Forestier Quentin & Herzig Melvyn

POO2 – 02.05.2021

Rivière

Labo3



Table des matières

[Introduction 2](#_Toc70870266)

[Implémentation 2](#_Toc70870267)

[Personnes 2](#_Toc70870268)

[IndependentPerson 3](#_Toc70870269)

[DependentPerson 3](#_Toc70870270)

[Considération sur les personnes 4](#_Toc70870271)

[Conteneurs 5](#_Toc70870272)

[Bank 5](#_Toc70870273)

[Boat 5](#_Toc70870274)

[Considération sur les conteneurs 5](#_Toc70870275)

[Contrôleur 6](#_Toc70870276)

[Entrées 6](#_Toc70870277)

[Tests 7](#_Toc70870278)

[Tests intégrations 7](#_Toc70870279)

[Tests utilisateurs 8](#_Toc70870280)

[Conclusion 11](#_Toc70870281)

[Annexes 11](#_Toc70870282)

# Introduction

Ce laboratoire porte sur l’implémentation du jeu de la rivière[[1]](#footnote-1) en mode console.

Plusieurs variantes sont possibles. La nôtre contient les personnages suivants :

* Le policier.
* La mère.
* Le père.
* Le voleur.
* Les deux garçons (paul et pierre).
* Les deux filles (julie et jeanne).

Les 8 protagonistes sont sur la rive gauche d’une rivière. Ils doivent tous se retrouver du côté droit en utilisant un bateau à 2 places.

Les situations suivantes ne doivent jamais avoir lieu ni dans le bateau ni sur les rives :

* Le voleur est avec au moins l’un des membres de la famille et sans le policier.
* Un des garçons est avec sa mère alors que son père n’est pas présent.
* Une des filles est avec son père alors que sa mère n’est pas présente.

Les enfants ainsi que le voleur ne peuvent pas piloter le bateau.

# Implémentation

## Personnes

La classe *Person* sert de base à toutes les personnes. Elle fournit le nom de la personne ainsi que trois méthodes abstraites :

*// La personne peut-elle conduire le bateau ?*

bool canDrive() const

*// La personne peut-elle rester avec celles en paramètres ?*

bool checkState(const list<const Person\*>& persons) const

*// Explication des relations de la personne.*

string getErrorMessage() const

Nous avons remarqué deux catégories de personnes :

* Les personnes indépendantes (*IndependentPerson*) qui peuvent conduire et qui n’ont pas de problèmes avec les autres. Cela inclut le policier, la mère et le père.
* Les personnes dépendantes (*DependentPerson*) qui ne peuvent pas rester avec n’importe qui sans leur « *tutor* ». Cela inclut les enfants et le voleur.

### IndependentPerson

Cette classe est une spécialisation de *Person*

Voici le comportement des trois méthodes héritées :

bool canDrive() const :

retourne true.

bool checkState(const list<const Person\*>& persons) const :

retourne true car n’a aucune relation problématique.

string getErrorMessage() const :

retourne un string vide. La personne n’a pas d’explications, car pas de relations problématiques.

Les personnages concernés héritent directement de cette classe

### DependentPerson

La classe *DependentPerson* offre une méthode pour ajouter un tueur et des personnes problématiques. Sans la présence du tuteur, la personne en question ne peut pas rester seule avec les personnes problématiques.

void setRule(const Person\* tutor, initializer\_list<const Person \*> persons)

Cette opération pourrait être faite via un constructeur. Toutefois, pour laisser plus de liberté dans l’ordre de création des personnages, nous avons décidé que la définition des « relations » serait faite par une méthode.

Cette classe implémente :

bool canDrive() const :

retourne false.

bool checkState(const list<const Person\*>& persons) const :

Vérifie pour chaque personne en paramètre si elle correspond à une personne passée dans la méthode *setRule*. Si c’est le cas, retourne false.

Nous avons ensuite défini deux classes qui permettent de spécifier le message d’erreur.

**Child :**

À la construction, définit si l’enfant est un garçon ou une fille à l’aide d’un booléen.

string getErrorMessage() const

Si c’est un garçon :  
Retourne : @nom ne peut pas rester avec sa mère sans son père.

