Documentation développeur

Cette section s'adresse à toute personne souhaitant reprendre, améliorer ou prolonger le projet *Ant Colony Rush*. Elle vise à orienter rapidement un(e) développeur(euse) dans la structure du code, les points d'entrée du programme, les constantes clés, et les fonctionnalités à développer en priorité.

1. Classes importantes à explorer en premier

Voici les classes essentielles pour comprendre le fonctionnement global du jeu :

- **`StartMenuController`** : point d'entrée de l'interface, gère le menu principal, la sélection de la difficulté, et le lancement de la partie. C'est ici que la méthode `launchGame()` initialise tous les éléments du jeu (terrain, score, vue, threads, etc.).
- **`JeuFrame`** : classe centrale de l'interface graphique. Elle contient les composants de la vue (terrain, panneaux de contrôle, chronomètre, boutons pause/reprise, etc.).
- **`TerrainController`** : gère les clics sur le terrain, les interactions avec les objets (Nid, abris, ressources) et déclenche les déplacements des fourmis.
- **`ThreadSet`** : conteneur central qui instancie et démarre les threads métiers (énergie, déplacement, ressources, etc.).
- **`Terrain`** : classe principale du modèle. Contient les objets fixes, les fourmis, les ressources et les déplacements en cours.

2. Où se trouve la méthode `main` ?

Le point d'entrée du programme se trouve dans la classe `Main.java` (non incluse ici mais généralement minimaliste), qui instancie une `MenuDemarrage` et un `StartMenuController`.

3. Constantes principales à modifier

Le comportement du jeu peut être ajusté en modifiant certaines constantes situées dans les classes suivantes :

- **`Fourmi`** : `MAX_ENERGIE`, vitesse, seuil de mort.
- **`Deplacement`** : `VITESSE` pour ajuster la rapidité des mouvements.
- **`Crapaud`** : constantes de satiété, temps de sommeil, champ de vision.
- **`Terrain`** : dimensions de la carte, nombre maximal de ressources.

— `ThreadSet` ou `GestionnaireRessources` : fréquence de génération des ressources.

Ces constantes permettent d'équilibrer la difficulté, la dynamique du jeu ou d'ajuster les performances.

4. Fonctionnalités à développer en priorité

Plusieurs fonctionnalités ont été préparées dans le code mais pas entièrement développées. Voici celles qu'il serait pertinent de terminer ou améliorer :

— **Animation de pousse des ressources** :
  — Actuellement, les ressources apparaissent instantanément. Il serait intéressant de les faire pousser progressivement avec une animation.
  — Idée d'implémentation : ajouter un champ `croissance` dans la classe `Ressource`, gérer son évolution dans `GestionnaireRessources`, et modifier l'apparence dans le `TerrainPanel`.
— **Sauvegarde automatique des meilleurs scores** :
  — Prévoir une classe `SauvegardeScore` qui lit/écrit un fichier texte ou JSON.
  — L'appel peut être fait à la fin d'une partie, via un menu de fin de jeu ou dans le `JeuFrame`.
— **Équilibrage de la difficulté** :
  — Actuellement, la difficulté affecte principalement le nombre d'éléments au départ. Il serait intéressant qu'elle modifie aussi la vitesse du crapaud, le temps de pousse des ressources ou la vitesse d'épuisement des fourmis.
— **Gestion de fin de partie** :
  — Actuellement, il n'y a pas de condition de victoire ou de défaite.
  — Une fin de partie pourrait être déclenchée si le crapaud mange trop de fourmis, ou si le joueur atteint un certain score.

5. Conseils pour reprendre le développement

— Commencer par faire tourner le jeu depuis le menu de démarrage, et observer les interactions de base.
— Lire les classes `StartMenuController`, `JeuFrame`, et `ThreadSet` pour bien comprendre l'initialisation du jeu.
— Utiliser les `Gestionnaires` métiers comme base pour ajouter des fonctionnalités autonomes (ex. un gestionnaire de météo).
— La logique de `pause/reprise` est déjà centralisée dans le `PauseController` et facilement extensible .