Documentation du Logiciel de Gestion d'un Zoo

Jean-Baptiste Lasserre & Chatelard Valentin

27 avril 2014

Table des matières

Introduction

Cette documentation permet à l'utilisateur de ce programme / code de comprendre les principes utilisés et les fonctionnements des fonctions utilisées. Ce programme a été réalisé sur *GitHub* et est à l'adresse est la suivante `https://github.com/BiarnGamer/pommeDePin`. Ce logiciel est publié sous la licence Apache v2, il est donc OpenSource (malgré que personne ne risquera de jamais s'en servir...).

1 Présentation des Approches

Notre programme oblige l'implémentation de classe afin de gérer tout l'ensemble. La gestion d'un zoo implique une classe *mère* Animal ainsi que ses 10 classes *filles* qui sont des animaux différents. Il nous faut aussi une classe Enclos afin de déposer ces animaux, et pour finir une classe Parc qui elle est au cœur du programme. Elle fait le lien entre les entrées de l'utilisateur et les classes Enclos et animaux. Elle se situe donc entre les fonctions de saisies de données et les classes simulant le zoo. C'est une classe ne faisant aucun affichage (sauf gestion erreurs) et ne demandant aucune informations à l'utilisateur. Ainsi les méthodes propres à Parc permettront de ne jamais communiquer directement entre les animaux et les enclos. Ceci nous assure une gestion dans les modifications et une main mise sur la sécurité du programme.
Les Classes et leurs méthodes sont définies dans les **.h**. Un diagramme de classe est aussi joint avec la documentation, celui ci étant trop grand pour rentrer sur une page.

2 Rôle des fonctions par classes

2.1 Animal

Les principales fonctions sont :
- Les constructeurs (par défaut, par passage de paramètre, et par recopie)
- Le destructeur
- Les getters publics
- Les setters, ceux ci ne sont pas tous publics car une fois l'ID donné il ne faut pas pouvoir en changer, de même pour le nom. Seule la classe Animal peut en avoir besoin à l'instanciation.
- La surchage des opérateurs = , == ,!= ainsi que l'affichage.

2.2 Classes filles de Animal

Les classes filles n'ont besoin que des fonctions qui sont propres à leur caractéristiques personnelles, celles ci sont :
- Leur constructeur personnel (par défaut, recopie et passage de paramètre)
- Le destructeur
- Les getters et setters pour leurs caractéristiques
- Leur surcharge aussi des opérateurs =, ==, != et aussi l'affichage.

2.3 Enclos

La classe Enclos est elle un peu particulière car elle possède un lien vers les animaux qui lui appartienne. Cependant les animaux sont indépendants de l'enclos, le lien entre les classes est une simple agrégation. Pour la représenter nous avons utilisé notre propre implémentation d'un set (ensemble au sens théorie des ensembles, sans possibilités de doublons car il n'est pas possible de mettre deux fois un même animal dans l'enclos) qui stocke ici des pointeurs sur Animal. Cette abstraction nous permet de ne plus avoir besoin de gérer les tableaux car Set est totalement indépendant. La gestion mémoire, la suppression ainsi que toutes les fonctions dont on peut avoir besoin sont implémenter dans celui ci. Cela nous permet de mieux se concentré la gestion du zoo.
La classe zoo possède elle :

- Les constructeurs (par défaut, recopie et passage de paramètre)
- Le destructeur
- Les getters et setters pour les caractéristiques, de même ici les setters ne sont pas tous publics comme par exemple setID car l'ID donné à l'instanciation ne doit pas pouvoir changer au cours de la vie de l'enclos.
- La surcharge des opérateurs =, ==, != et aussi l'affichage.
- Les fonctions essentielles à la gestion des animaux qui sont :
  - ajouter un animal
  - intervertir deux animaux (utiles pour les tris)
  - supprimer un animal