Si c’est une fille :  
Retourne : @nom ne peut pas rester avec son père sans sa mère.

**Thief**

string getErrorMessage() const

Le voleur ne peut pas rester avec la famille sans le policier.

### Considération sur les personnes

**Avantages**Cette implémentation est très simple. Les personnes sont soit des personnes indépendantes, sans relation problématique et qui peuvent conduire, soit des personnes dépendantes, dans l’incapacité de conduire, avec des problèmes de relation et un message d’erreur personnalisé.

**Inconvénient**Au contraire, il est difficile de faire évoluer les règles. Comment ferions-nous pour avoir des personnes qui peuvent piloter tout en ayant des relations problématiques ? Une solution consisterait à considérer canDrive et checkState comme des comportements spécialisés de Person. Chaque personnage hériterait des comportements qu’il souhaite implémenter.

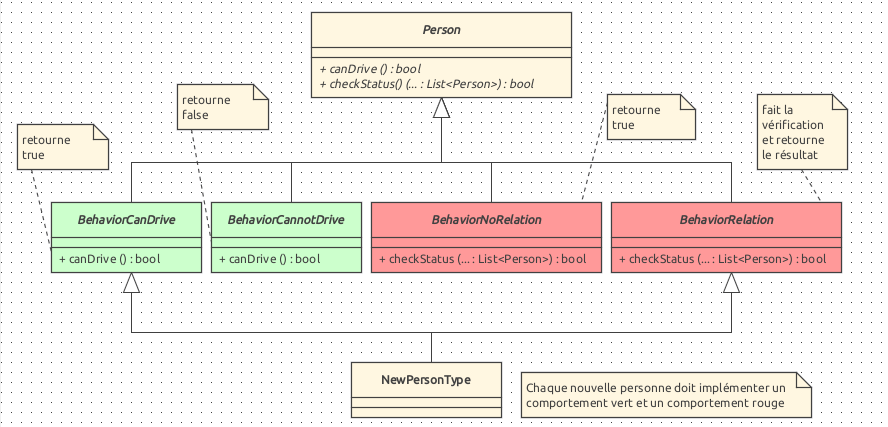


Figure Exemple partiel de l'implémentation des comportements

À l’aide d’héritage virtuel entre *Person* et les *Behavior*, nous serions libres de pouvoir créer de nouveaux types de personnes. Cependant cette technique augmente sensiblement le nombre de classes et de spécialisations.

Comme l’évolutivité des « règles » n’était pas un prérequis et que nous ne l’avons pas jugé nécessaire, nous avons implémenté la version simple des personnages.

**Autres remarques**L’implémentation par défaut, du constructeur de copie et de l’opérateur d’affectation, a été laissée. Elles sont suffisamment fonctionnelles et non utiles pour le laboratoire.

## Conteneurs

Les conteneurs modélisent les emplacements des personnes. Cela inclut, le bateau, la rive gauche et la rive droite.

Les conteneurs ont un nom et des personnes sous la forme.

std::list<const Person\*>

Les méthodes principales sont :

*// Ajouter et retirer des personnes, ignore les doubles et les nullptr.*

void addPersons(const std::list<const Person\*>& persons)

void addPerson (const Person\* person);

void removePerson (const Person\* person);

*// Permet de rechercher une personne par son nom*

const Person\* findByName(const std::string& name) const

*// Vérifie si les personnes sont compatibles entre elles, Si elle existe,*

*// retourne la première personne qui n’accepte pas la situation.*

const Person\* isValid() const

*// Définit la manière d’afficher le container dans un flux.*

virtual std::ostream& toStream (std::ostream& os)

### Bank

Cette classe hérite de *Container et* propose sa propre implémentation de toStream

### Boat

Cette classe spécialise *Container* en limitant le nombre de personnes qu’elle peut contenir.

Un bateau navigue entre deux rives :

LEFT\_BANK et RIGHT\_BANK

Il offre de nouvelles méthodes :

*//À tout moment, il est possible de savoir sur quelle rive se situe le bateau.*

const Bank\* getCurrentBank () const;

*//Il est possible de le faire changer de rive :*

void navigate();

*//Vérifie parmi les occupants, si une personne est capable de piloter :*

bool isDriverOnBoard()const

Finalement elle propose sa propre implémentation de toStream.

