UCLouvain

LINFO1104 : LethOz Company

Auteurs :
Liam Sawadogo
Matéo-Francisco
Willaime-Freiria

Groupe :
Y

Table des matières

1 Introduction

Dans le cadre du cours LINFO1104, il nous a été demandé d'implémenter un jeu stratégique dénommé LethOz Company. C'est un jeu multijoueur dans lequel les joueurs dirigent des vaisseaux spatiaux dans un champ d'astéroïdes. Le but étant de collecter des déchets. Nous nous sommes basés sur la nature multi-paradigme du langage de programmation Oz, en utilisant la programmation déclarative, la gestion de l'état du jeu et l'application d'effets dynamiques.

Dans ce rapport, il sera question des choix de conception de nos fonctions, des algorithmes choisis et de la mise en page de notre code. En plus, des difficultés que nous avons rencontré durant la conception de notre code. Ainsi que, la complexité des fonctions utilisées et le choix des extensions supplémentaires qui ont été implémenté.

2 Choix de conception

Pour implémenter nos fonctions Next et DecodeStrategy. Nous avons créé plusieurs fonctions auxiliaires :

— Tail_incr qui s'occupe de retourner tous les éléments d'une liste excepté le dernier ;

— Change_dir qui modifie la direction du spaceship ;

— Mouvement qui modifie une coordonnée x ou y de la tête du vaisseau ;

— Effect qui s'occupe d'appliquer les effets sur le vaisseau ;

— Scrap qui étend le vaisseau d'une case à la fin de son mouvement ;

— Repeat qui s'occupe de créer une liste de N éléments contenant N fonctions faisant appel à Next avec l'instruction donnée en paramètre ;

— IterativeInstr qui renvoie une liste avec une fonction faisant appel à Next avec l'instruction donnée en paramètre ;

— Reverse qui inverse le vaisseau.

Dont, nous faisons appel dans le corps des fonctions de Next et de DecodeStrategy.

3 Difficultés rencontrées

Durant ce projet, nous avons rencontré plusieurs difficultés. Notamment, durant la rédaction de notre code où une erreur survenait couramment "illegal arity in application" suite à l'oubli de mettre les fonctions appelées dans des variables locales.

Une autre difficulté concernait l'implémentation que nous devions utiliser pour modifier des éléments dans le Spaceship. Pour ce faire, nous avons décidé de faire appel à la fonction AdjoinAt, mais nous avons pris connaissance de l'existence de cette fonction assez tardivement.

La compréhension des consignes du projet nous avait aussi posé problème. Par exemple, nous ne savions pas si les directions devaient seulement changer pour le premier et le deuxième pos du Spaceship ou si d'autres devaient aussi être modifiés. Un autre exemple, concerne la fonction Scrap pour savoir si les coordonnées du nouveau pos avait de l'importance ou non.

La dernière difficulté que nous avons rencontré était liée à la compréhension de la fonction DecodeStrategy. Nous avions pris du temps pour comprendre que la fonction devait renvoyer une liste d'instruction faisant appel à Next et que le paramètre pris en compte était soit une liste d'instructions, soit l'élément nil.

De plus, nous n'avions pas une connaissance approfondie par rapport aux records. Par exemple, au début nous ne savions pas s'il était possible d'accéder à un élément d'un record et de le modifier directement en faisant +1 ou autre.

Un autre souci concernait l'exécution du programme sur MacOs, il fallait modifier le path ainsi que la librairie, dont les couleurs et le makefile.

4 Complexité des fonctions

Au niveau de la complexité de nos fonctions, la plupart d'entre elles sont de l'ordre de $\mathcal{O}(n)$ ou de $\mathcal{O}(1)$.

— Pour la fonction Tail_incr vu que nous parcourons une liste de manière récursive, la fonction est de l'ordre $\mathcal{O}(n)$;

— Pour la fonction Change_dir nous supposons qu'elle est de l'ordre de $\mathcal{O}(n)$ vu que nous faisons appel à la fonction AdjoinAt, qui s'occupe de créer un nouveau record avec un élément modifié ;

— Pour la fonction Mouvement nous supposons qu'elle est de l'ordre de $\mathcal{O}(n)$ vu que nous faisons appel à la fonction AdjoinAt, qui s'occupe de créer un nouveau record avec un élément modifié ;

— Pour la fonction Scrap vu que nous parcourons une liste de manière récursive, la fonction est de l'ordre de $\mathcal{O}(n)$;

— Pour la fonction Repeat, elle est de l'ordre de $\mathcal{O}(n)$ vu que nous rajoutons N fois des éléments dans une liste vide ;

— Pour la fonction IterativeInstr, elle est de l'ordre de $\Theta(1)$ vu que dans tous les cas nous retournons une liste contenant un seul élément ;

— La fonction Reverse est de l'ordre de $\mathcal{O}(n^2)$ vu que nous faisons appel à la fonction AdjoinAt avec le premier élément d'une liste, qui s'occupe de créer un nouveau record avec un élément modifié et qui s'appelle de manière récursive avec le reste de la liste ;

— La fonction DecodeStrategy est de l'ordre de $\mathcal{O}(n^2)$ vu que nous faisons appel à la fonction Repeat qui est de l'ordre de $\mathcal{O}(n)$ et qui s'appelle de manière récursive avec le reste de la liste ;

— La fonction Effect est de l'ordre de $\mathcal{O}(n^4)$ vu que nous faisons appel à la fonction Reverse qui est de l'ordre de $\mathcal{O}(n^2)$ qui est imbriqué dans la fonction AdjoinAt qui est de l'ordre de $\mathcal{O}(n)$ et qui s'appelle de manière récursive avec le reste de la liste ;

— La fonction Next est de l'ordre de $\mathcal{O}(n^4)$ vu qu'elle fait appel à la fonction Effect ;

— La fonction Malware est de l'ordre de $\mathcal{O}(n)$ vu qu'on parcourt une liste avec l'ajout d'un élément ;

— La fonction Shrink est de l'ordre de $\mathcal{O}(n)$ vu qu'elle parcourt une liste ;

— La fonction BigScrap est de l'ordre de $\mathcal{O}(n)$ vu qu'elle parcourt une liste ;

— La fonction Malware est de l'ordre de $\Theta(1)$ vu qu'elle ne modifie qu'une direction.

5 Extensions implémentées

Pour les extensions supplémentaires nous avons décidés d'implémenter les effets Malware, Shrink, BigScrap et Revert-all.

Malware est un effet qui inverse les directions turn(left) et turn(right) durant N tours.

Shrink réduit la taille du vaisseau en retirant un wagon du Spaceship.

BigScrap est une amélioration de Scrap qui augmente la taille du vaisseau avec 3 nouveaux wagons.

Revert-all inverse les positions de tous les vaisseaux présents sur la map.

6 Conclusion

En conclusion, ce projet nous a permis d'avoir une meilleure compréhension du langage de programmation Oz et d'acquérir pleinement des concepts du multi-paradigme. Malgré les quelques difficultés que nous avons rencontré, nous avons sû les surmonter grâce à la documentation Mozart et en posant nos questions lors des tp mais surtout sur des forums ou entre camarades.