2.4 Parc

Le cœur du programme se situe dans celui ci, c'est par lui que tout va passer, il est donc nécessaire d'assurer une sécurité du programme et une gestion des plus simples et des plus efficaces en empêchant fuite mémoire, et possibilité de faire ce qu'on veut dans le programme.
Sa conception a été compliqué car nous avons fait face à des problèmes que nous n'avions jamais soulevé jusqu'ici. Nous pensions simplement utiliser un set de pointeurs sur Animal puis d'utiliser les méthodes des classes filles pour tout gérer. A ce moment la l'héritage nous aurait permis une gestion ultra simplifiée. Malgré cela, il est vrai qu'un objet de la classe mère n'a pas accès aux classes filles et donc nous nous sommes retrouvés dans une impasse. Afin de remédier à cela, nous avons rajouté un set par animal de la classe fille, celui ci nous permettant de géré leurs méthodes et appels.
Voici la description de certaines fonctions **essentielles** que nous avons implémenter afin de comprendre comment le logiciel est géré.

2.4.1 Création d'enclos

On reçoit en paramètre le nom, le type et la capacité de l'enclos à créer, on en instancie un et on le stocke dans le tableau approprié. Ainsi c'est bien la classe Parc qui gère la liste des enclos sans soucis. Il sera impossible d'y accéder en dehors de la classe. C'est une notion essentielle pour la sécurité du programme.

2.4.2 Création d'un animal

Il nous aura donc fallut ici implémenter une fonction différente pour créer un animal différent. Ceci est possible car le compilateur choisira la fonction qui correspond en fonction du type de paramètre qu'on lui donnera. La gestion est identique à celle de la création d'un enclos.

2.4.3 Modification d'Enclos et d'Animaux

On prend en paramètre l'ID de l'animal où de l'enclos à modifié, ainsi que l'animal tel que nous le désirons après modification. La recherche est effectuée, les modifications sont alors appliquées via les setters. Cette fonction permet de modifier sans demander directement à l'utilisateur ce que l'on doit modifier. Toujours dans le principe que Parc ne communique pas avec l'utilisateur, elle est la maitresse dans la gestion.

2.4.4 Supprimer un Enclos ou un Animal

Le principe est semblable à la modification, on prend un ID, on applique la recherche qui nous permet de le récupérer. Par la suite on le supprime. On part du principe que lorsque l'utilisateur demande la suppression, la fonction traitant la demande s'est occupée de changer les animaux d'enclos selon les envies de l'utilisateur. Comme il n'y a pas de communication, si des animaux sont toujours présent, ils seront alors relâchés (supprimé donc) et ensuite nous supprimerons l'enclos. Pour l'animal on supprimera directement l'animal en évitant toute fuite mémoire et en l'enlevant de son enclos.

2.4.5 Accès, Rechercher

Ce sont des fonctions essentielles mais simples qui permettent la gestion du parc.

2.4.6 Déplacer un animal

Ici les choses commencent à se compliquer. La fonction déplacer animal permet de choisir l'enclos de départ par son ID, l'enclos d'arrivé par son ID ainsi que l'animal à déplacer par son ID. Les tests nécessaire sont fait, mais cette fonction ne fait que le déplacement.

2.4.7 Relations proies prédateurs

Nous avons créé dans Parc un tableau de relation entre les proies et les prédateurs. Notre vision particulière nous a amener à modéliser les choses de la façon suivante. Nous avons une liste<set Proie>. Proie étant une structure contenant le code de la proie, ainsi que 2 seuils. Ces seuils représente des valeurs particulière.

C'est une fonction qui détermine si une espèce A mange une espèce B dans un enclos. Elle reçoit en paramètre les nombre de prédateurs de l'espèce A ainsi que le nombre de proie de l'espèce B (B est donc une proie de A). Il y a trois cas possibles :

Situation	Actions possibles
n < seuil1	Il les mange
	Il sympathise
seuil1 ≤ n ≤ seuil2	Il les mange
	Il sympathise
	Les proies se défendent et le tuent
seuil2 < n	Les proies se défendent et le tuent
	Il sympathise

Table 1 – Schematisation des seuils

1. Les animaux de l'espèce A mangent tous les animaux de l'espèce B
2. Les animaux de l'espèce A et ceux de l'espèce B cohabitent dans la paix (temporairement du moins)
3. Les animaux de l'espèce B se défendent et tuent ceux de l'espèce A

