Exo 3 : Pathfinding avancé

L’idée est d’implémenter un algorithme A\* pour le déplacement des unités.

On sait qu’il y a une partie terrestre et une partie maritime. Il faudra donc distinguer les cases terrestres des cases maritimes (avec une énumération GROUND et WATER par exemple) mais également les cases correspondant à des ponts.

On sait qu’il existe plusieurs îles reliées par des ponts, qui constituent donc le seul passage d’île en île pour certaines unités terrestres. Il faudra définir les ponts (checkpoints) à emprunter dans le pathfinding => comme à l’image de l’exo 2, utilisation d’A\* pour générer plusieurs sous-chemins, il faut donc connaître à l’avance où se situent les ponts sachant que ceux-ci peuvent être bloqués ou détruits, il faudra aussi penser à mettre à jour les sous-chemins.

On sait que chaque unité a ses propriétés de déplacement particulières suivant la topologie du terrain et qu’en fonction, une unité pourra se déplacer jusqu’à un certain dénivelé voire pas du tout. Il faut donc, pour chaque unité, définir deux variables (pour le dénivelé « terre ferme » et celui « eau ») qui indiqueront le dénivelé max qu’une unité donnée peut franchir. On donnera comme valeur « -1 » pour indiquer qu’il n’y a pas de contraintes topologiques.

Pour les unités basiques :

* Ne pas traiter les cases adjacentes que l’unité ne peut pas franchir (WATER).
* Entre 2 cases adjacentes, calculer le dénivelé : soit h la hauteur d’une case, si on a : |case\_courante.h – voisin.h | <= 3, alors la case voisine peut-être considée comme à traiter.
* Qu’une case ne soit pas déjà occcupée par une unité => si occupée, on ne considère pas la case, sinon, on la considère comme à traiter.
* Doivent suivre plusieurs sous-chemins pour aller d’île en île.

Pour les unités amphibies :

* Entre 2 cases adjacentes, calculer le dénivelé : soit h la hauteur d’une case, si on a : |case\_courante.h – voisin.h | <= 2, alors la case voisine peut-être considée comme à traiter.
* Qu’une case ne soit pas déjà occcupée par une unité => si occupée, on ne considère pas la case, sinon, on la considère comme à traiter.

Pour les unités crawlers :

* Ne pas traiter les cases adjacentes que l’unité ne peut pas franchir (WATER).
* Considérer chaque case avec un h = 1 puisque les unités font abstraction de la topologie.
* Doivent suivre plusieurs sous-chemins pour aller d’île en île.

Pour les unités aériennes :

* Qu’une case ne soit pas déjà occcupée par une unité => si occupée, on ne considère pas la case, sinon, on la considère comme à traiter
* Considérer chaque case avec un h = 1 puisque les unités font abstraction de la topologie.

A\* considèrera la case à traiter en priorité dans la liste des cases à traiter où le coût total (coût so far + coût h de la case) est le moins élevé, et sinon, en cas d’égalité, où le coût h est la moins élevée.

*Note : On considère que 2 unités ne peuvent pas occuper les mêmes coordonnées.*