Labo 3 - River

POA

Walid Slimani et Timothée Van Hove

22 mai 2024

Table des matières

Introduction

Une famille composée d'un père, d'une mère, de deux filles et de deux garçons est accompagnée d'un policier et d'un voleur menotté. Ils doivent tous traverser une rivière à l'aide d'un bateau. Le but de ce laboratoire est de créer une application en C++ (console) permettant à l'utilisateur d'introduire des commandes pour embarquer et débarquer des personnes et déplacer le bateau, en satisfaisant des contraintes spécifiques.

Compilateur et options de compilation

Compilateur : `g++ version 12.2.0`

Version c++ : `23`

Build systems : `cmake 3.23.2` et `ninja 1.10.1`

Options de compilation : `-Wall -Wextra -Wpedantic -Wconversion -Wsign-conversion -Wvla -Werror`

Compiler et lancer le programme

Note: Ninja et CMake doivent être installés sur votre machine

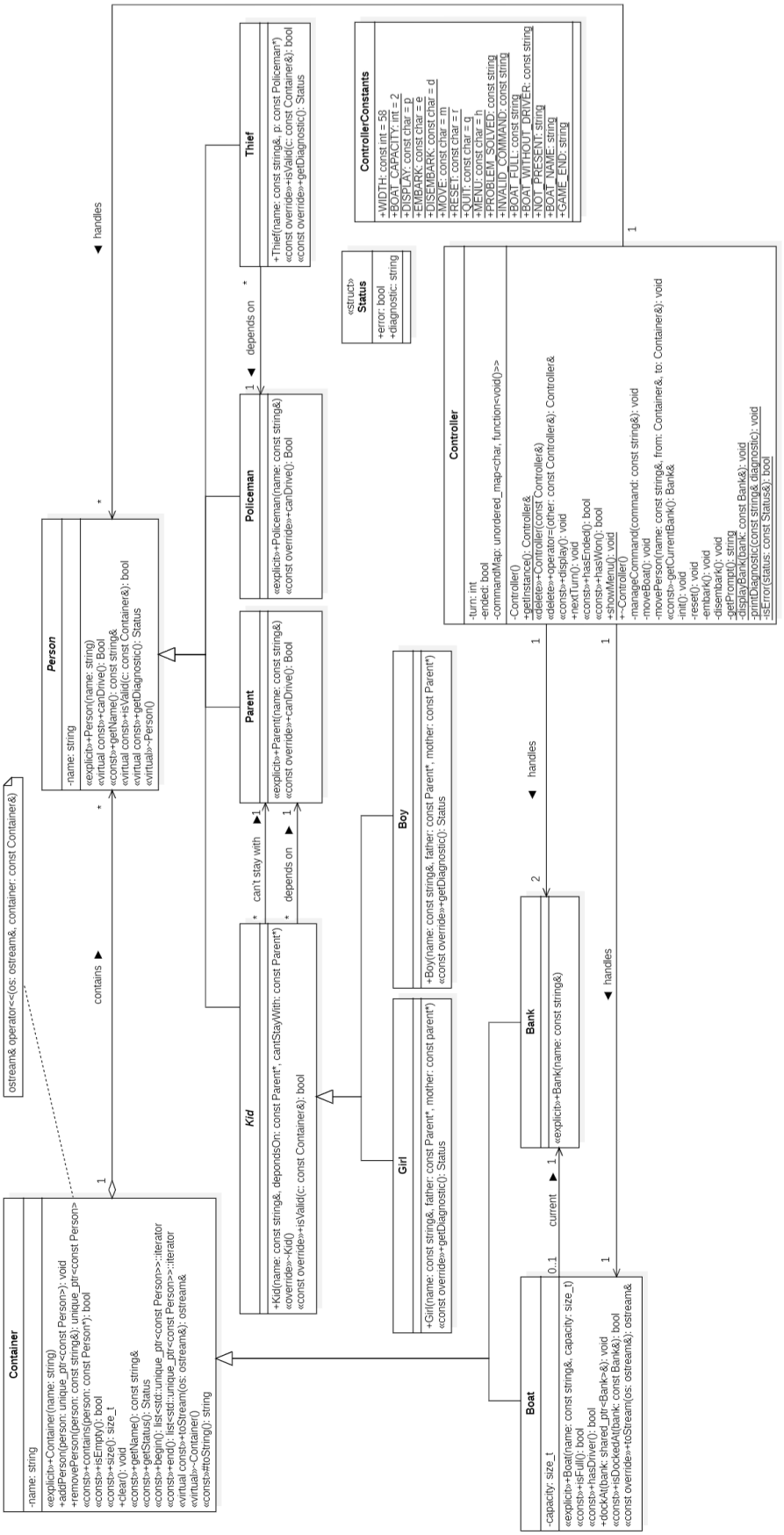
Windows

```
# Depuis la racine du projet
New-Item -Path build -ItemType Directory -Force; cd build; cmake .. -G "Ninja"; ninja; .\river.exe;
↪  cd ..
```

Linux

```
# Depuis la racine du projet
mkdir -p build && cd build && cmake .. -G "Ninja" && ninja && ./river && cd ..
```

Diagramme de classe



Choix d'implémentation

Singleton pour le Contrôleur

Le singleton est utilisé pour la classe `Controller`, assurant qu'une seule instance de ce contrôleur est créée durant l'exécution du programme. Ça garantit qu'il n'y a pas de conflits ou d'incohérences dues à la présence de multiples instances contrôlant le flux du jeu.

