TP4: Igel Ärgern – TP noté

Module ArcSys

TP à rendre pour le mardi 22 octobre 2024 avant 19h00 (CEST)
À faire en binômes

L'objectif de ce projet est d'implémenter une course entre hérissons, d'après le jeu de plateau "Igel Ärgern", par Frank Nestel et Doris Matthaus.

Si vous le souhaitez, vous pouvez implémenter intégralement ce projet sans malloc ni constructions avancées du C au-delà des structures de données. **La difficulté majeure de ce projet, c'est d'écrire un code propre et lisible.** C'est important pour le relecteur qui va vous noter, mais c'est également très important pour vous-même. Si votre code n'est pas très lisible, si vous n'êtes pas cohérent dans vos fonctions et notations, il sera bien plus dur d'implémenter ce projet jusqu'au bout. Utilisez `clang-format` et `clang-tidy` sur votre code.

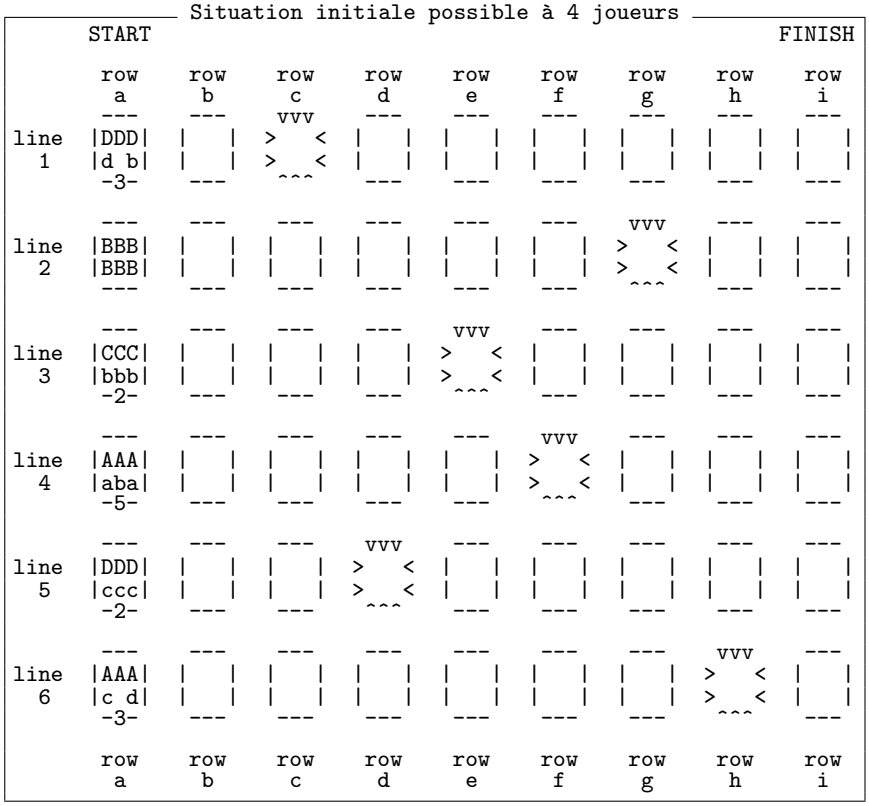
La propreté d'un programme est liée à son découpage en fonctions et modules. On doit voir l'architecture générale du programme en observant les structures de données définies et le prototype des fonctions. La lisibilité d'un programme tient à son indentation qui doit être consistante, à la présence de commentaires utiles (c-à-d, expliquant l'intention du code sans le paraphraser et explicitant les invariants et préconditions des parties complexes), et à l'usage d'identificateurs bien choisis pour les fonctions et variables. Si vous avez besoin de documenter l'intention d'un paramètre, peut-être devriez-vous plutôt le renommer.