### Considération sur les conteneurs

Les personnes sont stockées sous la forme de pointeurs constants. Cette manière de faire est légère et permet de retourner les personnes sans risque de les modifier. Deux personnes sont considérées comme égales si elles ont la même adresse mémoire. Dans une version future, il faudrait considérer l’utilisation de *shared\_ptr* car rien ne nous garantit que les personnes soient encore allouées lorsqu’elles sont utilisées.

Dans le cadre de l’application, les personnes utilisées sont créées et disparaissent en même temps que les conteneurs. Le risque est donc négligeable. Toutefois pour une utilisation plus générique, cela serait une meilleure pratique.

L’implémentation par défaut, du constructeur de copie et de l’opérateur d’affectation, a été laissée.

## Contrôleur

Cette classe se charge du déroulement de la partie, notamment l’affichage, le traitement et l’exécution des entrées de l’utilisateur.

Cette classe ne peut être créée que par un constructeur sans paramètre qui initialise la partie. Entre autres, il crée les personnes, les rives et le bateau.

La partie peut être lancée au moyen de la méthode :

void play();

La classe alloue dynamiquement les personnes et se charge de les détruire lorsque l’objet cesse d’exister. Pour des raisons de sécurité, le constructeur de copie et l'opérateur d’affectation ont été retirés pour éviter que deux contrôleurs pointent sur les mêmes personnes. Il est envisageable de les rajouter lorsque les personnes seront manipulées au moyen de shared\_ptr.

### Entrées

Toutes les entrées sont sensibles à la casse. Taper « Paul » à la place de « paul » ou « R » au lieu de « r » ne fonctionnera pas.

# Tests

Les tests ont été divisés en deux catégories : tests d’intégration puis tests utilisateurs.

Les tests d’intégration peuvent être exécutés en passant « test » comme premier argument du main sinon le programme se lance normalement.

## Tests intégrations

Les tests ont été effectués avec des conteneurs non constants afin de pouvoir ajouter/retirer des personnes. Les personnes dépendantes étaient non constantes afin de pouvoir appeler la méthode setRule. Les personnes indépendantes étaient constantes. Ils ont pour principale vocation de tester l’intégralité des incompatibilités.

|  |
| --- |
| Description :  Si une personne dépendante se trouve avec une personne incompatible sans son tuteur, la méthode *isValid()* du conteneur retourne le pointeur de la personne qui est incompatible. |
| Exécution :  La méthode testWithoutTutor est appelée en passant chaque personne dépendante ainsi que les personnes incompatibles en argument. Elle tente de placer la personne dépendante avec chacune de ses incompatibilités sur les deux rives et sur le bateau. |
| Validation :  Le test est réussi si :   * testThiefWithFamilyWithoutCop * testBoysWithMomWithoutDad * testGirlsWithDadWithoutMom   affichent : « REUSSITE » |
| Résultat :  Validé |
| Méthodes impliquées :  Container ::addPerson, Container ::removePerson, Container ::isValid, DependantPerson ::setRule  DependantPerson ::checkState |

|  |
| --- |
| Description :  Si une personne dépendante se trouve avec une personne incompatible avec son tuteur, la méthode *isValid()* du conteneur retourne nullptr. |
| Exécution :  La méthode testWithTutor est appelée en passant chaque personne dépendante avec son tuteur et les personnes incompatibles en argument. Elle tente de placer la personne dépendante avec chacune de ses incompatibilités sur les deux rives. (Bateau impossible, car max 2 places) |
| Validation :  Le test est réussi si :   * testThiefWithFamilyWithCop * testBoysWithMomWithDad * testGirldWithDadWithMom   affichent : « REUSSITE » |
| Résultat :  Validé |
| Méthodes impliquées :  Container ::addPerson, Container ::removePerson, Container ::isValid, DependantPerson ::setRule  DependantPerson ::checkState |

|  |
| --- |
| Description :  Le bateau ne peut pas contenir plus de deux personnes |
| Exécution :  testBoatOverload essaie de placer 3 personnes à bord (père, mère et policier). La dernière tentative doit retourner false lors de l’appel à addPerson. Ensuite, il retire la deuxième personne, la troisième doit ensuite être ajoutée. |
| Validation :  Le test est réussi si :   * testOverloadingBoat   affichent : « REUSSITE » |
| Résultat :  Validé |
| Méthodes impliquées :  Container ::addPerson, Container ::removePerson |

