Groupe F2

Choix de modélisation

Element

Nous avons fait le choix qu'un élément est un objet caractérisé par une ID qui permet de définir ce qu'il représente. (Rocher, coffre, clé ,trésor, navire etc...)

Parcelle

Nous avons fait le choix qu'une parcelle était un object qui contenait une liste d'Elements (potentiellement vide) ,un personnage (potentiellement null) et deux boolean pour les pièges (un pour chaque équipe).

Accessibilité des rochers

Nous avons fait le choix de nous assurer que seuls les élèments importants (navires, coffre et clé) devait être accessible sur l'île. Par conséquent certains des rochers de la carte ne sont pas accessible pour les personnages. Pour cela, nous avons un algorithme qui part du premier navire, et qui marque comme accessible la case actuelle et toutes les cases adjacente contenant un élément (rocher,navire,eau) et réinvoque cette méthode sur les cases adjacentes vide qui n'ont pas deja été marqué comme accessible (qui ne contienne rien). Ainsi toutes les cases que l'on peut atteindre à partir du premier navire sont marquées comme étant accessible. Nous vérifions ensuite, que les cases importantes sont elles aussi marquées , le cas échant nous regénerons une nouvelle île (max 100 générations, si aucune génération n'est trouvé en 100 essais nous demandons à l'utilisateur de réessayer). Cette méthode nous permet d'atteindre avec des îles de 15x15 des pourcentage de 30% de rochers très souvent (pouvant aller même jusque 45% mais moins régulièrement).

Validité des déplacements

Lorsque nous demandons à l'utilisateur de choisir un personnage à déplacer ou sa destination nous vérifions immédiatement que la case choisi est correcte. Dans le cas d'un personnage, nous vérifions que la case comporte un personnage et qu'il appartient au joueur. Dans le cas de la destination, nous vérifions que la distance à parcourir est possible pour le personnage choisi et qu'elle contient un élèment avec lequel le personnage peut intéragir. Ainsi, nous pouvons détecter un choix incorrect le plus tôt possible et demander à l'utilisateur d'en refaire un. De plus, cela nous a facilité notre fonction de déplacement, car nous n'avions que des coordonées correctes et aucune vérification n'était à faire.

Notion d'équipe

Nous avons choisi de créer une classe Equipe qui gère le nombre de personnage en vie dans l'équipe mais ne gère pas les objets directement. En effet, lorsqu'un personnage est crée (instancié) son attribut "equipe" le rattache à son équipe , ce qui nous permet d'identifié l'équipe à laquelle il appartient directement sur le plateau de jeu. De plus, lorsque les équipes sont composées, à chaque fois qu'un joueur ajoute un personnage (qu'il le crée) le compteur de personnage dans l'équipe est incrémenté. Lorsque qu'un personnage meurt, le compteur de personnage de l'équipe à laquelle il appartient (1 ou 2) est décrémenté.

Repos

La classe Equipe, ne gère pas l'ensemble des personnages de l'équipe néanmoins, elle gère les personnages "au repos" dans le navire. En effet, elle contient une List de personnage dans laquelle sont stockés les personnages qui sont dans le navire. Ainsi lorsqu'un personnage intéragit avec **son** navire (qu'il monte dedans), il est ajouté dans la liste de personnages au repos de son équipe et retiré du plateau de jeu. Tous les personnage qui sont la liste de personnages au repos d'une équipe récupère 10 points d'énergie au début de chaque tour.
Lorsque les équipes sont composées, l'ensemble des personnages sont considérés comme "au repos" et sont stockés dans le navire avant d'accoster sur l'île

Piège

Les pièges sont caractérisés par un boolean piege dans chaque parcelle. Par conséquent, si un personnage se déplace sur une parcelle où se boolean est à vrai, alors il incrémente son compteur "nbtourspiege" de 5. Ce compteur est décrémenté de 1 à chaque début de tour.

Stratégie de l'IA

Nous avons implémenté une IA qui execute une seule stratégie (nous aurions souhaité en ajouté d'autres, mais malheureusement nous avons rencontré des difficultés avec l'algorithme de pathfinding qui nous a retardé dans notre tâche). Nous avons fait le choix de composer l'équipe de l'IA de 5 explorateurs afin de chercher au plus vite le clé, puis de récupérer le coffre et enfin de retourner au navire. Pour cela, tant que la clé est sur l'île, nous choisissons le personnage le plus proche de la clé et nous lui indiquons la case adjacente qui propose le plus court chemin pour aller prendre la clé. Lorsque la clé n'est plus sur l'île, nous choissisons le personnage disposant de la clé , sinon le plus proche(ce qui dans les faits est inutile mais cela emêche l'IA d'abandonner et rend les parties plus longues) et l'envoyons vers le coffre. Lorsque le coffre n'est plus sur l'île, si un des personnages dispose du trésor alors il se dirige vers le navire, sinon l'IA passe son tour.

Algorithm de Pathfinding

Pour réaliser les déplacements de l'IA, nous avons choisi d'implémenter une version de l' "A* algorithm" que nous avons faites nous même. Cela rend l'IA très dure à battre mais le challenge en terme de programmation semblait intéressait et il nous a permis d'appliquer avec un cas concret nos cours de graphe. En effet, l'algorithme de Dijkstra qui calcule un plus court chemin d'une source à une destination est une instance de l' A* algorithm. Cette algorithme calcule le cout de chaque case voisine de la case courante et choisi le voisin le moins couteux pour être sa nouvelle case courante jusqu'à ce que la case courante soit la case recherchée.