Une autre difficulté du projet tient au fait qu'il est un peu long et qu'il est difficile de travailler séparément dès le début. Prévoyez de travailler plusieurs heures ensemble, au moins au début. Vous devriez utiliser git pour partager votre travail, si vous savez faire ou si vous avez le temps d'apprendre dès maintenant.

★ Règles du jeu

Le plateau de jeu se décompose en 6 lignes de 9 cases chacune. Initialement, tous les hérissons (4 par joueur) sont placés aléatoirement sur les cases de la première colonne nommée a. Quand il y a plusieurs hérissons sur la même case, ils s'empilent et seul celui du sommet de la pile peut bouger.

Situation initiale possible à 4 joueurs



Les hérissons qui parviennent à la colonne i ont fini la course. Lorsqu'une équipe parvient à placer trois de ses hérissons en colonne i, cette équipe gagne. Le tour termine puis la partie s'arrête.

La case $a1$ ci-contre contient 3 hérissons (c'est écrit sous la case). De haut en bas, ils sont dans les équipes "D", "D" puis "B".

La case $a2$ ne contient qu'un seul hérisson, de l'équipe B.

La case $a3$ contient deux hérissons : le "C" est au-dessus du "B".

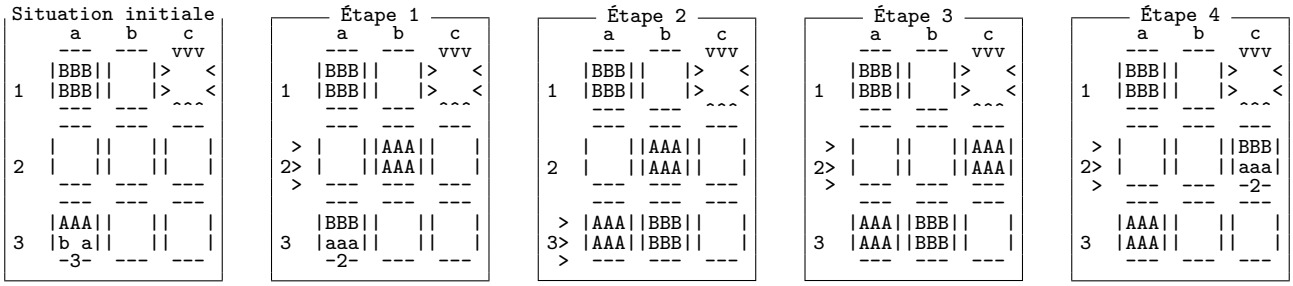
La case $a4$ contient cinq hérissons, mais seuls les quatre premiers sont représentés. De haut en bas, on trouve les équipes "A", "A", "B" puis "A" encore. Le 5ieme hérisson n'est pas près de bouger avec autant de monde sur la tête. Il n'est donc pas utile de connaître son équipe.

Certaines cases du plateau comme la c1, la g2 ou encore la e3 sont représentées avec une bordure particulière, car elles sont piégées : Un hérisson qui tombe dedans ne peut plus en ressortir tant qu'il reste du monde derrière lui sur sa ligne. Par exemple, un hérisson placé en c1 ne peut en sortir que si les cases a1 et b1 sont vides.

Déroulement d'un tour. Le joueur dont c'est le tour lance un dé, qui désigne la ligne sur laquelle un hérisson va pouvoir avancer. Le joueur peut déplacer verticalement l'un de ses hérissons d'une ligne s'il le souhaite, puis il choisit le hérisson qui avance horizontalement sur la ligne désignée par le dé. On ne peut faire monter ou descendre que ses propres hérissons (et à condition qu'ils n'aient personne sur la tête), mais on peut déplacer vers la droite les hérissons des autres joueurs. S'il existe au moins un hérisson pouvant se déplacer vers la droite, le joueur doit déplacer l'un d'entre eux.

Exemple de partie avec une grille 3x3 cases et deux hérissons par équipe.

