Biguenet Denis TP1
Séverin Gosset TP4

Rapport l'age des barons

Sommaire

Biguenet Denis TP1
Séverin Gosset TP4

Fonctionnement général

L'équipe qui joue en premier lors de chaque tour est aléatoire, chaque joueur peut donner des ordres à ses agents comme se déplacer, ne rien faire ou disperser l'unité, créer un château…
Les ordres sont exécutés à la fin du tour des deux équipes, à chaque tour on met à jour la fenêtre pour afficher les agents.

Les agents peuvent combattre d'autres agents par déplacements, l'issu du combat est aléatoire, mais les barons sont plus puissants que les guerriers, les manants eux, meurent si ils rencontrent un guerrier ou baron ennemi, mais peuvent combattre d'autres manants.
Les agents peuvent aussi détruire les châteaux adverses par déplacements, si celui contient une garnison (agents sur la même case) alors un combat s'ensuit, si un château n'a pas de garnison et qu'un adversaire l'attaque, le château est détruit et tous ses agents de types guerriers et barons le sont aussi. Les manants eux, sont rattachés au château de l'agent qui a détruit le château.
Le jeu continue jusqu'à ce qu'une équipe n'aie plus de châteaux.

On peut aussi, à chaque début de tour, sauvegarder la partie, pour pouvoir la reprendre plus tard.

Biguenet Denis TP1
Séverin Gosset TP4

Manuel utilisateur

Tours : chaque tour vous pouvez ordonner à vos agents de se déplacer, les barons peuvent se battre, créer un château et se déplacer, les guerriers peuvent aussi se battre et se déplacer, le manant peut produire de l'or pour cela il faut lui ordonner de ne rien faire. Attention car une fois cet ordre donné, il ne pourra bouger.

Déplacements : pour se déplacer il faut cliquer sur la case où l'on souhaite que l'agent aille. Attention ! On ne peut pas ordonner à un agent d'arrêter l'ordre en cours.

Combats : un combat s'enclenche quand deux agents adverses se trouvent sur la même case. Ils sont résolus de manière aléatoire, cependant les barons sont plus puissants que les guerriers. Tandis que les manants se font tuer s'ils rencontrent un agent ennemi (si il rencontre un manant ennemi il se bat avec).
On peut aussi détruire un château ennemi, pour cela il faut que le château soit vide (ou bien le château n'est pas protégé ou bien vos agents ont gagné le combat se situant sur le château), lorsqu'il est détruit tous ses agents de type barons et guerriers sont détruits, ses manants sont rattachés à votre équipe.

Production : Chaque château peut produire un agent (baron, guerrier ou manant), le baron met 6 tour à produire, le guerrier 4 et le manant 2, tous ces agents coûtent de l'or au recrutement.
On peut aussi décider que le château n'a rien à produire.

Conditions de victoires : pour gagner la partie il faut éliminer les châteaux de l'équipe adverse.

Biguenet Denis TP1
Séverin Gosset TP4

Fonctions principales

Aliste insertion_triee(Monde * partie, char genre, AListe château) :

Cette fonction permet d'ajouter un agent dans une équipe, on alloue d'abord l'espace mémoire pour l'agent, puis l'insère dans la liste dans cet ordre.
Cette liste contient d'abord les barons puis les guerriers puis les manants
Elle renvoie l'adresse de l'agent créé.

Elle est de complexité O(1) au mieux, O(k) au pire, avec k le nombre d'agent lié au château.

void ajoute_voisin(Monde *partie, Agent *ag, int x, int y) :

Ajoute un agent dans une liste de voisins
Si un agent se déplace sur une case où il existe déjà un agent, il est inséré dans le voisinage grâce à vsuiv et vprec
Les agent qui ne sont pas de la même équipe que l'agent sur laquelle la case pointe sont placés après cette équipe.

Elle est de complexité O(1) au mieux et O(k) au pire, avec k le nombre d'agents dans la liste de voisins

void affiche_contenu(Monde partie)

Affiche le contenu de chaque (les agents présent dessus) et affiche une image sur la grille si un agent est présent dessus.

