Rapport du TP noté *Igel Ärgern*

Benjamin WOJTECKI et Nowar KAZEM

October 22, 2024

Table des matières

1 Temps passé

Nous avons consacré chacun une vingtaine d'heures pour réaliser le projet. Ces heures incluent non seulement le temps dédié à coder, mais aussi les moments de réflexion sur la structure du projet et les réflexions sur le comportements de certaines fonctions que nous souhaitions modéliser.

2 Introduction

Nous avons implémenté le jeu *Igel Ärgen* créé par Frank Nestel et Doris Matthaus. Pour réaliser ce jeu, nous avons utilisé le langage de programmation C. De par sa simplicité conceptuel, il présente un grand intérêt pour une implémentation en C afin de s'entrainer à coder proprement. Il se joue à plusieurs joueurs sur une grille composée de n lignes et m colonnes, où n et m sont des dimensions choisies par les joueurs. Chaque équipe, disposant de k hérissons (nombre également configurable), doit faire avancer $k-1$ d'entre eux jusqu'à la dernière colonne pour remporter la partie, tout en respectant certaines règles spécifiques.

Le jeu, dans sa version sans extensions, peut être facilement affiché dans un terminal, ce qui en fait un projet à la fois intéressant, simple et accessible à implémenter. Il ne nécessite pas de techniques avancées de programmation en C, tout en offrant un bon exercice de manipulation des bases du langage.

3 Le travail

3.1 Comment faire marcher notre code ?

Un *Makefile* est inclus dans le projet, il suffit donc de taper la commande `make` dans le terminal, suivie de `./igelargern` pour lancer le jeu. Une interface interactive s'ouvre alors, et une série de questions est posée aux joueurs pour configurer et démarrer la partie. Contrairement à ce qui est indiqué dans l'énoncé du TP, il n'est pas nécessaire de passer des paramètres dans la fonction main. Cela est rendu possible grâce à une fonction appelée `playgame()`, qui gère les réponses des joueurs et appelle les fonctions appropriées. Rendant ainsi le jeu beaucoup plus interactif pour les utilisateurs.

Nous insistons sur le fait que nous avons pris le parti de rendre le jeu le plus interactif possible. En raison de ce choix, une grande partie de notre temps a été dédiée à rendre le jeu le plus interactif possible. Par exemple, lors d'une entrée invalide de l'utilisateur, nous affichons l'erreur de saisie commise, puis nous reposons la question et lui demandons de ressaisir la réponse. De par cette méthode, nous minimisons le risque que des erreurs, comme des segmentation faults, apparaissent. Cependant, nous ne pouvons malheureusement pas garantir avec certitude qu'elles soient totalement évitées, malgré un ensemble de tests exhaustifs.

Pour configurer le nombre d'équipes, le nombre de hérissons et les dimensions de la grille, il faut modifier les valeurs des variables correspondantes directement dans le code (ce sont des `define`), situées en tête du fichier `plateau.h`. Ce sont les seuls paramètres qui ne peuvent pas être changés via l'interface. Nous aurions pu utiliser des variables globales pour permettre leur modification, ou même ajouter des paramètres à nos fonctions.

3.2 L'extension 1 : Ajout d'un menu et d'une interface interactive

Une fois l'implémentation du jeu réussie, nous avons choisi de rendre le jeu interactif en demandant à l'utilisateur de saisir des informations. Pour ce faire, nous avons d'abord ajouté un menu lorsqu'on lance le jeu. On obtient l'affichage suivant :

```
--------------------------------MENU-------------------------------------
Bienvenue dans le jeu :
1. Jouer sans extension
2. Jouer avec extension
3. Credits
```

Quand on demande à l'utilisateur de saisir une valeur, notre première approche a été d'utiliser un `scanf()`. Malheureusement, cette approche est peu efficace, car si l'utilisateur entre des caractères non voulu, on obtient une erreur de type segmentation fault. Pour résoudre ce problème, nous avons adopté une autre approche qui consiste à utiliser `fgets()` pour obtenir la saisie de l'utilisateur. Ensuite, nous travaillons sur le tableau de caractères renvoyé par la méthode `fgets`. Au cas où le joueur entre une réponse suffisament longue pour dépasser le nombre de bits alloués à la réponse, l'entrée est évidement invalide et pour éviter que le message d'erreur s'affiche plusieures fois nous vidons le buffer à l'aide de la fonction `clean_buffer`. Cette implémentation a été extrêmement chronophage. Enfin, pour rendre le menu plus joli, nous avons ajouté un *Igen Ärgern* en ASCII art.