Hiérarchie de la classe Person

La classe `Person` sert de classe de base pour différents types de personnages dans le jeu, comme `Parent`, `Kid`, `Boy`, `Girl`, `Policeman`, et `Thief`. Cette hiérarchie permet de spécialiser le comportement de chaque type de personnage, notamment la capacité de conduire le bateau et les règles spécifiques de validation de leur présence dans les différents `Container`. Chaque sous-classe implémente les méthodes pour spécifier ces comportements. La hiérarchie Kid, Girl, Boy a été choisie pour simplifier l'implémentation. Kid gère les règles/contraintes, et Girl/Boy gèrent les messages d'erreur. Nous avons aussi décidé d'utiliser la structure `Status` pour retourner une paire erreur / message d'erreur.

Utilisation de smart pointers

Les smart pointers sont utilisés pour gérer la mémoire de manière sécurisée pour éviter les memory leak et pour simplifier la gestion du cycle de vie des objets.

- **`std::unique_ptr` dans Controller et Container :** Chaque personne est encapsulée dans un `std::unique_ptr` pour garantir qu'il ne puisse appartenir qu'à un seul conteneur à la fois. Ça facilite le transfert de propriété des personnages entre les conteneurs sans risque de duplication ou de perte de référence.
- **`std::shared_ptr` pour les rives dans le Controller :** L'utilisation de `shared_ptr` pour les rives permet de partager la propriété des banques entre le contrôleur et le bateau, sans se soucier de leur destruction prématurée.
- **`std::weak_ptr` dans Boat :** Le `weak_ptr` est utilisé pour référencer la banque à laquelle le bateau est amarré sans en prendre possession. Ça empêche les cycles de références qui pourraient entraîner des fuites de mémoire si `shared_ptr` était utilisé des deux côtés de la relation.

Bien que l'utilisation de smart pointers permet une gestion efficace des transferts et garanti qu'une personne ne puisse être que dans un container à la fois, cela nous oblige à déplacer les personnes dans un container **avant** de vérifier le respect des règles. Cela nous contraint à faire un `rollback` dans le cas ou les règles sont enfreintes. Cela implique que l'ordre des personnes dans les containers source peut changer si une règle est enfreinte.

Aucune contrainte ne nous est imposée concernant l'ordre des personnes dans les `Containers`. En prenant en compte la sécurité, simplicité d'utilisation et la garantie des transferts sans duplication entre `Contrainers` qu'apportent les `unique_ptr`, nous pensons que cette approche est justifiée.

Gestion des commandes par des fonctions

L'implémentation de la gestion des commandes à travers un `unordered_map` qui associe des commande clavier à des fonctions lambda permet une gestion claire et modulaire des différentes actions du jeu. Ça facilite l'ajout, la modification ou la suppression de commandes, tout en rendant le code plus lisible et plus facile à maintenir.

Tests

Une série de test a été effectuée pour vérifier le bon fonctionnement de notre programme.

Compiler et lancer les tests unitaires

Windows

```
# Depuis la racine du projet
New-Item -Path build -ItemType Directory -Force; cd build; cmake .. -G "Ninja"; ninja; .\tests.exe;
↪  cd ..
```

Linux

```
# Depuis la racine du projet
mkdir -p build && cd build && cmake .. -G "Ninja" && ninja && ./tests && cd ..
```

Tests unitaires: Règles

#	Description	Resultat
1	Pas plus que 2 personnes sont autorisées sur le bateau	ok
2	Le voleur et les enfants ne peuvent pas piloter le bateau	ok
3	Le voleur ne peut être laissé uniquement seul ou en présence du policier	ok
4	Les garçons peuvent être laissés seuls, avec leur frère/soeurs ou avec leurs parents, mais pas avec la mère si le père n'est pas présent.	ok
5	Les filles peuvent être laissées seules, avec leur frères/soeurs ou avec les deux parents, mais pas avec le père si la mère n'est pas présente.	ok
6	Les parents peuvent être seuls, ensemble, ou avec le policier	ok

Tests unitaires : Controller

#	Description	Resultat
1	Une personnes peut être ajoutée sans erreur dans un container	ok
2	Déplacer une personne d'un container vers un autre	ok
3	Après un transfert, la personne correct se trouve dans le container de destination	ok
4	Ajouter un `nullptr` dans un Container n'a pas d'effet	ok
5	L'intégrité du container est préservée lors d'un retrait d'une personne non-existante.	ok
6	Stress test: peut ajouter et supprimer un grand nombre de personnes	ok

Tests manuels

#	Description	Observation	ok?
1	Changer la rive du bateau vide	Erreur: "Le bateau a besoin d'un chauffeur pour bouger"	ok
2	Changer la rive du bateau avec un pilote dedans	Le bateau change de rive	ok
3	Changer de rive avec uniquement enfant ou le voleur dedans	Erreur: "Le bateau a besoin d'un chauffeur pour bouger"	ok
4	Résoudre le problème (tous sur la rive droite)	Message : "Problème resolu !"	ok
5	Essayer toutes les commandes p, e, d, m, r, q, h	Commandes ont l'effet escompté	ok
6	Essayer des commandes aléatoires (mots/ entiers / doubles)	Erreur: "### Commande invalide !"	ok
7	Essayer de charger/décharger une personne qui n'existe pas	Erreur: "### n'est pas present(e) sur "	ok

Conclusion

Ce projet a été l'opportunité d'appliquer des concepts POO en C++, en mettant l'accent sur une gestion efficace de la mémoire et une architecture logicielle cohérente. L'implémentation des personnes et des containers à travers une hiérarchie de classes bien pensée permet une extension facile et une maintenance du code. L'utilisation de smart pointers aide à prévenir les fuites de mémoire et assure le transfert des personnes de manière simple et cohérente.

Les tests manuels unitaires jouent un rôle important dans la vérification de la logique métier et des règles du jeu, en s'assurant que le système est fiable. Les résultats de ces derniers renforcent la confiance dans la qualité de notre programme.