On détermine le cas de figure selon le nombre nPred de prédateurs et nProies de proies. Pour chaque prédateur de A, on défini le nombre maximum de proies de B qu'il peut manger s'il est seul (seuil1), ainsi que le nombre de proies de B qu'il faut pour se défendre contre un prédateur de A (seuil2). Ainsi, si un prédateur se retrouve avec peu de proies il les dévore (ex : un tigre contre deux marmottes) mais s'il est face à trop de proies il peut se faire tuer (ex : un titre contre 15 éléphants). On peut donc déterminer le cas dans lequel on est en utilisant les deux seuils définit :

Pour un prédateur contre n proies : S'il y a p prédateurs, tous les seuils sont multiplié par p.

À noter : Afin de laisser un peu de suspens, il y a toujours plusieurs actions possibles à chaque seuil (cf. le tableau ci-dessus). Ainsi, si on appelle deux fois la fonction avec les mêmes données, le résultat ne sera pas toujours le même. La distinction sera faite par une fonction aléatoire.

2.4.8 Consequence Deplacement Animal

Les déplacements n'étant pas toujours sans conséquence dans les enclos une autre fonction est nécessaire.
Afin de prévoir ce qu'il pourrait se passer nous avons générer une fonction qui retourne un paramètre particulier, un entier, qui a une signification précise que voici. Nous lui donnons en paramètre l'enclos ainsi que l'animal considéré.
Si la fonction retourne :

- -1 : L'enclos est invalide ou inexistant.
- 0 : Tout est ok pour le déplacement.
- 1 : L'enclos est plein.
- 2 : L'animal se noie.
- 3 : L'animal s'envole.
- 4 : L'animal possède des prédateurs dans l'enclos.
- 5 : L'animal possède des proies dans l'enclos.
- 6 : L'animal possède des proies et des prédateurs dans l'enclos.

Cette fonction n'applique rien, elle permet juste de signaler à l'utilisateur si le déplacement peut engendrer des conséquences.

2.4.9 Relations Proies Predateurs

Si l'utilisateur accepte les conséquences, c'est cette fonction qui va définir qui mange qui ou si une paix est établie.
Cette fonction **n'est pas déterministe**, nous avons voulu nous fier à la réalité et donc parfois des proies et des prédateurs se font peur mutuellement à cause de leur nombre et donc personne n'attaque. C'est pour cela que par exemple même si le tigre peut manger la marmotte, si il y en a 200 il ne va pas forcement attaquer (imaginez des marmottes qui tuent un tigre). Donc même lors d'une relation, nous assurons 75% de chance qu'il les mange, mais 25 autre que la paix soit établies. Donc pour un même déplacement il ne se produira pas toujours les mêmes conséquences. Le tableau 2 - Schematisation des seuils, explique bien cela.
C'est cette fonction qui va appelé la fonction AnimauxMangerOuTuesDansEnclos avec le paramètre retourné par relation proies prédateurs et donc supprimer les animaux tués (s'il y en a).

2.4.10 Tris divers

Quelques tris ont été implémenter afin de permettre une meilleure visualisation par l'utilisateur.
- Tri Animaux ou Enclos par Nom
- Tri par Espèce des Animaux
- Tri Animaux par Nom ou Espèce par Enclos
- Tri par Occupation des enclos
- Tri par Capacité des enclos
- Tri par Taux Occupation des Enclos

3 Menu du Programme et fonctionnalités

Différents Menus et Sous menus :

```
************************
***  Menu principal  ***
************************

1. Gestion des enclos et des animaux
2. Affichage des informations sur les enclos
3. Affichage des informations sur les animaux
4. Sauvegarder le parc
5. Charger un parc
9. Quitter
Choix :

****************************************
***  Gestion des enclos et des animaux  ***
```

```
*****************************************

1. Créer un enclos
2. Supprimer un enclos
3. Demander à Barney de capturer un animal
4. Déplacer un animal
5. Relâcher un animal
9. Retour
Choix :
```

3.1 Gestion des enclos et des animaux

3.1.1 Créer un enclos

Utilisation des fonctions pour l'utilisateur afin de récupérer les informations pour la création de l'enclos dans le parc. Puis message indiquant la bonne création de l'enclos.

