**Document Technique pour PathfindingAStar.cs**

**Nom du script** : PathfindingAStar

**Namespace** : Tools

**Vue d'ensemble :**

Le script PathfindingAStar implémente l'algorithme de pathfinding A\* pour trouver un chemin entre deux points sur une grille. Il prend en compte les obstacles et les tuiles actives pour déterminer le meilleur chemin possible.

**Composants clés :**

* **Variables privées** :
  + TileSystem tileSystem : Instance de TileSystem utilisée pour obtenir des informations sur les tuiles et leur état.

**Méthodes principales :**

* **Constructeur** :
  + public PathfindingAStar(TileSystem \_tileSystem) :
    - Initialise le système de pathfinding avec une instance de TileSystem.
* **Méthodes publiques** :
  + public List<Vector3Int> FindPath(Vector3Int start, Vector3Int target, bool IsConsideringObstacle) :
    - Trouve le chemin du point de départ au point cible.
    - Utilise l'algorithme A\* pour explorer les voisins, calculer les scores, et reconstruire le chemin une fois le cible atteint.
* **Méthodes privées** :
  + private List<Vector3Int> GetNeighbors(Vector3Int current, bool IsConsideringObstacle) :
    - Retourne les voisins de la position actuelle en tenant compte des obstacles si nécessaire.
  + private bool IsTileActive(Vector3Int position) :
    - Vérifie si une tuile est active (non nulle) sur la grille.
  + private bool IsTileWalkable(Vector3Int position) :
    - Vérifie si une tuile est praticable en utilisant la méthode Ground.VerifieWalkability.
  + private float Heuristic(Vector3Int a, Vector3Int b) :
    - Calcule la distance heuristique (Manhattan) entre deux points pour estimer le coût du chemin.
  + private Vector3Int GetLowestFScore(HashSet<Vector3Int> openSet, Dictionary<Vector3Int, float> fScore) :
    - Retourne le voisin avec le score f le plus bas parmi les voisins ouverts.
  + private List<Vector3Int> ReconstructPath(Dictionary<Vector3Int, Vector3Int> cameFrom, Vector3Int current) :
    - Reconstruit le chemin en remontant les positions depuis la cible jusqu'au point de départ en utilisant le dictionnaire cameFrom.

**Remarques :**

* **Gestion des Objets Actifs** :
  + IsTileActive vérifie si une tuile existe sur la grille. Cette vérification est utilisée pour éviter d'explorer des tuiles non présentes.
* **Praticabilité des Tuiles** :
  + IsTileWalkable utilise la méthode Ground.VerifieWalkability pour s'assurer que les tuiles sont praticables. Assurez-vous que cette méthode est correctement implémentée pour refléter les propriétés du terrain dans le jeu.
* **Calcul de la Distance** :
  + La distance heuristique est calculée en utilisant la distance de Manhattan, ce qui est approprié pour les mouvements sur une grille de type orthogonal. Si vous utilisez une grille hexagonale ou un autre type de grille, ajustez cette méthode en conséquence.
* **Performance** :
  + Le script utilise des structures de données efficaces comme HashSet et Dictionary pour gérer les ensembles ouverts et fermés, ainsi que les scores, ce qui améliore la performance de l'algorithme A\*.

Ce document fournit une vue d'ensemble du fonctionnement du PathfindingAStar, expliquant comment il gère la recherche de chemins sur une grille tout en prenant en compte les obstacles et la praticabilité des tuiles. Assurez-vous que les méthodes auxiliaires comme Ground.VerifieWalkability sont correctement définies pour que le système fonctionne comme prévu.