**Rapport Projet Fil Rouge POOI**

**Parc Zombillenium**

**Réalisé par :**

-Thibaut Tucoulou

-Benoît Feurgard

**Gestion mise en place**

Afin de répondre au problème posé, nous avons mis en place un programme pouvant s’apparenter à un logiciel de gestion du personnel. Nous avons donc une administration principale composée de deux listes : une liste de personnel, et une d’attractions qui vont être l’objet de nos actions.

En ce qui concerne le squelette du logiciel, nous avons utilisé le diagramme de classes fourni ainsi que le fichier Listing.CSV pour le peuplement de base du parc Zombillenium. Nous avons ensuite créé de nombreuses méthodes permettant d’interagir avec les données comme l’ajout manuel de membres du personnel ou attractions, la modification de certains caractères, les tris de ceux-ci… Toutes ces méthodes se retrouvent dans la classe Administration de notre projet pour plus de lisibilité et permettent ainsi d’implémenter une gestion optimisée du parc Zombillenium.

**Démonstration**

La démonstration se compose d’un enchaînement de chacune des actions énoncées dans le sujet du projet. Chacune des étapes est suivie par une vérification des modifications réalisées ce qui permet à l’utilisateur d’avoir une idée précise de ce que le programme fait. À la fin de chaque étape, l’utilisateur n’aura qu’à presser une touche pour nettoyer la console et passer à l’étape suivante. Ce fonctionnement lui permettra de mieux comprendre ce que fait le programme.

Toutefois, nous avons pensé qu’une simple démonstration sans interaction ne permettrait pas à l’utilisateur de prendre conscience de ce que peut réaliser le programme. C’est pourquoi nous avons décidé de mettre en place un menu, en plus de la démonstration, bien que ce ne soit pas spécifié dans le sujet. Ce menu permettra à l’utilisateur de pouvoir agir à sa guise sur les données de Zombillenium et ainsi de voir l’étendue des possibilités liées à ce logiciel avant que nous ne mettions en place l’interface du projet.

**Méthodes mises en place**

Afin de simuler la gestion du personnel du parc, nous avons mis en place deux programmes pouvant être choisis directement par l’utilisateur en début de manipulation.

Le premier programme se présente sous la forme d’une démonstration permettant à l’utilisateur de voir les différentes fonctions de notre programme sans aucune interaction. Toutes les étapes sont gérées par le code.

Le second programme se présente sous la forme d’un menu interactif permettant à l’utilisateur de tester chaque méthode à sa guise. Comme voulu, il se trouve dans la classe Administration.

Ci-dessous nous allons vous expliquer comment fonctionne chaque méthode.

1. Lire un fichier CSV :

La lecture du fichier CSV va nous permettre d’ajouter les différents membres et attractions du parc dans le logiciel. Cet ajout se fait grâce à la fonction AjoutMembresFromCSV(string chemin) qui va accéder au fichier CSV grâce au chemin placé en paramètre et qui va, en fonction de la nature de l’individu, instancier un nouveau membre ou attraction. Si la nature de l’individu en lecture est un démon, alors on appellera la méthode AddDemon(tab) qui va instancier un nouveau démon en fonction des valeurs de la ligne lui correspondant grâce à son constructeur.

1. Ajout de nouvelles attractions et personnel :

L’ajout d’un nouveau membre/attraction peut se faire de deux manières : dans la démonstration, l’ajout va pouvoir se faire en dur en instanciant directement un nouveau membre dans la liste toutLePersonnel ou attractions de la classe Administration. Dans le menu, nous faisons appel à la méthode AjouterMembres() qui va elle-même appeler les méthodes Ajouter\_MembreManuellement() ou Ajouter\_AttractionsManuellement() en fonction du choix de l’utilisateur. Chacune de ces fonctions va, en fonction du choix de l’utilisateur, instancier un membre ou une attraction de la nature correspondante (par exemple AjouterDemonManu(), AjouterSpectacleManu()).

1. Évolution des fonctions, affectations et maintenances :

Dans la démo, les modifications sur les fonctions du personnel, des affectations et des maintenances se font à l’aides des méthodes ChangeFonction(Personnel personnel, string new\_fonction), ChangeAffectation(Monstre monstre, int new\_affectation) et ChangeOuverture() qui vont modifier directement les critères correspondants chez les individus voulus. Le changement de maintenances va se faire en dur dans la méthode elle-même.

