Luttes du mido-age

Projet Programmation C, L2 MIDO 2020-2021

Résumé

La maison bleue (Stark) et la maison rouge (Lannister) se disputent un territoire. Le but est de simuler ce combat qui s'effectue tour par tour.

1 Dates importantes

- ▷ Deadline composition des groupes : 18 décembre 2020 à 23h59
- ▷ Rendu du projet : avant le début des premières soutenances
- ▷ Soutenances : probablement fin janvier

2 Versions

Ce sujet est susceptible d'être modifié.

- ▷ 11 décembre 2020 : Mise en ligne du sujet.

3 Description

Deux groupes (bleus et rouges) sont présents. Chaque groupe comporte 4 types d'éléments :

1. Les *châteaux*, qui ne se déplacent pas et produisent les autres éléments.
2. Les *seigneurs*, qui se déplacent et attaquent et qui peuvent devenir des châteaux (et devenir sédentaires).
3. Les *guerriers*, qui se déplacent et attaquent.
4. Les *manants*, qui se déplacent, attaquent et produisent des ressources.

Dans la suite, on appelle *agent* les éléments qui se déplacent. Un agent ne peut se déplacer que d'*une case*, vers une des 8 cases voisines.

3.1 Initialisation

Le plateau de jeu est considéré comme un quadrillage de cases carrées, où chaque case possède 8 cases adjacentes (sauf pour les bords, qui ne sont pas franchissables).

On fait jouer une couleur l'une après l'autre. Au sein d'une couleur, on demande une action à effectuer pour chaque élément du groupe. On commence par demander l'action pour chaque château en attente, puis l'action pour chaque agent en attente.

Initialement, il y a un château, un seigneur et un manant pour chaque couleur. Ces trois éléments pour la couleur rouge sont placés dans le coin nord-ouest du plateau de jeu (la case du coin pour le château et ses deux cases voisines (horizontalement et verticalement) pour les deux agents), la couleur bleue se place symétriquement dans le coin sud-est du plateau. Chaque couleur possède un trésor (alimenté par les manants), initialement fixé à 50.

3.2 Châteaux

Un château peut produire :

- ▷ Un seigneur (prend 6 tours de jeu et coûte 20 dans le trésor).
- ▷ Un guerrier (4 tours et coûte 5).
- ▷ Un manant (2 tours et coûte 1).

Le choix de la production ne peut être changé avant la fin de sa réalisation. Une fois l'agent produit, le château est en attente d'un nouvel ordre de production. Un château est lié à tous les agents produits par celui-ci – ainsi, si un château disparaît, les seigneurs et les guerriers préfèrent se suicider que de passer chez l'ennemi et disparaissent aussi ! Par contre, les manants n'ont aucun honneur et changent d'allégeance. Ils sont alors reliés au château de l'agent qui a détruit leur ancien château. Lorsqu'un seigneur se transforme en château, il n'est plus attaché à l'ancien château.

3.3 Mobilité

Deux châteaux ne peuvent pas se situer sur une même case.

Chaque agent possède une destination. Si sa position actuelle est différente de sa destination, il se déplace d'une case vers sa destination en suivant un des chemins le plus court. Un agent ne peut pas recevoir de nouvel ordre d'immobilité ou de nouvelle destination avant d'être arrivé à sa destination. Sinon il attend un nouvel ordre pouvant être :

- ▷ Son suicide.
- ▷ Une nouvelle destination.
- ▷ Son immobilité.

Un manant immobile le reste jusqu'à la fin du jeu ou jusqu'à sa destruction par l'ennemi. Un manant immobile ajoute 1 au trésor de sa couleur à chaque tour de jeu. Il redevient mobile lorsqu'il change d'allégeance (c'est-à-dire lorsque son château est détruit par l'ennemi). Un manant immobile a, par convention, des coordonnées de destination négatives.

Un seigneur immobile se transforme en château au prix de 30 et s'il n'y a pas de château déjà présent sur sa case (et si le trésor est suffisant !).

3.4 Combats

Si un agent veut se placer sur une case occupée par des éléments d'une couleur différente de la sienne, un combat a lieu. Son issue est déterminée par un tirage aléatoire (fonction `random` par exemple, voir TPs) pondéré par le coût de production de l'agent ou du château. Ainsi, par exemple, un seigneur (coût de production : 20) attaquant un château (coût de production : 30) a 20

chances sur 50(=20+30) de l'emporter, i.e. 40% de chance. À la fin du combat, le perdant est détruit. Si plusieurs éléments d'une même couleur sont sur une même case (niveau 4), il y a combat jusqu'à destruction d'une des couleurs (les guerriers devront intervenir en premier, puis les seigneurs, puis les manants, puis les châteaux). Par exemple, un guerrier rouge attaque un château bleu défendu par un manant. Le premier combat est entre le guerrier et le manant. Si le guerrier en sort gagnant, le manant est détruit et le second combat est entre le guerrier et le château.