3.3 L'extension 2 : Ajout de cases "portail"

Cette extension s'inscrit dans la même logique que toute extension visant à modifier les règles du jeu. L'idée de l'extension est simple (elle n'est pas tirée du site). En plus de l'existence de cases piégées, nous ajoutons des cases portail. Une case

portail se comporte comme une case normale, sauf que si un hérisson tombe dans cette case (qu'il soit déplacé horizontalement ou verticalement), il est transféré vers une autre case portail. Cela implique qu'il existe au moins deux cases portail sur le plateau. Un champ a été ajouté à la structure `casePlateau` nommé `portal`, qui indique si la case est un portail ou non. Nous avons modifié les fonctions `forward_move`, `horizontal_move`, `cell_print`, ainsi que `create_board` pour gérer cette extension. Pour `forward_move` et `horizontal_move`, il fallait traiter le cas où la case vers laquelle le hérisson avançait était un portail. Pour cela, nous avons ajouté une fonction `find_other_portal` pour trouver un autre portail où transférer le hérisson. Pour `cell_print`, c'était une question d'affichage, et nous avons décidé de présenter les portails de la manière suivante :

Dans notre implémentation actuelle, nous générons deux portails aléatoirement. Avec du temps supplémentaire, on pourrait déterminer de façon aléatoire le nombre de portails. En plus de cela, s'il existe plusieurs portails, il faudrait définir une méthode permettant de déterminer le sens lors du passage à travers un portail. Pour implémenter cette méthode, on pourrait soit faire en sorte que dès qu'un hérisson se trouve sur une case portail, il ait une équiprobabilité de se téléporter à un autre portail. Une autre manière de procéder est que, lors de la création des cases portails, on leur attribue un numéro représentant l'ordre de création. Lorsque le hérisson passe par le portail 1, il sera téléporté au dernier portail. Enfin, pour éviter de dupliquer le code pour faire la distinction entre jouer avec ou sans l'extension, nous avons tout simplement ajouté un paramètre à `create_board`, qui indique l'intention du joueur. Si ce paramètre est faux, nous n'ajoutons aucun portail (la valeur de `portal` est `false` pour toute case), et le code se comporte comme avant.

4 Synthèse

Ce projet est notre premier de l'année, et nous y avons porté une attention particulière. Il a été un excellent moyen de mieux nous connaître et de nous habituer à travailler ensemble. À travers ce travail, nous avons appris plusieurs choses importantes.

Nous avons découvert comment utiliser Git pour gérer les versions et collaborer à distance, ainsi que quelques outils utiles pour programmer en C. Dès le début, nous avons mis l'accent sur un code propre, ce qui a grandement facilité notre travail par la suite.

Cependant, nous avons rencontré plusieurs difficultés. La première était de travailler à distance, car il fallait s'accorder sur les structures de données et les fonctions de base. La partie la plus difficile a été de gérer les interactions avec les joueurs, avec de nombreux cas à traiter tout en gardant un code lisible et propre.

Une fois le jeu de base terminé, il a été plus facile d'ajouter des extensions. Nous regrettons seulement de ne pas avoir eu assez de temps pour en ajouter d'autres.

5 bibliographie

- Le site https://www.w3schools.com/ nous a été utile pour revoir quelques fonctions en C.
- Mathieu LAURENT nous a aidé lors du TP didié au projet ainsi que lors du cours MPI.

- Le site https://stackoverflow.com/ a répondu a certaines de nos questions.
- Le site https://openclassrooms.com/ pour son cours sur le C.
- Notre cours en SYS et surtout la fiche *Mémento du langage C* qui nous a fourni le fonction `clean_buffer`.