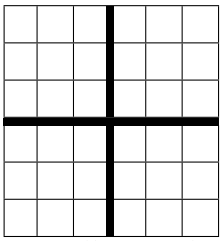
Projet en python : le Quintago

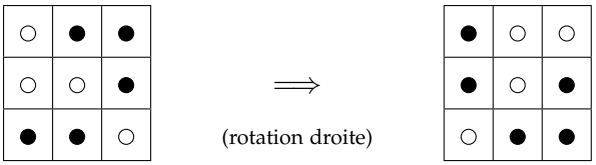
Le principe du jeu

Le Quintago (ou Pentago©) est un jeu se jouant au tour par tour par deux joueurs, et dont le principe est d'aligner des pions de sa couleur sur une grille de manière semblable à un morpion, mais en rajoutant un mouvement de rotation d'un des quadrants de la grille.

La grille. La grille est un tableau carré de dimensions 6×6. Elle est organisée en quatre sous-grilles de dimensions 3×3 comme détaillée dans la Figure 1.



La grille initiale.



Exemple de rotation vers la droite d'une sous-grille.

Figure 1 – Grille de jeu et exemple de rotation d'une sous-grille

Déroulement d'une partie.

Le tour de jeu d'un joueur se décompose en deux phases.

Phase pion : le joueur joue un pion de sa couleur sur n'importe quelle case libre de la grille.

Phase rotation : le joueur choisit une des quatre sous-grilles, sur laquelle il effectuera une unique rotation d'un quart de tour vers la gauche ou vers la droite.

Fin de la partie. Le premier joueur qui aligne 5 de ses pions horizontalement, verticalement ou en diagonale gagne la partie. Il se peut également que les deux joueurs alignent simultanément 5 de leurs pions. Dans ce dernier cas une déclaration d'égalité a lieu. Il y a également égalité dans le cas où la grille est complètement remplie sans qu'aucun des joueurs n'ait réussi à aligner 5 pions.

Travail demandé

Le projet consiste, en implantant les fonctions et procédures imposées dans les instructions ci-dessous et en respectant les différentes contraintes sur les structures de données, des fichiers, *etc*, à réaliser un programme `Python` permettant de jouer une partie de *Quintago* à deux joueurs humains selon les règles du jeu définies ci-dessus.

Instruction 0. Tester le jeu, en jouant « *à l'ancienne* » (avec un crayon et quatre morceaux carrés d'une feuille quadrillée), ou en ligne (par exemple à `https://perfect-pentago.net/` ou `https://pentago.vercel.app/`).

Instruction 1. Créer les différents fichiers `Python` nécessaires au jeu. Ceux-ci devront suivre un découpage cohérent tenant compte du rôle et du but des fonctions qui y seront codées, et devront contenir *a minima* la liste des modules figurant dans les contraintes.

Instruction 2. Implanter une procédure `afficher_grille` permettant d'afficher la grille.

Instruction 3. Implanter les fonctions ou procédures permettant d'effectuer un tour de jeu, en implantant les fonctions et procédures suivantes :

(i) la procédure `jouer_case` permettant de saisir une case valable, puis de la jouer
(ii) les procédures `rotation_droite` et `rotation_gauche` permettant d'effectuer les rotations des sous-grilles
(iii) la procédure `jouer_tour` pour effectuer un tour de jeu complet d'un joueur (case à jouer puis rotation)

Instruction 4. Implanter des fonctions de tests d'alignements, de paramètres la grille et le numéro d'un joueur et retournant un booléen si le joueur concerné a un alignement dans la grande grille *(indication : on pourra s'aider d'une variable auxiliaire en copiant la grille dans une unique grille à deux dimensions qui ne tient plus compte des séparations en sous-grilles)* :

(i) `alignement_horizontal` testant s'il y a un alignement horizontal
(ii) `alignement_vertical` testant s'il y a un alignement vertical
(iii) `alignement_diagonal` testant s'il y a un alignement diagonal

Instruction 5. Implanter la fonction `test_victoire`, de paramètre la grille de jeu et retournant 0 si aucun joueur n'est victorieux, -1 si les deux joueurs sont tous deux victorieux (*i.e.* ont chacun un alignement gagnant), et le numéro du joueur victorieux dans le cas où il n'en y a qu'un seul.