3.1.2 Supprimer un enclos

Affichage des enclos possibles à supprimer. Choix par l'utilisateur de l'enclos. Si le choix de l'enclos est bon alors on demande confirmation, puis pour chaque animal présent dans l'enclos on propose de le mettre dans un autre enclos (compliqué car risque de se faire manger ou de manger des animaux mais en meme temps quelle idée de supprimer un enclos plein !).

3.1.3 Demander à Barney de capturer un animal

Choix de l'animal puis Utilisation des fonctions pour l'utilisateur afin de récupérer les informations pour la création de l'animal considéré. Puis message indiquant la bonne création de l'enclos.

3.1.4 Déplacer un animal

Proposition des enclos disponibles (non vide) pour récupérer l'animal à déplacer, ensuite on affiche les enclos disponibles (dont il reste des places) afin de déplacer l'animal dans l'enclos choisi (affichage des conséquences puis confirmation avant déplacement).

3.1.5 Relâcher un animal

Affichage des enclos non vide, puis choix de l'enclos, puis choix de l'animal puis suppression de celui ci.

```
************************************************
***  Affichage des informations sur les enclos  ***
************************************************

1. Afficher le taux de remplissage des enclos
2. Afficher le détail d'un enclos
3. Afficher le détail de tous les enclos
4. Afficher le détail et les animaux d'un enclos
```

```
5. Afficher le détail et les animaux de tous les enclos
9. Retour
Choix :
```

3.2 Affichage des informations sur les enclos

3.2.1 Afficher le taux de remplissage des enclos

Utilisation de afficherTauxRemplissageEnclos(Parc1)

3.2.2 Afficher le détail d'un enclos

Utilisation de rechercheEnclos(Parc1) car cette fonction demande l'ID de l'enclos dans cette fonction.

3.2.3 Afficher le détail de tous les enclos

Utilisation de afficherDetailDeTousLesEnclos(Parc1)

3.2.4 Afficher le détail et les animaux d'un enclos

Utilisation de afficherDetailEnclosEtAnimaux(Parc1) qui s'occupe des liens avec les utilisateurs.

3.2.5 Afficher le détail et les animaux de tous les enclos

Utilisation de afficherDetailEtAnimauxDeTousLesEnclos(Parc1)

```
**************************************************
***  Affichage des informations sur les animaux  ***
**************************************************

1. Afficher le détail d'un animal
2. Afficher le détail de tous les animaux d'un enclos
3. Afficher le détail de tous les animaux du parc
9. Retour
Choix :
```

3.3 Affichage des informations sur les animaux

3.3.1 Afficher le détail d'un animal

Utilisation de rechercheAnimal(Parc1) qui affiche l'animal recherché.

3.3.2 Afficher le détail de tous les animaux d'un enclos

Utilisation de afficherDetailEnclosEtAnimaux(Parc1).

3.3.3 Afficher le détail de tous les animaux du parc

Proposition de tri via triAnimaux(Parc1) puis affichage des animaux via afficheAnimauxParc(Parc1).

4 Gestion des erreurs

Nous avons décidé, afin de blinder le programme, de ne pas supprimer les conditions de tests avant les **try**. C'est à dire par exemple :

```
1 iRangEnclos = rechercheEnclos(Parc1);
2 if(iRangEnclos != -1) {
3         try {
4                 iIDEnclos = Parc1.getEnclos(iRangEnclos).
                      getID();
5         }
6         catch(string const& chaine){
7                 cerr << chaine << endl;
8         }
```

Il aurait été possible de supprimé le *if* car le try aurait affiché l'erreur. Or afin de blinder le programme, nous avons préféré garder ces conditions car nous jugeons qu'il ne doit pas être possible d'aller récupérer un enclos d'un rang invalide. La gestion d'erreur étant normalement utilisée pour des erreurs plus complexes comme par exemple les allocations dynamiques qui échouent.