TP7 Classes – SMP

Sommaire

[1. Introduction 1](#_Toc189409561)

[2. Fonctions 2](#_Toc189409562)

[2.1 Makefile et GitHub 2](#_Toc189409563)

[2.2 Création d’un point dans l’espace 2](#_Toc189409564)

[2.3 Surcharge d’opérateurs pour la manipulation des données 4](#_Toc189409565)

[2.4 Création d’une forme géométrique abstraite 4](#_Toc189409566)

[3. Conclusion 6](#_Toc189409567)

Introduction

Dans ce TP, nous allons nous intéresser au principe de classes en C++, et allons manipuler ces dernières en appliquant certaines notions vues en cours (notion de fonctions amies, héritage…). L’objectif est également de travailler en collaboration, en créant et manipulant un projet à l’aide de la plateforme GitHub. L’objectif de ce TP est donc double, puisqu’il nous demandera d’appliquer des notions sur le long terme, qu’il nous faudra ensuite réutiliser dans la suite de notre formation pour différents projets.

Nous commencerons donc par présenter les différentes fonctions réalisées ainsi que les fiches de tests associées attestant du bon fonctionnement du programme. Puis, nous apporterons une conclusion sur notre projet, des difficultés rencontrées ainsi que des compétences mises en œuvre.

Fonctions

2.1 Makefile et GitHub

Le fichier Makefile permet de générer les fichiers objet (.o) en associant le programme en .cpp et le header en .h. Le compilateur crée pour chaque programme un fichier source avant de lier les programmes entre eux (via l’appel des bibliothèques en début de programme).

La réalisation du Makefile peut prendre un certain temps, mais il est rare de revenir dessus pour y apporter des modifications.

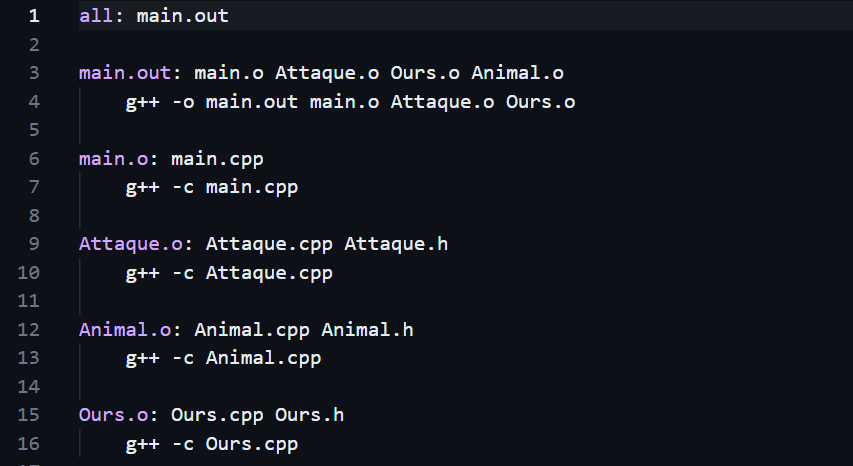
**Voici à quoi ressemble notre fichier Makefile :

Figure 1: Makefile

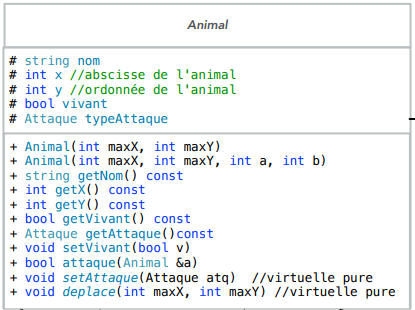
Une fois le Makefile créé, on peut faire appel à plusieurs commandes afin de compiler et/ou de lancer notre programme. Ces commandes sont directement utilisées dans le terminal de commande de VsCode.

Nous n’avons utilisé que la commande « make » pour compiler les programmes (création des fichiers objet) ainsi que la commande « ./main.out » pour lancer le projet.

Dans un second temps, nous avons utilisé la plateforme GitHub pour la bonne réalisation de notre projet. Nous avons directement programmé dans l’espace de codage du site. En effet, nous avons eu beaucoup de difficultés à initialiser notre projet directement à partir du PC de la salle, ce qui nous a fait perdre du temps sur la réalisation des fonctions.

2.2 Création d’une classe Animal

On crée dans un premier temps une classe Animal, qui constitue la principale étape de notre projet. Cette classe permet de gérer l’ensemble des éléments sur le plateau, en y attribuant divers paramètres (nom, type d’attaque…). A compléter

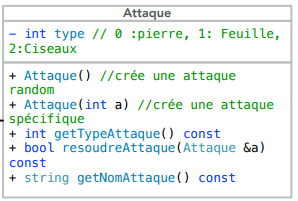
 En paramètres protégés, on veut intégrer les coordonnées de l’élément sur le plateau (abscisse et ordonnée), ainsi que le nom, l’état (vivant ou mort) et le paramètre d’attaque de l’animal.

On ajoute en tant que constructeur la fonction Animal, qui permet de générer un animal au hasard ou sur un endroit précis du plateau. On crée également des accesseurs pour récupérer le nom, les coordonnées ainsi que l’état de l’animal (pour le faire disparaître du plateau).