On proposera aussi un mode où il est possible de choisir l'issue des combats.

3.5 Fin de jeu et scores

La partie est terminée si une des couleurs ne possède plus de château. Le joueur pourra également forcer la fin de partie. Un score (défini selon votre choix) sera calculé à la fin de la partie. On sauvegardera les 10 meilleurs scores (avec le nom du joueur) dans un fichier sur l'ordinateur (possibilité de voir ces scores dans l'ordre depuis votre programme).

Une fois la partie finie, votre programme devra libérer proprement toute la mémoire allouée.

3.6 Sauvegarde et chargement

Au début de chaque tour d'une couleur, on peut choisir de sauvegarder la partie en cours dans un fichier en local sur l'ordinateur (sur le disque dur) avec l'extension `.got`. L'utilisateur pourra choisir le nom de ce fichier. On devra pouvoir choisir de charger une partie mémorisée.

On peut supposer qu'un agent est rattaché au château précédent dans l'ordre du fichier.

Le format de sauvegarde est fixé et doit être compatible entre les différents projets. La première ligne contient `R` ou `B` en majuscule (pour la couleur du premier groupe devant jouer) suivi de son trésor. La seconde ligne contient l'autre couleur, suivi de son trésor. Chaque ligne suivante représente un élément du jeu avec ses caractéristiques dans cet ordre :

- ▷ son groupe (`R` ou `B` en majuscule),
- ▷ son type (les lettres suivantes minuscules `c`, `s`, `g` ou `m`),
- ▷ sa position actuelle (deux entiers),
- ▷ si c'est un agent, sa destination (deux entiers) (rappel, un manant immobilisé a une destination négative),
- ▷ si c'est un château, le type et le temps de production effectué de la production actuelle (une lettre et un entier).

Un agent est relié au premier château de sa couleur qui le précède dans le fichier.

4 Ce qui est demandé

Toute question sur le projet devra être posée dans l'équipe Teams (et non par mail ou en individuel).

4.1 Structures

Tous les éléments devront être représentés par une même structure `Personnage`. On utilisera une structure `Monde` regroupant le plateau et deux pointeurs vers les châteaux rouge et bleu initiaux (ce qui est noté Rouges et Bleus sur le dessin). Le plateau est un tableau à deux dimensions de `Cases`. Une `Case` contient un pointeur vers un château s'il y en a un, et un pointeur vers un `Personnage` pour les autres habitants (vous pouvez fusionner ces 2 pointeurs en un seul si vous le souhaitez). Un `Personnage` possède sa couleur, son type (optionnellement, utiliser des `enum` pour ces deux champs), sa position, sa destination pour un agent ou le type de sa production en cours et le temps de production actuel pour un château, deux pointeurs vers les personnages suivant et précédent dans le château. À partir du niveau 2 (voir plus loin), il possède en plus deux pointeurs vers les personnages suivant et précédent symbolisant les personnages situés sur la même case ou la liste des châteaux d'une couleur (on appelle abusivement dans la suite cette liste la "liste des voisins"). Vous êtes autorisés à créer d'autres structures si vous le souhaitez.

4.2 Apparence

On affichera le plateau de jeu (monde quadrillé avec indications des éléments par des lettres par exemple) dans le terminal à chaque tour (ou après chaque action avec l'élément en attente d'action mis en avant). L'utilisateur doit pouvoir gérer ses actions au clavier, mais également quitter/sauver/etc. L'affichage en lui-même reste libre, il faut toutefois qu'il soit clair. L'utilisation d'une interface graphique est optionnel (voir plus loin).

4.3 Niveaux de notation

Ces niveaux de difficulté croissante correspondent à des notes maximales croissantes. Il faut qu'un niveau soit entièrement fonctionnel avant de passer au suivant.

Niveau 1

On gère uniquement les déplacements (destination, immobilisation, destruction) des agents et la production des châteaux. Deux éléments ne peuvent pas être sur une même case du plateau – un personnage créé par le château ira sur une case libre voisine. Une case du plateau de jeu pointe donc vers l'unique élément situé sur cette case. La liste des agents d'un château est doublement chaînée pour gérer plus facilement la destruction d'agents (voir Figure 1).

