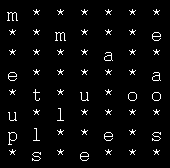
Introduction à la programmation - Projet 2022

Nuplémo : le Mémo à n-uplets

L’objectif de ce projet est de programmer un jeu de Mémo en mode console, en C#, avec une approche procédurale (pas d’utilisation de classes autres que celles vues en TP).

# Règles du jeu

Le Mémo est un jeu de cartes qui se joue habituellement seul. Un ensemble de cartes ornées de dessins, identiques deux à deux, est mélangé et étalé faces cachées sur un plateau. À chaque tour, le joueur essaye de reconstituer une paire en retournant deux cartes du plateau. Quand une paire a été trouvée, elle est retirée du plateau. Quand il n’y a plus de cartes sur le plateau, le jeu est fini. Le but pour le joueur est de se souvenir des positions auxquelles des dessins ont déjà été rencontrés afin de reconstituer les paires plus facilement au fur et à mesure des retournements de cartes.

Ici il s’agit d’une version en console du jeu, donc les dessins seront remplacés par des caractères ASCII entre ‘a’ et ‘z’, entre ‘A’ et ‘Z’, et/ou entre ‘0’ et ‘9’.

***Les n-uplets*** : la norme pour ce genre de jeu, telle qu’elle a été décrite ci-dessus, est que les cartes identiques aillent **par paires**, mais on propose ici que le nombre de cartes identiques à retourner **puisse être différent de 2**, d’où le terme plus générique de “n-uplets”.

Ainsi, en début de partie, le jeu demandera à l’utilisateur combien de cartes identiques sont dans les n-uplets à reconstituer. Pour que le jeu ait un intérêt, ce nombre devrait être de 2 ou plus, mais pour l’exercice de style on tiendra aussi compte du cas où ce nombre est 1.

# Fonctionnalités

Le jeu attendu devra donc produire des parties selon le déroulement suivant :

* demander au joueur le nombre de cartes identiques formant les n-uplets (nbCartesParUplet)
* demander au joueur combien de n-uplets doivent être présents sur le plateau (nbUplets)
* construire le plateau de jeu comme une grille de forme nbCartesParUplet x nbUplets
* Tour : jusqu’à ce que le joueur retourne une carte différente de la précédente, ou que toutes les cartes d’un n-uplet aient été retournées, offrir au joueur de choisir une carte du plateau à retourner (en entrant ses coordonnées de ligne puis de colonne)
* réafficher le plateau après chacun de ces choix en montrant quelles cartes ont été retournées
* en cas d’échec (une carte différente de la précédente a été retournée) remettre les cartes du tour face cachée et commencer un nouveau tour
* en cas de réussite (n-uplet complet retourné) laisser les cartes visibles, les rendre inaccessibles pour les prochains tours, et commencer un nouveau tour
* quand toutes les cartes du plateau ont été retournées, annoncer la fin de partie.

Des fonctionnalités bonus pourront être implémentées **s’il vous reste du temps et dans l’ordre que vous préférez** :

* optimiser les dimensions du plateau pour le rapprocher autant que possible d’un carré (plutôt que nbCartesParUplet x nbUplets),
* ajouter une condition d’échec de la partie après un nombre donné de mouvements,
* englober dans une boucle principale pour pouvoir recommencer une partie,
* intervention de 2 joueurs (ou plus) tour à tour, avec un système de score comptabilisant les n-uplets retournés,
* sauvegarde d’une partie en cours pour la reprendre ultérieurement,
* … Si vous avez des idées, n’hésitez pas à les proposer !

# Contraintes techniques

La programmation est faite en C#, en mode console, sous Windows, dans les conditions habituelles des TPs, sans utilisation de bibliothèques de fonctions externes (autre que les bibliothèques standards du framework .NET évidemment). **La version 6 de .NET est la seule acceptée.**

**La programmation orientée objet est exclue** (à part les habituels appels à des méthodes systèmes utilisées en TP telles que *Console.WriteLine*),

**L’utilisation des collections est exclue** (listes, tables de hachages…), le traitement sera basé sur des manipulations de tableaux,

Le code source doit répondre aux règles de programmation et conventions de nommage suivantes :

* Indentation partout correcte.
* Commentaires pour expliquer les parties complexes et/ou importantes du code.
* Respect de la norme [camelCase](https://fr.wikipedia.org/wiki/Camel_case) (exemple : *exempleDeVariable*).
* Nommage des sous-programmes à l’aide d’un verbe à l’infinitif et en commençant par une majuscule (exemple : *CalculerRésultat*).

# Organisation

Le travail est effectué en binôme. Pour les groupes en nombre impair, un seul trinôme ou monôme sera autorisé, le nombre total de groupes étant minimisé (pas de monôme pour convenance personnelle). Les binômes intra-groupes sont à privilégier.

L’application fait obligatoirement l’objet d’une gestion des versions avec Git et son code source est partagé via la plateforme GitHub.

Chaque membre de l’équipe doit cliquer sur le lien suivant pour accepter l’invitation, créer ou rejoindre une équipe (dont le nom doit comprendre les noms de tous les membres) et obtenir accès à un dépôt privé dédié au projet dans l’organisation **ensc-iprog** : <https://classroom.github.com/a/JIvT6pZt>

Chaque membre de l’équipe doit ensuite cloner ce dépôt sur sa machine de travail.

# Livrables attendus

La date limite pour remettre l’ensemble des livrables est fixée au **vendredi 16 décembre 2022 avant minuit**.

Le **code source** sera publié sur GitHub, dans le dépôt du projet.



Afin de préserver la fragile santé mentale des correcteurs, il est **TRÈS IMPORTANT** que le projet soit aussi facile à tester que possible. Le code déposé sur GitHub doit permettre, après clonage, de lancer l'application sur un nouveau poste sans aucun problème. **Merci de faire cette vérification après votre dernier *commit*.** Tout projet présentant des difficultés d’installation sera violemment pénalisé.

Vous devez également rédiger un **document de justifications techniques** expliquant :

* les choix de modélisation des données ;
* la structuration du code (découpage en sous-programmes, dynamique d’exécution) ;
* l’organisation de l’équipe, la répartition des tâches dans le groupe et le planning de réalisation.

Ce document sera ajouté au format PDF à la racine du dépôt GitHub du projet.

La **matrice d’implication** située à la racine du dépôt devra également être complétée.

# Critères d’évaluation

Nous serons particulièrement attentifs aux points suivants :

* L’implémentation des fonctionnalités demandées
* La structuration du code
* La clarté du code (nom d'identifiants, commentaires...)
* La justification des choix techniques (format des données, algorithmes)
* Les plus-values et originalités de votre code (modes supplémentaires, fonctionnalités additionnelles...)
* **Le strict respect des consignes et des délais**