Sa complexité est O((i*j)+k) avec i le nombre de case en largeur et j le nombre de case en longueur et k le nombre d'agent car cette fonction parcourt aussi les listes de voisins de chaque case.

void supprimer_voisin(AListe ag, Monde *partie)

Supprime un agent d'une liste de voisins, par exemple s'il se déplace ou s'il est tué lors d'un combat.
On extrait alors l'agent de cette liste.
La complexité de cette fonction est O(1)

AListe supprime_chateau(Monde *partie, AListe château)

Biguenet Denis TP1
Séverin Gosset TP4

Supprime un château d'une équipe si celui ci est détruit par un agent adverse, si le premier château est détruit, la liste de l'équipe pointe sur son suivant, on extrait le château de la liste puis on libère l'espace mémoire.
Elle renvoie la liste des agents étant liés à ce château.
Sa complexité est O(1)

void rattache_agent(Monde *partie, AListe lst_agent, Agent *ag)

Supprime les guerriers et les barons liés au château pris en argument et rattache les manants au château de l'agent qui a détruit le château.
Sa complexité est de O(k+n) avec k le nombre d'agents dans la liste d'agents à supprimer/rattacher et n le nombre d'agent après l'agent qui a détruit le chateau.

void combat(Monde *partie, int x, int y)

Résous les affrontements entre deux les agents des deux équipes présents sur les cases jusqu'à ce que l'une des deux équipes ai gagné.
Si l'équipe adverse gagne et qu'il y a un château, on le détruit.

Sa complexité est de O(k) avec k la taille de la liste des voisins

void deplacement_agent(Monde *partie, Agent *ag)

Permet de déplacer les agents selon leur destination et de gérer les listes de voisins (enlever un voisin, en ajouter un)
Sa complexité est O(1)

Aliste ajout_chateau(Monde *partie, AListe ag)

Permet de créer un château et de l'insérer dans la liste de l'équipe.
Elle prend en argument un agent (un baron) et place le château sur la case où celui ci est placé.
Le château est placé à la fin de la listes de voisinage de chateaux d'une équipe
Sa complexité est de O(k) avec k le nombre de château.

void donne_ordre(Monde * partie, char couleur)

Permet de donner des ordres aux agents, elle parcourt chaque liste d'agents liée à chaque château.Si l'agent exécute déjà un ordre alors on ne peut pas lui en affecter un nouveau.
Sa complexité est de O(n) avec n le nombre d'agent dans l'equipe car on parcourt

Biguenet Denis TP1
Séverin Gosset TP4

chaque agent de l'équipe pour vérifier si il a déjà un ordre ou non.

void execute_ordre(Monde * partie, char couleur)

Permet d’exécuter l'ordre donné à un agent, on parcourt chaque liste d'agent liée à chaque château dans l'équipe, selon l'ordre on exécute tel ou tel mouvement ou action.
Puis on reparcourt la boucle pour vérifier s'il n'y a pas de combat à faire, en effet, il faut attendre que tous les agents se soient déplacer pour pouvoir respecter les priorités du combat (les guerriers d'abord puis les barons etc.) ce qui fait une complexité de O(k) avec k le nombre d'agent total.

void sauvegarde(FILE * fic, Monde * partie, char couleur)

Sauvegarde la partie actuelle dans un fichier. Elle prend le fichier à sauvegarder, la partie et la couleur à faire jouer en premier en argument.
On parcourt la liste de tout les agents de la partie, en écrivant dans le fichier les caractéristiques de l'agent. Les agents liés à un château son écrits juste après ce château.

Sa complexité est toujours de O(k) avec k le nombre d'agents dans la partie.

char charge(FILE * fic, Monde * partie)

Charge la partie à partir d'un fichier dans lequel a été sauvegardé une partie. Elle prend le fichier à sauvegarder et la partie en argument.
On parcourt le fichier en ajoutant à chaque équipe ses agents, en respectant les allégeances de chaque agent et les chateaux.

Sa complexité est toujours de O(k) avec k le nombre d'agents dans la partie.

Biguenet Denis TP1
Séverin Gosset TP4

Répartition du travail

Nous avons effectué la partie 1, 2 et 3 ensemble.
Pour la suite nous nous sommes réparti le travail : Séverin a fait la partie 4 et Denis la partie 5, puis nous avons debugué et épuré le code ensemble après avoir fait ces deux parties.
Nous avons aussi commenté ensemble nos fonctions.