Niveau 2

On ajoute la possibilité pour les seigneurs de se transformer en un nouveau château. Il se retire de la liste chaînée des agents de son château d'origine. Le nouveau château est ajouté à la liste doublement chaînée des "voisins" des châteaux de sa couleur (voir Figure 2).

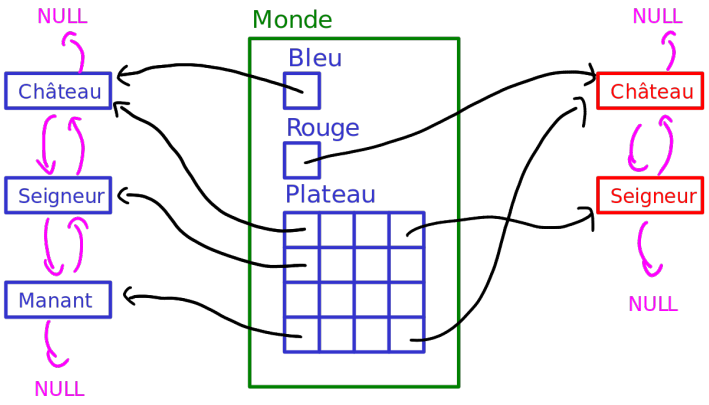

FIGURE 1 – Pour le niveau 1. Les pointeurs roses correspondent à la liste doublement chaînée des agents pour un château.

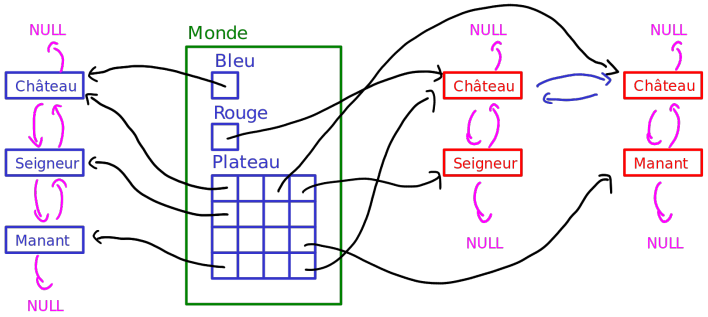

FIGURE 2 – Pour le niveau 2. Les pointeurs bleus correspondent à la liste doublement chaînée des "voisins" pour les châteaux.

Niveau 3

On ajoute la gestion des combats entre personnages de différentes allégeances.

Niveau 4

On autorise la possibilité d'avoir plusieurs agents sur une même case du plateau. On utilise pour cela la liste doublement chaînée des "voisins" des agents (voir Figure 3).

Améliorations

On peut envisager de nombreuses améliorations, prises en compte uniquement si les niveaux précédents sont fonctionnels.

Par exemple :

- ▷ L'utilisation d'une interface graphique (par exemple, SDL un tutoriel). On pourra utiliser un quadrillage pour le plateau de jeu, avec des carrés de différentes couleurs pour représenter les différents éléments. L'élément en

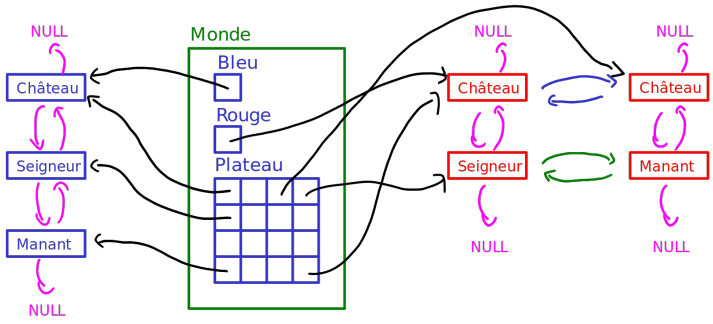

FIGURE 3 – Pour le niveau 4. Le manant du second château rouge se trouve sur la même case que le seigneur du premier château. Les pointeurs verts représentent la liste doublement chaînée des "voisins" de la case.

attente aura une couleur particulière et sera en attente d'un ordre donné par clic (destination, ou un bouton pour une production/destruction/...). Des boutons pour la gestion du type quitter/sauvegarder/... seront également sur la fenêtre.

- ▷ Des déplacements en patrouille (allers-retours entre un point et le château).
- ▷ Un système de points de vie et de points d'attaque, différent pour chaque personnage.
- ▷ Relier la survie et/ou la rapidité de production d'un château au nombre de manants qui lui sont rattachés.
- ▷ Une "intelligence artificielle" décidant de toutes les actions automatiquement.

