatoires sur la carte périodiquement.
- **Niveaux et Environnement :**

    - Chargement de cartes multi-niveaux à partir de fichiers `.properties` (avec support de compression basique).
    - Chargement d'une carte par défaut si aucun fichier n'est spécifié.
    - Portes (`Door`) : Permettent de passer d'un niveau à un autre. Les portes "fermées" (`DoorNextClosed`) s'ouvrent lorsque toutes les carottes du niveau sont ramassées.
    - Types de terrain : Herbe (`Grass`), Terre (`Dirt`/`Land`), Arbres (`Tree` - obstacle), Fleurs (`Flowers` - obstacle).
    - Affichage du niveau actuel, de l'énergie, du nombre de bombes et du nombre d'effets de poison actifs dans la barre de statut.
- **Interface Utilisateur (UI) :**
    - Affichage graphique du jeu.
    - Barre de statut mise à jour en temps réel.
    - Messages de victoire ("Victoire!") et de défaite ("Perdu!").
    - Menu pour charger une partie (depuis un fichier ou par défaut) et quitter.

4. Informations Spécifiques / Remarques Utiles

- **Format des Fichiers de Carte :** Les cartes sont définies dans des fichiers `.properties`.
    - `levels=N` : Nombre total de niveaux.
    - `levelX=...` : Chaîne de caractères représentant le niveau X. Les lignes sont séparées par 'x'.
    - `compression=true/false` : Si `true`, la chaîne `levelX` peut utiliser une compression RLE simple (ex: `G3` -> `GGG`).
    - D'autres propriétés de configuration du jeu (énergie, fréquences, etc.) peuvent aussi être définies dans ce fichier.
- **Exécution :** Le jeu se lance via la classe `fr.ubx.poo.ubgarden.Main`.
- **Configuration :** De nombreux aspects du jeu (énergie du joueur, dégâts des ennemis, timers, etc.) sont configurables via le fichier `.properties` ou par défaut dans `GameLauncher` si non spécifiés.