## Tests utilisateurs

|  |  |
| --- | --- |
| Condition préalable : aucune | Résultat |
| Description :  Entrer ‘p’ affiche l’état des rives et du bateau | Validé |

|  |  |
| --- | --- |
| Condition préalable : aucune | Résultat |
| Description :  Entrer ‘e’ ou ‘d’, avec un nom inexistant, réaffiche l’état du jeu | Validé |

|  |  |
| --- | --- |
| Condition préalable : aucune | Résultat |
| Description :  Entrer ‘e’ ou ‘d’, avec respectivement une rive vide ou un bateau vide ne fait rien à part réafficher l’état précédent. | Validé |

|  |  |
| --- | --- |
| Condition préalable :  État initial, bateau rive gauche | Résultat |
| Description :  Entrer ‘e’ et un nom de personnage qui laisse la rive gauche dans un état incompatible (p.ex mere). Le personnage ne monte pas dans le bateau et une explication s’affiche. Incrément du tour | Validé |

|  |  |
| --- | --- |
| Condition préalable :  Bateau rive droite, pouvoir créer une incompatibilité sur la rive droite en embarquant. | Résultat |
| Description :  Entrer ‘e’ et un nom de personnage qui laisse la rive dans un état incompatible. Le personnage ne monte pas dans le bateau et une explication s’affiche. Incrément du tour. | Validé |

|  |  |
| --- | --- |
| Condition préalable :  Bateau rive gauche, pouvoir créer une situation de conflit dans le bateau en embarquant. P.ex. : faire entrer Julie alors que le voleur s’y trouve | Résultat |
| Description :  Entrer ‘e’ et un nom de personnage qui est incompatible avec la personne du bateau.  Le personnage reste sur la rive et un message d’explication est affiché. Incrément du tour. | Validé |

|  |  |
| --- | --- |
| Condition préalable :  Bateau rive droite, pouvoir créer une situation de conflit dans le bateau en embarquant. P.ex. : faire entrer Paul alors que sa mère s’y trouve. | Résultat |
| Description :  Entrer ‘e’ et un nom de personnage qui est incompatible avec la personne du bateau.  Le personnage reste sur la rive et un message d’explication est affiché. Incrément du tour. | Validé |

|  |  |
| --- | --- |
| Condition préalable :  Bateau rive gauche, pouvoir embarquer sans créer de conflit. P.ex. : e voleur depuis l’état initial. | Résultat |
| Description :  Entrer ‘e’ et un nom de personnage qui peut embarquer sans conflit.  Le personnage quitte la rive et entre dans le bateau. Incrément du tour. | Validé |

|  |  |
| --- | --- |
| Condition préalable :  Bateau rive droite, pouvoir embarquer sans créer de conflit. | Résultat |
| Description :  Entrer ‘e’ et un nom de personnage qui peut embarquer sans conflit.  Le personnage quitte la rive et entre dans le bateau. Incrément du tour. | Validé |

|  |  |
| --- | --- |
| Condition préalable :  Bateau rive gauche avoir une personne dans le bateau qui ne peut pas débarquer.  P.ex. :Jeanne et sa mère sont dans le bateau. Jeanne tente de débarquer avec son père sur la rive. | Résultat |
| Description :  Entrer ‘d’ et un nom de personnage qui ne peut pas débarquer sans créer de conflit. Il reste dans le bateau et un message d’erreur s’affiche. Incrément du tour. | Validé |

|  |  |
| --- | --- |
| Condition préalable :  Bateau rive droite avoir une personne dans le bateau qui ne peut pas débarquer.  P.ex. :voleur et policier sont dans le bateau. Voleur tente de débarquer avec membre de la famille sur la rive. | Résultat |
| Description :  Entrer ‘d’ et un nom de personnage qui ne peut pas débarquer sans créer de conflit. Il reste dans le bateau et un message d’erreur s’affiche. Incrément du tour | Validé |

