BARBAULT Antoine

RAPPORT PROJET DOMINION

La version de Dominion présentée se joue entièrement sur un seul terminal Linux (voir annexe), grâce à des messages colorés qui guident les joueurs et un panel de commandes possibles pour le joueur assez flexibles (entrer le nom de la carte ou son id, avec insensibilité à la case, ou 'FIN' pour ne rien prendre d'autre). Le débogage est possible grâce à 'GODMODE'.

Le jeu se lance à l'aide de "make run", et il sera ensuite demandé de taper le nombre de joueurs présents. Il n'y a actuellement pas d'IA joueurs. Les dix cartes royaumes de base sont automatiquement choisies, et le tour du premier joueur commence. Les phases action ou achat qui ne peuvent pas être jouées si par exemple il n'y a pas de carte Royaume dans la main seront sautés, de la même manière que dans le jeu en ligne. Les messages du terminal permettent à tout moment de savoir quel joueur joue et quelle est la phase actuelle.

Le jeu est composé de fichiers .cpp et .h, séparés dans des dossiers, pour définir les classes et leurs membres représentés dans le diagramme UML en annexe. Chaque classe de carte et de phase crée une unique instance dont le programme manipule les pointeurs. Cela permet de déplacer des pointeurs plutôt que des instances, afin de favoriser la simplicité et l'économie de ressource.

Au début du développement, la partie IHM et la partie non IHM ont été séparées en accord avec le modèle MVC (Modèle-Vue-Contrôleur) afin de faciliter le potentiel développement d'une interface graphique pour le jeu. Ainsi les méthodes affichant du texte, directement ou non, sur le terminal sont situées en dessous d'une démarcation 'IHM' en commentaire, et les autres méthodes sont situées au-dessus. Les méthodes "IHM" peuvent appeler les méthodes non IHM mais pas l'inverse. Mais malheureusement cela n'a pas pu être tenu jusqu'au bout.

Un message apparaît à chaque pioche d'une carte avec le nom de la carte, même à la phase d'ajustement, pour signifier que pour un potentiel futur UI on voit les cartes être piochées. Chaque message est associé à un élément visuel de cet UI.

A l'origine, Lorraine et moi s'étions mis d'accord pour travailler ensemble sur le projet. Malheureusement, malgré les rappels de la part de Lorraine j'ai attendu le dernier moment pour y travailler. Nous avons réglé ce conflit en parlant tout les deux à Joël Gay, et nous avons convenu de faire la soutenance individuellement. J'ai donc hérité du travail de Lorraine, avec la tâche de tout finir en 4 jours.

Lorraine a écrit les classes de phase, les classes des cartes victoire et trésor, ainsi que les classes Jeu et Joueur.

Je me suis ensuite occupé d'implémenter les 10 cartes Royaume de base, des classes héritant de la classe 'Royaume', en suivant le modèle de Lorraine. J'ai également écrit la vérification des conditions de victoire après chaque tour et le 'godmode', pour avoir toutes

les cartes dans la main et 100 actions et achats possibles et 100 valeurs d'achat à chaque tour.

Au début de mon travail, j'ai rencontré une difficulté pour tester le jeu. Le programme écrit par Lorraine, qui permettait déjà de jouer à une version primaire de Dominion sur le terminal de son ordinateur Linux, ne fonctionnait pas sur le terminal WSL de mon ordinateur Windows. Le programme affichait les messages de début du jeu, qui clignotaient chez Lorraine mais pas chez moi, puis une segmentation fault faisait crasher le programme. En voyant cette différence j'ai installé toute une machine virtuelle Linux sur mon ordinateur, pour au final obtenir la même erreur, avec cependant les messages qui clignotent. ChatGPT m'a ensuite proposé d'utiliser AddressSanitizer pour tracer le problème jusqu'à la racine. En utilisant des std::cout, il a pu trouver ce qui causait l'erreur. Dans Phase.cpp, le membre statique :

```
Phase* Phase::phaseCourante = PhaseAction::getInstancePhaseAction();
```

appelle lors de son initialisation la fonction membre de PhaseAction :

```
PhaseAction* PhaseAction::getInstancePhaseAction() {
      return instancePhaseAction;
}
```

lui même renvoyant la valeur du membre statique de PhaseAction définie ainsi :

```
PhaseAction* PhaseAction::instancePhaseAction = new PhaseAction();
```