- *Étape 1 :* 'A' a tiré un 2. Il a décalé vers le haut son pion de a3 vers a2, puis l'a fait avancer vers b2.
- *Étape 2 :* 'B' a tiré un 3. Il n'a décalé personne, et fait avancer son pion de a3 jusqu'en b3.
- *Étape 3 :* 'A' a tiré un 2. Il a fait avancer son pion de b2 pour le placer sur la colonne d'arrivée. Avec deux hérissons par équipe seulement, il suffit d'en placer un pour que la partie s'arrête à la fin du tour.
- *Étape 4 :* 'B' a tiré un 2. Il a décalé son pion de b3 vers le haut, puis lui a fait franchir la ligne également. La partie se termine sur un match nul : 1 partout.



★ Étape 1 : Faire le plateau.

La première étape du projet consiste à implémenter la représentation mémoire du plateau de jeu, et le code permettant d'initialiser aléatoirement le plateau puis de l'afficher.

Devoir jouer une partie entière est relativement fastidieux pour debugger. Nous allons donc faire en sorte que toutes les grandeurs du jeu soient définies par des constantes dans notre programme, de façon à pouvoir jouer sur une grille 3x3 comme ci-dessus, avec 2 hérissons par équipe pour jouer vite ou au contraire 300 hérissons par équipe pour tester le code d'affichage. Avoir plus de 26 joueurs est difficile sans changer d'alphabet, et ne présente pas un grand intérêt. Les cases piégées sont toujours aux mêmes coordonnées, visible sur le plateau page précédente. Si le plateau est trop petit, celles hors plateau sont ignorées.

Les cases. On va représenter chaque case du plateau avec un `struct` C spécifique. La pile de hérissons peut se représenter avec un tableau de caractères de taille fixe et un entier indiquant le sommet de la pile dans le tableau.

Le plateau. Le nombre de cases étant fixe dans une partie, il ne semble pas nécessaire d'avoir le moindre `malloc` dans le programme. On peut tout faire avec des tableaux de taille fixe, avec les constantes de dimensions définies plus haut.

On a besoin de fonctions pour ajouter ou retirer un hérisson au sommet de la pile d'une case donnée (`push` et `pop`), pour connaître le nombre de hérissons sur une case (`height`), et pour connaître l'équipe du hérisson à une position donnée de la pile d'une case (`top` et `peek`).

```
void board_push(board_t* b, int line, int row, char ctn);
char board_pop(board_t* b, int line, int row);
int board_height(board_t* b, int line, int row);
char board_top(board_t* b, int line, int row);
char board_peek(board_t* b, int line, int row, int pos); // pos=0 => top
void cell_print(board_t* b, int line, int row, int slice);
void board_print(board_t* b, int highlighted_line); // hl_line=-1 => rien de sélectionné
```

La fonction `cell_print()` permet de dessiner une tranche de la cellule, c'est-à-dire l'une des lignes de caractères la représentant. Si `slice==0`, on dessine la bordure nord. Pour `slice==1`, on dessine la ligne de contenu du haut, et ainsi de suite. Cette fonction est un peu fastidieuse à écrire, mais elle simplifie grandement l'écriture de `board_print()`, qui dessine tout le plateau avec les légendes autour, et une indication de la ligne actuellement sélectionnée par le dé.

Il est important de tester ce code correctement, pour différentes dimensions de jeu, avant de passer à la suite.

★ Étape 2 : Logique de base du jeu

Il s'agit maintenant d'implémenter la logique nécessaire pour faire avancer les hérissons. Tant que la partie n'est pas finie, on organise le tour de chaque joueur.

Étape du tour d'un joueur.