Instruction 6. Implanter la procédure/fonction `jouer` contenant la boucle principale permettant de jouer une partie à deux joueurs humains.

Instruction 7. Implanter les procédures/fonctions `sauvegarder_grille` et `charger_grille` permettant de sauvegarder une grille dans un fichier externe `partie_en_cours.txt`, et de le charger pour reprendre la partie en cours.

De plus, les contraintes suivantes devront impérativement être respectées.

Contraintes sur les structures de données. Chaque sous-grille est implantée en `Python` par une liste à deux dimensions, et la grille complète est une liste à deux dimensions, dont les éléments sont les sous-grilles (la grille de jeu est donc une liste à 4 dimensions).

Contraintes sur la structures des fichiers. Les fichiers suivant devront obligatoirement figurer dans le projet (il n'est pas interdit de créer d'autres modules) :

- `quintago.py` : fichier principal contenant la fonction de lancement du jeu ; tous les autres fichiers devront y être importés
- `parametres.py` : fichier contenant tous les paramètres de la simulation, implantés sous forme de variables (interprétées comme des constantes)
- `initialisation.py` : fichier contenant la ou les fonctions nécessaires à la création d'une configuration initiale
- `affichage.py` : fichier contenant la ou les fonctions nécessaires à l'affichage d'une configuration
- `saisie.py` : fichier contenant la ou les fonctions de saisies
- `tests_alignements.py` : fichier contenant les fonctions nécessaires aux tests de victoires d'un joueur
- `tour_de_jeu.py` : fichier contenant la ou les fonctions nécessaires à l'exécution d'un tour de jeu (choix d'une case et d'une rotation, et application de la rotation)

On rappelle que l'import de la totalité des fonctions dans un autre module se fait par l'instruction :
```
from module_a_importer import *
```

Contrainte sur la saisie. Il est de plus demandé de gérer la *protection de la saisie au clavier* : le programme ne devra pas s'interrompre si le joueur ne saisit pas les bons types de données demandés. Par exemple, si on saisit « `!!?3@nimp&#%` » au lieu d'un entier, le programme ne devra pas s'arrêter avec un message d'erreur mais demander de ressaisir une entrée valide.

Contraintes sur la documentation. Le projet et les fichiers devront être commentés de manière complète et pertinente, avec notamment :

- des « *docstrings* », *i.e.* des commentaires en en-tête des fonctions avec leurs spécifications (avec `""" commentaires sur plusieurs lignes """`)

- des commentaires de code (avec `#`) pour expliquer les portions de code ou condition subtiles
- un fichier `README`, dont le rôle est de donner les principales informations sur le projet, et devront y figurer *a minima* les informations suivantes :
  - auteurs
  - nom du projet
  - date et version du programme
  - langage utilisé et version (par exemple, `Python3.6` et versions ultérieures)
  - fichier principal à exécuter pour jouer
  - le cas échéant, description du menu d'accueil et des options
  - le cas échéant, arborescence du projet
  - liste et rôles des paramètres
  - liste des fichiers / modules implantés
  - liste des autres modules `Python` utilisés

Consignes et évaluation

Le code source de votre projet est à envoyer par mail intitulé `[NSI] Prénom 1 Prénom 2 - projet quintago` à l'adresse `senotprof@gmail.com` **avant le lundi 13/03/2023 à 23h59**. Le code source sera joint au mail dans un format de dossier compressé de type `zip`, ou d'archive compressée au format `.tar.gz` (aucun format propriétaire, type `.rar`, ne sera accepté), le dossier devant être nommé au format `prenom_1_prenom_2` (sans accent ni majuscule). Aucun délai supplémentaire ne sera accordé, la date butoir est immuable, et tout retard sera pénalisé : il convient donc de commencer le projet au plus tôt.