On crée enfin des mutateurs afin de modifier l’état de l’animal, et on ajoute des fonctions virtuelles (pas utilisées dans le projet, du moins sous cette forme) de déplacement et de gestion d’attaque. Une fois cette classe initialisée, on peut ajouter les autres éléments du projet, notamment la classe Attaque.

2.3 Création de la classe attaque

La classe Attaque est directement liée à la classe Animal et est utilisée par cette dernière pour déterminer le type d’attaque ainsi que le résultat d’une interaction entre deux animaux.

On intègre un paramètre privé qui correspond au type d’attaque (pierre, feuille ou ciseaux). On ajoute ensuite deux constructeurs afin de créer une attaque spécifique ou aléatoire.

Enfin, on crée des accesseurs, afin de récupérer le type d’attaque. On ajoute également une fonction permettant de résoudre une interaction entre deux animaux (ex : la pierre bat les ciseaux…) ainsi qu’un accesseur sur le nom de l’attaque.

Cette classe est utilisée par la classe Animal, et permet de gérer les interactions entre les éléments du plateau. On verra par la suite qu’elle est utilisée différemment en fonction du type d’animal en jeu.

2.4 Création des classes animales

Ici, on crée des classes différentes en fonction du type d’animal en jeu (pour ce TP, nous n’avons fait que les classes ours, loup et pierre). Ces classes héritent des paramètres de la classe Animal, et utilisent par conséquent la classe Attaque lors du jeu.

Basé sur l’énoncé de TP, on peut en dégager plusieurs spécificités :

- Les pierres ne se déplacent pas, et attaquent uniquement avec la pierre (attaque spécifique).

- Les ours ont 1/8 chance de se déplacer dans une direction aléatoire (voir énoncé de TP), et attaquent uniquement avec la feuille.

- Les loups se déplacent de manière aléatoire sur le plateau, et peuvent effectuer les trois types d’attaque.

Nous pouvons passer à présent aux différents jeux de test afin de valider le fonctionnement de notre projet.

2.5 Jeux de test et simulation

Fiche de tests :

|  |  |
| --- | --- |
| **1 - Fonctionnalité testée** | Génération du plateau |
| **2 - Scénario** |  |
| Donnée(s) d’entrée(s) | Aucune donnée d’entrée, on cherche uniquement à générer un plateau basique de dimension 10x10. |
| Situation(s) visée(s) | Pour l’instant, on veut s’assurer de la bonne initialisation du plateau de jeu. On lance le programme et on affiche à la console la grille, représentée par un ensemble de cases. |
| **3 - Résultat attendu** | La console est censée nous renvoyer un tableau 10x10 avec les cases délimitées. |
| **4 - Résultat obtenu** | On obtient bel et bien un tableau 10x10 avec des caractères représentant les cases du tableau. |
| **5 - Conclusion** | La génération du tableau s’effectue correctement. |

|  |  |
| --- | --- |
| **1 - Fonctionnalité testée** | Ajout d’éléments sur le plateau |
| **2 - Scénario** |  |
| Donnée(s) d’entrée(s) | On reprend le programme utilisé pour générer le tableau. |
| Situation(s) visée(s) | Cette fois-ci, on veut également y ajouter des animaux, une case à ¼ chance de faire apparaître un loup, un ours ou une pierre. |
| **3 - Résultat attendu** | On est censé obtenir le même tableau mais avec certaines cases indiquées avec un « p », un « L » ou un « o » en fonction de l’animal qui apparaît. |
| **4 - Résultat obtenu** | On obtient bien le même tableau avec le type d’animal qui est apparu. |
| **5 - Conclusion** | La génération d’éléments sur le plateau se fait correctement. |

|  |  |
| --- | --- |
| **1 - Fonctionnalité testée** | Interactions entre les éléments du plateau |
| **2 - Scénario** |  |
| Donnée(s) d’entrée(s) |  |
| Situation(s) visée(s) |  |
| **3 - Résultat attendu** |  |
| **4 - Résultat obtenu** |  |
| **5 - Conclusion** |  |

|  |  |
| --- | --- |
| **1 - Fonctionnalité testée** | Continuité du jeu sur plusieurs tours |
| **2 - Scénario** |  |
| Donnée(s) d’entrée(s) |  |
| Situation(s) visée(s) |  |
| **3 - Résultat attendu** |  |
| **4 - Résultat obtenu** |  |
| **5 - Conclusion** |  |

3. Conclusion

Nous avons à nouveau mis en application certains éléments du cours, en nous intéressant au fonctionnement d’une classe ainsi qu’à l’initialisation d’un projet sur GitHub. Nous avons mis moins de temps à initialiser le projet (voir la conclusion du TP précédent), ce qui nous a permis de nous concentrer sur le programme. Nous avons également mis en place un système de branches, afin de collaborer sans générer de conflits entre les versions.

Le principe fondamental d’une classe reste cependant pertinent, et nous avons la possibilité de revenir/d’avancer sur ce projet, et ce même après avoir rendu notre rapport.

Ce TP nous aura permis de nous familiariser un peu plus avec le langage C/C++, et nous a permis de mettre en application les principes vus en cours liés aux classes.