- On tire un dé et l'on affiche le résultat.
  Rappel de C : on tire un nombre aléatoire entre 1 et 6 avec le code de la ligne 3 ci-dessous. Il faut au préalable avoir initialisé le générateur pseudo-aléatoire comme à la ligne 6.

```
1 #include <time.h>
2 int de() {
3   return rand() % 6 + 1;
4 }
5 void main() {
6   srand(time(NULL));
7
8   /* Le reste de votre code */
9 }
```

- On demande au joueur s'il veut déplacer un hérisson vers le haut ou vers le bas, et lequel. Attention à bien gérer toutes les fautes de frappe de l'humain avant d'appliquer le changement demandé. Par exemple, on ne déplace pas les hérissons des autres joueurs à cette étape.
- On demande au joueur quel hérisson il veut déplacer sur la ligne désignée par le dé s'il en existe un, ou on l'informe qu'il doit passer son tour dans le cas contraire. On peut maintenant déplacer les hérissons des autres, mais on ne peut pas passer son tour s'il existe un hérisson déplaçable.

Cases piégées. Il faut veiller à ne pas laisser bouger un hérisson se trouvant dans une case piégée s'il reste d'autres hérissons à sa gauche sur la même ligne.

Conditions de victoire. Quand un hérisson atteint la dernière colonne, il a fini (il ne faut d'ailleurs plus autoriser les joueurs à le déplacer). On ajoutera un champ à la structure `board_t` pour comptabiliser le nombre de hérissons par équipe ayant terminé la course.

Quand tous les hérissons d'une équipe sauf un a fini, l'équipe a gagné. La partie se termine après que le dernier joueur a joué. On affiche ensuite les résultats de la partie en gérant les ex æquo de la manière suivante.

```
- Place #1 : équipe A, équipe C (avec 3 hérissons);
- Place #3 : équipe B (avec 1 hérisson)
```

★ Extensions

Les auteurs de ce jeu proposent de très nombreuses variantes de règles, que l'on trouve à l'adresse suivante : `https://www.gamecabinet.com/rules/IgelArgernVariants.html` (copie locale). On peut imaginer des extensions au delà de cette liste, comme une interface graphique basée sur ncurses ou SFML, ou bien une intelligence artificielle, ou encore une variante de règle originale. Comme c'est un projet de programmation, l'intérêt d'une extension donnée ne dépend pas du gameplay apporté, mais plutôt de la difficulté de programmation induite en termes algorithmiques et/ou design logiciel.

La chose la plus importante pour avoir une bonne note est d'écrire un code parfaitement propre. On peut avoir une très bonne note sans implémenter la moindre extension, si le code et le rapport sont de grande qualité. Libre à vous cependant d'implémenter quelques variantes (ce qui apportera des points bonus), mais dans tous les cas **il est interdit d'implémenter plus de trois extensions du jeu**.

★ Rendu et attendus

Ce projet est à faire en binôme. Il est interdit de travailler seul. Chaque groupe doit rendre une archive tar compressée contenant les sources du programme, un rapport en pdf d'au plus cinq pages, et les éventuels fichiers de tests que vous aurez écrits. L'archive doit être envoyée à l'adresse `martin.quinson@ens-rennes.fr`. Tout votre programme doit être écrit en C. Si vous avez utilisé git, vous pouvez donner l'adresse de ce git à la place de l'archive (en vous assurant que nous y avons accès).

■ Le code

Votre code doit rester modulaire et permettre de choisir les extensions à activer depuis la ligne de commande avec `argc/argv`. La taille du plateau de jeu doit également être configurable facilement, que ce soit en ligne de commande ou avec des `#define` clairement indiqués dans le rapport. Chaque extension doit être décrite convenablement dans votre rapport (voir ci-dessous). Vous fournirez un Makefile ou CMakeLists.txt permettant de compiler votre projet, en veillant à utiliser quelques options de compilation `-W???` bien choisies.