Le projet est **à coder obligatoirement en binôme** (sauf dérogation pour un unique trinôme, si l'effectif de classe est impair). La note sera commune au binôme, sauf cas de travail déséquilibré flagrant.
Il n'est pas interdit (et c'est même conseillé) de discuter du projet entre les différents groupes et de s'échanger des idées ou de demander un peu d'aide (à commencer par celle de votre enseignant), mais l'intégralité du code devra être produite par les membres du groupe. Des questions individuelles pourront être posées durant la période de réalisation du projet et/ou après sa remise, et le but étant de s'assurer que le programme a bien été conçu et implanté en `Python` par les membres du groupe.
Tout plagiat fera l'objet de sanctions avec une tolérance strictement négative (une fonction ou une portion de code bêtement recopiée ne sera pas utile dans un autre contexte que le programme initial, et surtout ne sera ni comprise, ni assimilée, et sera, en plus des risques de sanctions pour fraude et plagiat, une perte de temps pour tout le monde, élèves plagiés, plagieurs et enseignant).

L'évaluation prendra en compte :

- les erreurs de syntaxe : les fichiers `.py` doivent être directement interprétés sans erreur par la console `Python`
- l'**originalité du code** (dans le sens de « code personnel et individuel »)
- la propreté et la lisibilité du code ainsi que tout ce qui facilitera sa compréhension par le correcteur (commentaires du code, docstrings, noms des variables, commentaires du code, noms des fonctions, commentaires du code, « aération » du code, commentaires du code, *etc*)
- l'implantation des fonctions imposées et le **respect des consignes** (règles du jeu, cahier des charges, consignes sur le code, les structures de données et les spécifications des fonctions, mais aussi les consignes annexes, comme l'intitulé du mail et le format de fichier, ainsi que bien évidemment le respect des délais)
- la correction et la performance du code
- la facilité d'utilisation et la jouabilité
- la paramétrabilité et possibilité de généralisation du jeu (choix des symboles de jeu, taille des sous-grilles et de la grille, longueur de l'alignement gagnant, *etc*)
- le(s) éventuel(s) bonus implanté(s)

Travail optionnel proposé

[que dalle en points]

Si le temps le permet, il est possible, pour se familiariser avec `Python` et la conduite d'un projet, d'apporter diverses améliorations au programme, comme par exemple :

- suivre l'adage des programmeurs : « *less is more* », et simplifier au maximum le code (tout en respectant les consignes)
- améliorer la présentation et l'interface du jeu

- ajouter des options : par exemple proposer de rejouer en fin de partie, compter les parties gagnées par les joueurs, identifier les joueurs par leurs prénoms, *etc*
- adapter le jeu à des grilles de tailles quelconques (soit avec des sous-grilles de taille $n \times n$ soit avec une grille constituée de 9, 16… sous-grilles 3×3), et définir une longueur d'alignement gagnant adaptée aux nouvelles grilles
- faire un mode 1 joueur en programmant une ou plusieurs intelligences artificielles pour permettre à l'ordinateur de jouer de manière autonome en suivant une stratégie prédéfinie

L'éventuelle implantation d'un bonus ne sera prise en compte que si tout le reste est fonctionnel et conforme au cahier des charges (et même dans ce cas, une fraction de point bonus n'est même pas sûre d'être attribuée).

Annexe : un exemple

Un exemple d'interface et d'une portion du déroulement d'une partie est donné ici. Cet exemple a pour vocation à donner des idées et servir d'inspiration, et non à être recopié à l'identique*.

```
Tour du joueur 1

        1 2 3   4 5 6
   1  ○ . ●   . . ●
   2  . ○ .   . ● .
   3  . ● ○   ○ . .
   4  . ● ○   ○ . .
   5  . ● .   ○ ● .
   6  . . .   . . .
Jouer un pion en case :
Choisir la ligne : 6
Choisir la colonne : 2

        1 2 3   4 5 6
   1  ○ . ●   . . ●
   2  . ○ .   . ● .
   3  . ● ○   ○ . .
   4  . ● ○   ○ . .
   5  . ● .   ○ ● .
   6  . ● .   . . .
Tourner la sous-grille numéro :
-----------
|  1 |  2 |
-----------
|  3 |  4 |
-----------
Choisir le numéro de la sous-grille à tourner : 3
Choisir le sens de rotation (g/d) : d

Tour du joueur 2

        1 2 3   4 5 6
   1  ○ . ●   . . ●
   2  . ○ .   . ● .
   3  . ● ○   ○ . .
   4  . . .   ○ . .
   5  ● ● ●   ○ ● .
   6  . . ○   . . .
Jouer un pion en case :
Choisir la ligne :
```

*. Ça, c'est mon projet© et mon interface de jeu ©. Trouvez votre style.