Si PhaseAction::instancePhaseAction est définie après Phase::phaseCourante, alors ce dernier n'aura pas de mémoire allouée. J'ai résolu le problème en initialisant la première variable statique à nullptr, et en lui donnant sa vrai valeur lorsque la deuxième variable est initialisée. J'ai ensuite rencontré l'énigmatique erreur heap-use-after-free, qui disparaît lorsque l'on enlève AddressSanitizer.

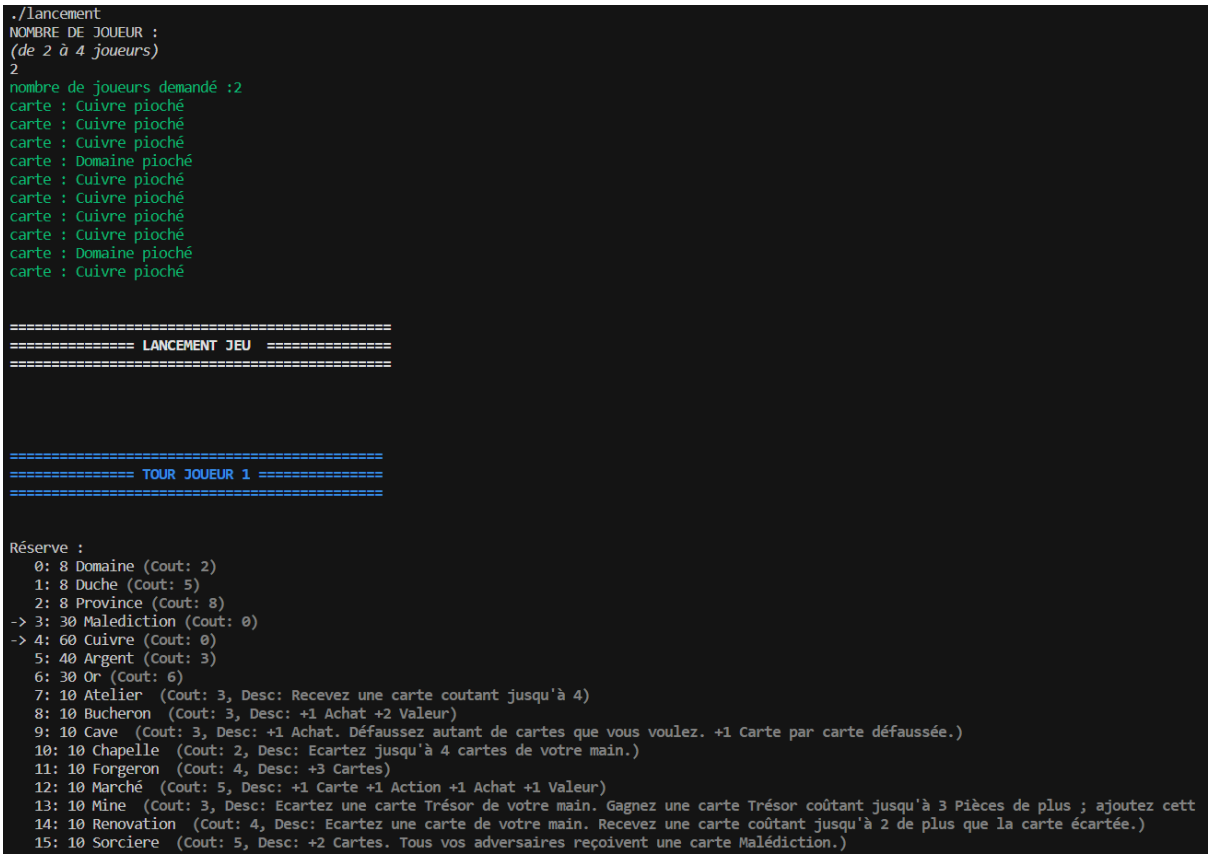
Après la séparation du binôme, j'ai modifié entièrement les classes phases, ce qui m'a pris plus de temps que prévu, m'empêchant de commencer des fonctionnalités bonus du projet ou même de terminer complètement les actions de base. Je voulais mettre la fonction jouer achat et jouer action dans les phases achat et action, plus précisément dans la fonction 'jouerPhase' héritée de 'Phase'. Je voulais également que le cycle de phase pour un joueur ressemble à ceci :

```
while(1){
    m_phaseActuelle->jouerPhase(jeu, *this);
    if(m_phaseActuelle->dernierePhase()) {
       jeu.commandePartieEstFinie();
       break;
```

```
    }
    m_phaseActuelle = &m_phaseActuelle->getPhaseSuivante();
}
```