5 Conditions de rendu

Le projet est à effectuer en **binôme**. En cas de nombre impair d'étudiants, **un seul** groupe sera autorisé à effectuer le projet en **trinôme** (notation plus sévère). Pour former les groupes, utiliser le canal Teams correspondant.

Étant données les circonstances, contacter les enseignants si impossibilité de le faire un groupe.

Le projet est à rendre avant la date et l'heure indiquées plus haut sur l'espace MyCourse dédié. **Chaque heure de retard sera pénalisée d'un point sur la note finale** (une heure entamée étant due). Une fois votre fichier envoyé, vous devez avoir un écran de confirmation avec votre fichier et la date de l'envoi. Le format de rendu est une archive au format ZIP contenant :

- ▷ Le code-source de votre projet (éventuellement organisé en sous-dossiers).
- ▷ Un fichier `README` à la racine, indiquant comment compiler et exécuter votre projet.
- ▷ Un document `dev.pdf`, devant justifier les choix effectués, les avantages et inconvénients de vos choix, expliquer les algorithmes et leur complexité, indiquer quelles ont été les difficultés rencontrées au cours du projet ainsi que la répartition du travail entre les membres du binôme. Un programmeur averti devra être capable de faire évoluer facilement votre code grâce

à sa lecture. En aucun cas on ne doit y trouver un copier/coller de votre code source. Ce rapport doit faire le point sur les fonctionnalités apportées, celles qui n'ont pas été faites (et expliquer pourquoi). Il ne doit pas paraphraser le code, mais doit rendre explicite ce que ne montre pas le code. Il doit montrer que le code produit a fait l'objet d'un travail réfléchi et minutieux (comment un bug a été résolu, comment la redondance dans le code a été évitée, comment telle difficulté technique a été contournée, quels ont étés les choix, les pistes examinées ou abandonnées...). Ce rapport est le témoin de vos qualités scientifiques mais aussi littéraires (style, grammaire, orthographe, présentation).

- ▷ Un ou plusieurs fichiers `.got` de tests.
- ▷ Optionnellement un makefile.

L'archive aura pour nom `Nom1Nom2.zip`, où Nom1 et Nom2 sont les noms des membres du binôme par ordre alphabétique. L'extraction de l'archive devra créer un dossier Nom1Nom2 contenant les éléments précisés ci-dessus.

Il va sans dire que les différents points suivants doivent être pris en compte :

- ▷ Projet compilant sans erreur ni warning avec l'option -Wall de gcc.
- ▷ On préfère un projet fonctionnant parfaitement avec peu de fonctionnalités qu'un projet qui a tenté de répondre à tout mais mal.
- ▷ Vérification des données entrées par l'utilisateur et gestion des erreurs.
- ▷ Uniformité de la langue utilisée dans le code (anglais conseillé) et des conventions de nommage et de code.
- ▷ Les sources doivent être commentées, dans une unique langue, de manière pertinente (pas de commentaire "fait un test" avant un if.).
- ▷ Le code doit être propre et correctement indenté.
- ▷ Bonne gestion de la libération de la mémoire (utilisez valgrind pour vérifier que le nombre de `free` est égal au nombre de `malloc` : il faut d'abord compiler avec l'option `-g`, puis lancer valgrind avec en argument le nom du programme. Par exemple :

  ```
  gcc -g main.c
  valgrind ./a.out
  ```

  Valgrind peut également vous indiquer la ligne où un segmentation fault a eu lieu, pratique !)
- ▷ Le projet doit évidemment être propre à chaque binôme. **Un détecteur automatique de plagiat sera utilisé**. Si du texte ou une petite portion de code a été emprunté (sur internet, chez un autre binôme,...), il faudra l'indiquer dans le rapport, ce qui n'empêchera pas l'application éventuelle d'une pénalité. Tout manque de sincérité sera lourdement sanctionné (conseil de discipline) – c'est déjà arrivé.

La documentation (rapports, commentaires...) compte pour un quart de la note finale.

5.1 Soutenance

Une soutenance d'une dizaine de minutes aura lieu pour chaque binôme, 2 binômes par 2 binômes dans l'ordre (2 jurys), à la date et à l'heure indiquées

plus haut. Elle doit être préparée et menée par le binôme (*i.e.* fonctionnant parfaitement du premier coup, avoir préparé des jeux de tests intéressants, etc.).

Pendant la soutenance, ne perdez pas de temps à nous expliquer le sujet : nous le connaissons puisque nous l'avons écrit. Essayez de montrer ce qui fonctionne et de nous convaincre que vous avez fait du bon travail.