|  |  |
| --- | --- |
| Condition préalable :  Bateau rive gauche avoir une personne dans le bateau qui peut débarquer. | Résultat |
| Description :  Entrer ‘d’ et un nom de personnage qui peut débarquer sans créer de conflit. Il passe sur la rive. Incrément du tour. | Validé |

|  |  |
| --- | --- |
| Condition préalable :  Bateau rive droite avoir une personne dans le bateau qui peut débarquer. | Résultat |
| Description :  Entrer ‘d’ et un nom de personnage qui peut débarquer sans créer de conflit. Il passe sur la rive. Incrément du tour. | Validé |

|  |  |
| --- | --- |
| Condition préalable :  Bateau sur la rive gauche, placer deux enfants dans le bateau ou le bateau vide | Résultat |
| Description :  Appuyer sur m ne déplace pas le bateau à droite. Incrément du tour | Validé |

|  |  |
| --- | --- |
| Condition préalable :  Bateau sur la rive droite, placer deux enfants dans le bateau ou le bateau vide | Résultat |
| Description :  Appuyer sur m ne déplace pas le bateau à gauche. Incrément du tour | Validé |

|  |  |
| --- | --- |
| Condition préalable :  Bateau sur la rive gauche, avoir un conducteur dans le bateau | Résultat |
| Description :  Appuyer sur ‘m’ déplace le bateau sur la rive droite. Incrément du tour. | Validé |

|  |  |
| --- | --- |
| Condition préalable :  Bateau sur la rive droite, avoir un conducteur dans le bateau | Résultat |
| Description :  Appuyer sur ‘m’ déplace le bateau sur la rive gauche. Incrément du tour. | Validé |

|  |  |
| --- | --- |
| Condition préalable :  Bateau sur la rive gauche, impossible de faire entrer trois personnes dans le bateau.  P.ex : Paul et pierre sont dans le bateau, le père ne peut pas y entrer aussi. | Résultat |
| Description :  Embarquer une troisième personne incrémente le tour, mais elle reste sur la rive. Rafraîchissement de l’état. | Validé |

|  |  |
| --- | --- |
| Condition préalable :  Bateau sur la rive droite, impossible de faire entrer trois personnes dans le bateau.  P.ex : Jeanne et Julie sont dans le bateau, paul ne peut pas y entrer aussi. | Résultat |
| Description :  Embarquer une troisième personne incrémente le tour, mais elle reste sur la rive. Rafraîchissement de l’état. | Validé |

|  |  |
| --- | --- |
| Condition préalable :  Bateau sur la rive droite, impossible de faire entrer trois personnes dans le bateau.  P.ex : Jeanne et Julie sont dans le bateau, paul ne peut pas y entrer aussi. | Résultat |
| Description :  Embarquer une troisième personne incrémente le tour, mais elle reste sur la rive. Rafraîchissement de l’état. | Validé |

|  |  |
| --- | --- |
| Condition préalable :  Avoir éparpillé les personnages pour voir l’effet, mais pas de réelle condition. | Résultat |
| Description :  Entrer ‘r’ réinitialise la partie. Le tour reprend à 0, les personnages et le bateau sont repositionnés. La partie peut être recommencée sans blocage. | Validé |

|  |  |
| --- | --- |
| Condition préalable :  Aucune | Résultat |
| Description :  Entrer ‘h’ affiche le menu suivi de l’état du jeu. | Validé |

|  |  |
| --- | --- |
| Condition préalable :  aucune | Résultat |
| Description :  Entrer ‘q’ termine l’application. | Validé |

|  |  |
| --- | --- |
| Condition préalable :  Terminer le jeu. | Résultat |
| Description :  Affiche un message de félicitation. | Validé |

# Conclusion

Au terme de ce laboratoire, nous sommes arrivés à implémenter une solution fonctionnelle du jeu de la rivière. Nous avons choisi la légèreté d’implémentation par rapport à l’évolutivité des règles.

Les tests se sont voulus le plus complets possible et nous pensons avoir fait le tour de ce qui était raisonnable à mettre en place pour garantir une application fonctionnelle.

# Annexes

* Code source (.h et .cpp)
* UML du projet

1. Inspiré de : https://en.wikipedia.org/wiki/River\_crossing\_puzzle [↑](#footnote-ref-1)