1 tp5_puissance4

1.1 But du travail pratique

Le but de ce travail pratique est de réaliser un jeu de puissance 4 et de mettre en pratique les notions vues jusqu'ici. Le jeu doit implémenter trois modes différents :

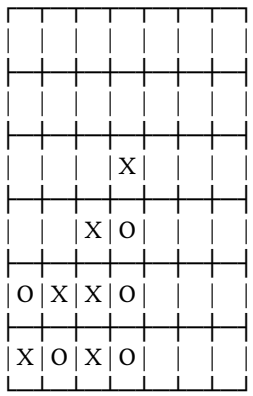
- un mode deux joueurs humains;
- un mode un joueur humain contre l'ordinateur qui joue aléatoirement ;
- un mode un joueur humain contre l'ordinateur qui joue intelligemment: il essaie de gagner au prochain coup ou de ne pas perdre.

Il est primordial d'avoir un design correct. Pour ce faire il faut **modulariser** votre code à l'aide des modules et des fonctions Réfléchissez aux modules de votre code avant de commencer à programmer!

Exemple: l'affichage fait-il partie du même module que la logique du jeu? De quelles fonctions l'affichage a-t-il besoin?

Le rendu visuel du jeu sera réalisé en mode texte "ASCII Art" tel qu'illustré ci-dessous :

```
Column number? (starts at 1):
4
```




```
Player one won!
```

1.2 Règles du jeu

1.2.1 Mode deux joueurs

Le jeu se joue à deux. Chacun leur tour, les joueurs choisissent une colonne. Quand le choix est fait, une pastille "tombe" au bas de la colonne spécifiée. Le premier joueur qui aligne quatre pastilles soit verticalement, soit horizontalement, soit en diagonale a gagné.

- Le plateau de jeu est composé d'un nombre arbitraire de lignes et colonnes (définies à l'aide de constantes); le nombre de lignes ou colonnes doit être supérieur ou égal à quatre.
- A chaque joueur correspond une pastille ; celle-ci est représentée par le caractère 'X' pour le premier joueur et 'O' pour le deuxième (ou l'ordinateur).
- Chacun leur tour les joueurs choisissent une colonne à l'aide du clavier ; la première colonne est numérotée '1'.
- Si une colonne invalide est spécifiée, le programme doit redemander à l'utilisateur d'entrer une colonne (valide).
- Une fois une colonne sélectionnée, la pastille du joueur va se placer en bas de la colonne sur la première case non-occupée (si toute la colonne est occupée il est impossible de poser la pastille et il faut demander au joueur de rejouer).
- Le premier joueur à avoir aligné quatre pastilles horizontalement, verticalement, ou en diagonale, a gagné.
- Si toutes les cases sont remplies et qu'aucun joueur n'a gagné la partie se termine par un match nul.

1.2.2 Mode ordinateur aléatoire

Programmer le jeu pour une seule personne face à l'ordinateur. Dans la première version de cette variante l'ordinateur jouera toujours au hasard (il remplace ainsi le joueur numéro deux ci-dessus).

1.2.3 Mode ordinateur "intelligent"

Dans une version plus sophistiquée, l'ordinateur gagnera au prochain coup s'il le peut, ou le cas échéant, empêchera la victoire de son adversaire si celui-ci se trouvait en position de gagner à son prochain coup. Dans tous les autres cas l'ordinateur jouera au hasard.

1.3 Cahier des charges

1. Programmer le jeu à deux joueurs humains :
   - Afficher l'état de la partie à l'écran où une case libre est représenté par un ' ' (espace), et les cases de chacun des joueurs par un 'X' ou un 'O' respectivement. Le tableau de jeu sera affiché comme dans l'exemple fourni au début ce cet énoncé.
   - La victoire doit être vérifiée après chaque coup.
   - Un match nul est possible si aucun des joueurs n'a gagné et qu'il n'y a plus de case libre.
2. Modulariser le code à l'aide de modules et de fonctions, gérer les erreurs possibles de l'utilisateur, et commenter le code. Votre code doit également être correctement indenté!
3. Programmer le jeu entre un humain et un ordinateur :
   - L'ordinateur jouera toujours un coup au hasard.

4. Programmer une “intelligence artificielle” un peu plus évoluée :
    - L’ordinateur gagnera s’il le peut au prochain coup.
    - L’ordinateur empêchera le joueur de gagner au prochain coup s’il le peut.

1.4 Remarques

Il se pourrait que vous ayez besoin de comparer une instance de type énuméré. Pour cela votre type énuméré doit implémenter certaines fonction. Cela se fait avec la commande `#[derive(PartialEq)]` sur la ligne précédant votre type énuméré. Pour initialiser un tableau statique avec des instances de type énuméré, il faut que ces types soient `Clone` et éventuellement `Copy`. Pour ce faire, vous avez besoin de la ligne suivante précédent votre type énuméré `#[derive(Clone, Copy)]`. Il se pourrait finalement, que pour des raisons de débugging vous vouliez afficher une instance de type énuméré il faut donc qu’il soit `Debug`. Pour ce faire ajoutez la ligne `#[derive(Debug)]` avant votre type énuméré. Finalement, une combinaison de tous ces derive peut être faite, il suffit de les mettre les uns après les autres `#[derive(Clone, Debug, PartialEq)]`.

1.5 Rendu et évaluation

Ce travail pratique est **noté** et **individuel**.

Il est à rendre **au plus tard** le lundi 5.11 à 23h55 (tout retard sera sanctionné). Une seule archive `.zip` à votre nom devra être déposée sur cyberlearn dans 18_HEPIA_Programmation Sequentielle en Rust . Si votre nom est Michel Lazeyras, l’archive sera donc nommée **michel_lazeyras.zip** et devra contenir l’arborescence `michel_lazeyras/puissance4` (si `puissance4` est le nom de votre projet).

Le jeudi 8.11, vous devrez présenter brièvement votre programme et ses fonctionnalités, et être capables d’expliquer toutes les parties de votre algorithme. N’hésitez donc pas à relire votre code et à vous préparer en conséquence avant le 8.11.

Vous serez évalués sur votre capacité à remplir le cahier des charges et votre présentation (votre capacité à expliquer ce que vous avez fait), ainsi que sur “l’élégance” de votre code. En d’autres termes, l’utilisation de types énumérés, d’options, de fonctions, de modules sera également évaluée. Lors de la présentation orale, vous devez nous convaincre que vous avez bien compris ce que vous avez implémenté et compris les concepts de programmation utilisés lors de l’implémentation de votre algorithme.