1 tp4_reine

1.1 Buts

- Utilisation des tableaux à deux dimensions en Rust.
- Utilisation du type énuméré et de l'Option.
- Utilisation des fonctions.
- Continuation des efforts d'apprentissage de l'ownership.
- Utilisation des commentaires dans le code pour la documentation.

1.2 Énoncé

Il s'agit de donner la couverture de la reine sur un échiquier 8×8.

Exemple pour la position de la reine $(5;\ f)$ (cinquième ligne, f-ème colonne)

```
8 . . * . . * . .
7 . . . * . * . *
6 . . . . * * * .
5 * * * * * R * *
4 . . . . * * * .
3 . . . * . * . *
2 . . * . . * . .
1 . * . . . * . .
  a b c d e f g h
```

1.3 Marche à suivre

Le programme comportera quatre parties

1. Création d'un tableau correspondant à l'échiquier, chaque élément étant de type énuméré.
2. Saisie des coordonnées de la reine.
3. Calcul et placement dans le tableau de toutes les cases couvertes par la reine.
4. Affichage du contenu du tableau.

1.3.1 Remarques

1. Vous devez impérativement structurer votre programme avec des fonctions.
2. Vous devez également commenter votre code et rédiger une documentation.
3. A l'aide d'une `Option` faites en sorte que si l'utilisateur rentre une ligne ou une colonne trop grande (après 8 ou après 'h') il soit averti et puisse en rentrer une nouvelle.

1.4 Problèmes supplémentaires

1. Problème des huit reines: Pour ceux · celles qui le souhaitent, écrire un programme qui place sept autres reines de manière à ce qu'aucune ne soit prise.
2. Transformez les fonctions `read_int` et `read_char` de `rust_hepia_lib`, afin de traiter l'erreur et permettre à l'utilisateur de pouvoir corriger son erreur en rentrant une nouvelle ligne ou colonne s'il s'est trompé.

1.5 Remarques

1. Il pourrait vous être utile d'utiliser la fonction `read_char()` de `rust_hepia_lib`. N'oubliez pas de télécharger la dernière version sur https://githepia.hesge.ch/orestis.malaspin/rust_hepia_lib.

2. Afin de créer un tableau de tableau vous devez écrire la ligne suivante

   ```
   let tableau = [[Instance; SIZE]; SIZE];
   ```

   où SIZE est un entier connu à la compilation et `Instance` est une instance d'un `Type`. Si ce `Type` n'est pas `Copy` cette ligne produira une erreur de compilation (essayez pour voir le message d'erreur) car il n'y a qu'une instance de ce type et il faut le dupliquer (tous les éléments du tableau ne peuvent pas avoir l'ownership en même temps). Si votre type n'est pas `Copy`, Rust peut le rendre copiable automatiquement sous certaines conditions. Pour ce faire il faut rajouter la ligne `#[derive(Clone, Copy)]` avant la déclaration du type (il doit également être clonable).

   ```
   #[derive(Clone, Copy)]
   enum Binaire {
       Zero,
       Un,
   }
   ```

   La condition afin qu'un type puisse être rendu `Copy` automatiquement, est que tous les types qu'il contient doivent être eux-même `Copy`. Essayez de compiler les lignes suivantes pour voir ce qu'il se passe.

   ```
   #[derive(Clone, Copy)]
   enum BinaireBis {
       One(Vec<i32>),
       Two,
       Three,
   }
   ```

   Une erreur s'est-elle produite? En effet, `Vec<i32>` n'est pas `Copy`. Rust ne peut donc pas savoir comment rendre votre type `BinaireBis` automatiquement.