Dans le menu, les changements se font de manière interactive dans la méthode Change(Administration adm) de la classe Program qui va proposer à l’utilisateur de choisir quel critère modifier, sur quel membre (en vérifiant s’il existe), tout en spécifiant la nouvelle fonction ou affectation, et donc appeler les fonctions précédentes dans le cas correspondant. Pour le changement de maintenance, tout se fait de la même manière après vérification de la validité de chaque critère entré par l’utilisateur puis en appelant la méthode ChangeOuverture2(Attraction a,string maintenance) qui va faire la modification.

1. Sorties consoles et CSV :

Demo : Dans la démonstration nous faisons tout en dur et nous faisons donc un exemple où nous sortons la liste des vampires du parc.

Menu : Dans le menu, la sortie de données se fait avec la méthode SortieSelonCritere() qui va demander à l’utilisateur de déterminer s’il veut un mode de sortie sur la console (1) ou dans un fichier CSV (2). La méthode va ensuite interpréter sa réponse sous forme de booléen pour appeler la méthode Sortie\_Critere(bool console) ou sinon renvoyer un message d’erreur dans le cas où la demande n’est pas interprétable. La méthode Sortie\_Critere va ensuite demander si l’utilisateur veut faire une sortie de la liste du personnel par catégorie ou s’il veut sortir la liste des attractions en maintenance. Dans le premier cas, un renseignement sur la nature du personnel à sortir sera demandé avant la sortie finale, et dans le second cas on sortira toutes les attractions en maintenance.

1. Tris :

Menu : Ici nous avons décidé de réaliser plusieurs tris et surtout de manières différentes. Dans la méthode Tri(), nous allons offrir la possibilité à l’utilisateur de trier :

1. Les monstres par cagnottes (Tri\_Cagnottes()) qui créera une liste de monstres à partir de celle du personnel et qui fera appel à l’interface ICompare,déjà déclarée dans l’univers VisualStudio, pour comparer les cagnottes des monstres entre elles dans la méthode Sort().
2. Les démons par force(Tri\_Demon()) qui se fera de la même manière.
3. Les loup-garous par cruauté qui se fera de la même façon sans toutefois le faire dans une méthode dédiée.
4. Les vampires par luminosité de la même manière que les loup-garous.

Dans la démo, nous avons décidé de n’appeler que les tris de cagnottes et de force pour l’exemple.

1. Modification des cagnottes :

Dans la démo, nous appelons la méthode ModifierCagnotte(Monstre monstre, int cagnotte), méthode que nous vous expliquerons plus précisément par la suite et qui va modifier directement la cagnotte des montres, à condition qu’il s’agisse d’une manipulation faite par Rose, puis vérifier s’il y a des modifications d’affectations à effectuer suite à cette action.

Dans le Menu, tout cela se fera en interaction avec l’utilisateur.

1. Gestion des conséquences du changement des cagnottes :

La gestion des conséquences des actions sur les cagnottes se fait donc directement lors d’une action sur celles-ci.

Afin de réaliser ces étapes, nous avons dû mettre en place d’autres méthodes utiles comme :

* Les méthodes de CAST qui sont utilisées notamment lors de la lecture du fichier CSV et de la déclaration de membres et attractions afin d’éviter les erreurs de transcription.
* ReturnIndexList(int matricule, bool PersonnelOuAttraction) : méthode qui va nous permettre de connaitre directement la position (index) d’un membre ou d’une attraction dans leurs listes respectives afin d’éviter de réaliser des boucles de parcours de liste à chaque fois et donc dans le but d’optimiser le code.
* La méthode ModifierCagnotte(Monstre monstre, int cagnotte) est une méthode qui nous a posé pas mal de soucis. En effet, seule Rose a accès à cette méthode alors que tous les types de monstre ont une cagnotte… Nous avons donc décidé de créer la méthode virtuellement dans la classe Monstre et dans chaque classe fille, nous avons fait des overrides vides de cette méthode volontairement pour ne pas faire crasher le programme. Cette méthode n’est donc réellement implémentée que dans la classe Vampire à condition que le vampire en question soit bien Rose VonBloodt.
* La classe Personnel est une classe abstraite car elle n’est jamais incrémentée, elle sert de classe mère pour rendre encore une fois le code plus lisible.

**Gestion d’erreurs**

La gestion des erreurs va se faire sous forme d’un try-catch comme par exemple dans la méthode AfficherSelonCategorie(bool console) où un message d’erreur venant de la méthode MessageErreur() sera renvoyé si la saisie de l’utilisateur n’est pas valide (chaine de caractères au lieu d’un entier par exemple).

Dans d’autres méthodes comme Sortie\_Critere(bool console), nous avons créé un cas par défaut (default) qui renverra le même message d’erreur dans le cas où la saisie n’est pas valide.