J’ai également pris du temps pour afficher les cartes avec des flèches indiquant les cartes jouables ou acquérables, et la possibilité d’écrire un numéro plutôt que le nom de la carte (le numéro change en fonction de sa place dans la liste et de si la réserve ou la main est incluse dans les cartes jouables ou acquérables). J’aurais sûrement dû simplement utiliser la version qui fonctionnait avec toutes les cartes actions, quitte à avoir le même affichage que Lorraine, pour me concentrer sur les fonctionnalités bonus.

Annexe



```
carte : Domaine pioché
carte : Atelier pioché


==========================================
============== TOUR JOUEUR 1 ==============
==========================================


Réserve :
   8 Domaine (Cout: 2)
   8 Duche (Cout: 5)
   8 Province (Cout: 8)
   30 Malediction (Cout: 0)
   60 Cuivre (Cout: 0)
   40 Argent (Cout: 3)
   30 Or (Cout: 6)
   9 Atelier  (Cout: 3, Desc: Recevez une carte coutant jusqu'à 4)
   10 Bucheron  (Cout: 3, Desc: +1 Achat +2 Valeur)
   10 Cave  (Cout: 3, Desc: +1 Achat. Défaussez autant de cartes que vous voulez. +1 Carte par carte défaussée.)
   10 Chapelle  (Cout: 2, Desc: Ecartez jusqu'à 4 cartes de votre main.)
   10 Forgeron  (Cout: 4, Desc: +3 Cartes)
   10 Marché  (Cout: 5, Desc: +1 Carte +1 Action +1 Achat +1 Valeur)
   10 Mine  (Cout: 3, Desc: Ecartez une carte Trésor de votre main. Gagnez une carte Trésor coûtant jusqu'à 3 Pièces de plus ; ajoutez cette carte à votre main.)
   9 Renovation  (Cout: 4, Desc: Ecartez une carte de votre main. Recevez une carte coûtant jusqu'à 2 de plus que la carte écartée.)
   10 Sorciere  (Cout: 5, Desc: +2 Cartes. Tous vos adversaires reçoivent une carte Malédiction.)
   10 Village  (Cout: 3, Desc: +1 Carte  +2 Actions)
Cartes en cours d'utilisation:
PHASE ACTION du joueur 1
1 Action | 1 Achats
Main:
   0: 2 Cuivre (Cout: 0)
   1: 1 Domaine (Cout: 2)
-> 2: 2 Renovation  (Cout: 4, Desc: Ecartez une carte de votre main. Recevez une carte coûtant jusqu'à 2 de plus que la carte écartée.)
ECRIRE NOM CARTE OU ID
2
Réserve :
   8 Domaine (Cout: 2)
   8 Duche (Cout: 5)
   8 Province (Cout: 8)
   30 Malediction (Cout: 0)
   60 Cuivre (Cout: 0)
   40 Argent (Cout: 3)
   30 Or (Cout: 6)
   9 Atelier  (Cout: 3, Desc: Recevez une carte coutant jusqu'à 4)
   10 Bucheron  (Cout: 3, Desc: +1 Achat +2 Valeur)
```

Le diagramme de classes n'est pas définitif : possible changements futurs
(future petite coquille trouvée et/ou ajout de fonctionnalités (ex : nouvelle carte))

Antoine Barbault Lorraine Grandjean