Vous porterez un soin particulier à l'écriture du code. *Pensez à nettoyer pour ne pas rendre un brouillon.* Le code doit être bien écrit et commenté pour être facilement lisible. On écrit un programme pour que des humains puissent le lire, et (accidentellement seulement) pour que les machines puissent l'exécuter. Nous analyserons également votre code avec les outils `clang-tidy` et `valgrind`, même si la présence de quelques erreurs indiquées par ces outils n'est pas rédhibitoire. Il peut être utile que vous les utilisiez sur votre code avant de le rendre. Enfin, nous lirons attentivement votre code. Quelques conseils se trouvent sur wikipédia[1] et dans le poly du cours.

■ **Le rapport**

Votre rapport doit apporter une description claire et détaillée de chaque extension implémentée, sans reprendre trop de code. La note finale tiendra compte à la fois de votre rapport et de votre code. Relisez-vous avant d'envoyer votre travail ! Votre rapport doit comporter (au moins) les parties suivantes. Ce plan peut paraître un peu artificiel, mais il reprend le plan classique d'un écrit scientifique comme vous devrez en écrire pour votre rapport de stage. Autant chercher à s'entraîner dès maintenant.

- **Le nom des membres du binôme,** ainsi qu'une indication du temps passé par chaque membre du binôme sur le projet. Soyez honnête, car votre réponse ici n'aura aucun impact sur la notation. Il s'agit simplement d'adapter la complexité du projet les années prochaines, au besoin.
- **Introduction :** quelques mots pour présenter ce projet (ce que l'on va faire et pourquoi c'est intéressant).
- **Une description claire de chaque extension implémentée :** soyez pédagogiques pour expliquer comment vous avez procédé en reprenant aussi peu de code que possible. Une explication dans le rapport de la méthode envisagée sera considérée même si le code n'est pas réalisé.
- **Synthèse :** une conclusion sur vos observations, expérimentations et résultats, etc. Éventuellement, si vous en avez, des commentaires sur ce qui pourrait être amélioré pour l'année prochaine ou des idées d'extensions intéressantes.
- **Bibliographie :** Donnez la liste de toutes les sources (sites, livres ou individus) qui vous ont aidé, avec quelques mots de ce que vous en avez retiré. Attention, la frontière est mince entre *l'oubli* de certaines sources et le plagiat. N'oubliez rien, ne trichez pas.

■ **Ne trichez pas**

Par tricher, nous entendons notamment :

- Rendre le travail de quelqu'un d'autre avec votre nom dessus ;
- Obtenir une réponse sur internet ou autre et mettre votre nom dessus. Changer le nom des variables et fonctions ou leur ordre avant de mettre votre nom dessus est considéré comme une tricherie aggravée bien que cela ne suffise pas à tromper un système anti-plagiat comme MOSS[2];
- Permettre à un collègue de *s'inspirer* de votre travail. Assurez-vous que votre dépôt est privé.

En cas de litige grave, seul un historique progressif de vos travaux (comme en offre git) constitue une preuve de votre innocence. *Commit soon, commit often.*

En revanche, il est possible (voire conseillé) de discuter du projet et d'échanger des idées avec vos collègues. Mais vous ne pouvez rendre que du code écrit par vous-même, et vous devez détailler brièvement l'intégralité de vos sources inspirations dans la partie bibliographie de votre rapport. La ligne jaune à ne pas dépasser est la lecture du code : on peut discuter, et même faire des schémas au tableau, mais **vous ne devez jamais lire du code écrit par un autre groupe. Sous aucun prétexte.**

Votre travail est à rendre pour le **mardi 22 octobre 2024 avant 19h CEST**.

★ À propos de ce document

L'idée d'un projet autour du jeu *Igel Ärgern* vient d'un Nifty Project par Zachary Kurmas en 2014. Les Nifty sont une collection d'idées de projets de programmation publiés chaque année pendant la conférence SIGCSE.

[1] Notions basiques sur la lisibilité d'un code C : `http://fr.wikibooks.org/wiki/Conseils_de_codage_en_C/Lisibilit_des_sources`.

[2] MOSS, `http://theory.stanford.edu/~aiken/moss/`