

# 1 Le cahier des charges

1. Les groupes de 2 ou 3 sont imposés.
2. Le choix des sujets est contraint :
  - une liste de sujets est proposée, tout sujet en dehors de la liste doit être validée par le professeur qui peut le refuser;
  - deux groupes ne peuvent choisir le même sujet;
  - les sujets faciles et guidés de la liste sont réservés aux élèves les moins à l'aise;
  - le choix d'un sujet difficile sera valorisé.
3. Le rendu doit s'effectuer sous deux formes, au plus tard le **jeudi 15/05** :
  - Un **rendu collectif** (un seul par groupe de projet) :
    - **Sur 6 points** : dont 2 points évaluant la difficulté du sujet le code du projet et tous les fichiers ressources dans une archive zip nommée projet-final-eleve1-eleve2-eleve3.zip.
  - Un **rendu individuel** :
    - **Sur 4 points** : sous la forme d'une capsule vidéo de 4 minutes minimum à 8 minutes maximum au format mkv ou mp4. Vous pouvez utiliser Vokoscreen : <https://linuxecke.volkoh.de/vokoscreen/vokoscreen-download.html>.  
La capsule doit être structurée en parties :
      - \* Une introduction avec présentation des objectifs du projet.
      - \* Une partie sur l'organisation du travail au sein du groupe (répartition des tâches, planification)
      - \* Une partie expliquant un extrait du code
      - \* Une partie libre sur un thème lié au sujet : histoire ou culture informatique
      - \* Une conclusion sur ce que vous avez appris ou développé comme compétences dans ce projet.

# 2 Quelques règles à respecter

- **Règle n°1** Avant de coder je réfléchis crayon en main à l'architecture de mon programme et j'essaie de bien distinguer les données des fonctionnalités..
- **Règle n°2** J'applique le principe de programmation modulaire :« *Chaque fois qu'une tâche peut être clairement séparée dans un programme, je l'encapsule dans une fonction et chaque fonction peut être écrite par une personne distincte.* »..
- **Règle n°3** J'écris des tests unitaires pour chaque fonction de mon programme.
- **Règle n°4** Je prête attention à la lisibilité de mon programme : noms de variables et de fonctions explicites, autodocumentation par une docstring pour chaque fonction et commentaires pertinents des points les moins lisibles.
- **Règle n°5** Je prête attention à la portée des variables et j'essaie d'éviter l'usage de variables globales dans les fonctions.

- **Règle n°6** Si possible j'inclus un petit code client dans mon programme afin que le professeur puisse facilement le tester.
- **Règle n°7** Je prête attention à la portabilité de mon programme : j'utilise un encodage en UTF-8 et j'insère la directive `# -*- coding: utf-8 -*-` au début de mon script.
- **Règle n°8** Je ne perds jamais de vue mes objectifs, je fais un point régulier avec les membres de mon trinôme et les professeurs, j'utilise des outils de travail collaboratif.

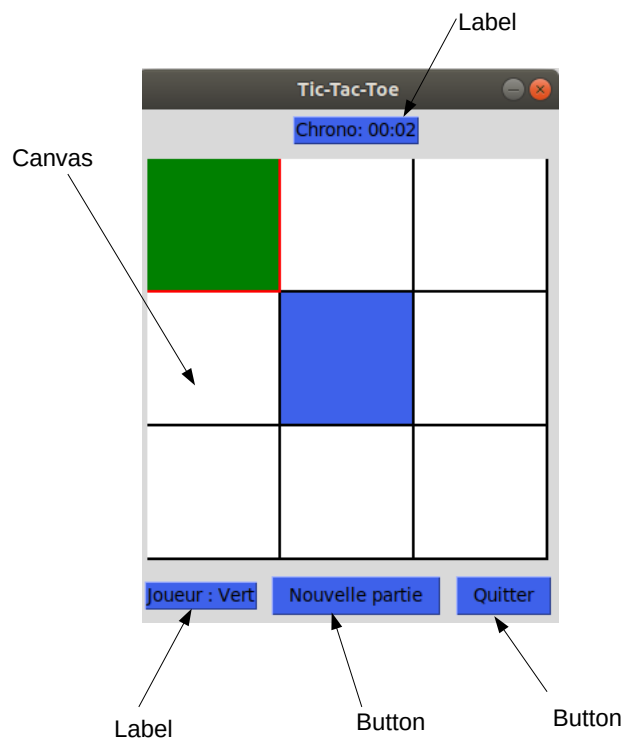
## 3 Liste de projets

### 3.1 Quatre Sujets guidés

Une archive [ressources-pf-2025.zip](#) est fournie avec les ressources (images et fichiers textes) nécessaires pour traiter ces sujets guidés.

#### 3.1.1 Sujet 1 : simulation avec interface graphique d'un jeu de morpion (*moyen*)

L'objectif de ce mini projet est de programmer un jeu de Tic-Tac-Toe ou Morpion avec une opposition de deux joueurs humains. L'application devra se présenter sous la forme d'une interface graphique similaire à celle présentée ci-dessous :



Le projet guidé s'appuie sur le document et le squelette de code disponibles dans le dossier morpion de [ressources.zip](#).

#### 3.1.2 Sujet 2 : simulation en console d'un jeu de 2048 (*moyen*)

Le 2048 est un jeu vidéo développé par Gabriele Cirulli en 2014. Le principe inspiré de celui du taquin consiste à faire glisser des tuiles numériques sur une grille carré de  $4 \times 4$  cases jusqu'à obtenir le nombre 2048. Par

exemple si on déplace les tuiles vers la droite, elles glissent toutes horizontalement si des cases vides le permettent et deux cases adjacentes sur la même ligne et portant le même nombre se combinent en une seule case portant le double du nombre. Si on déplace les tuiles vers la gauche ou verticalement vers le bas ou vers le haut, on peut de même combiner des cases sur une même ligne ou une même colonne. Avant chaque mouvement un 2 ou un 4 apparaît aléatoirement sur l'une des cases vides.

L'objectif de ce projet est de développer un jeu de 2048 qui s'exécute dans la console Python. On donne ci-dessous une capture d'écran d'un début de partie. La grille est affichée avant chaque mouvement avec le 2 ou le 4 apparu aléatoirement dans l'une des cases vides après le mouvement précédent.

```
In [*]: 1 jeu_2048()

| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 4 |
| 0 | 0 | 0 | 0 |
-----
Déplacement 4 (gauche) 6 (droite) 8 (haut) 9 (bas) ? 4
| 0 | 0 | 0 | 0 |
| 0 | 4 | 0 | 0 |
| 4 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
-----
Déplacement 4 (gauche) 6 (droite) 8 (haut) 9 (bas) ? 9
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 2 | 0 | 0 |
| 4 | 4 | 0 | 0 |
-----
Déplacement 4 (gauche) 6 (droite) 8 (haut) 9 (bas) ? 4
| 0 | 0 | 0 | 0 |
| 0 | 0 | 2 | 0 |
| 2 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 |
-----
Déplacement 4 (gauche) 6 (droite) 8 (haut) 9 (bas) ? 
```

On s'appuiera sur l'énoncé fourni dans le document DM-2048V1 . pdf inclus dans [ressources.zip](#).

### 3.1.3 Sujet 3 : simulateur et solveur du jeu Sutom (*difficile*)

L'objectif de ce projet est de réaliser un simulateur du jeu sutom version française du jeu Wordle lui-même inspiré du jeu télévisé français motus.

Vous pouvez suivre les indications et compléter les squelettes de code fournis dans [ressources-pf-2025.zip](#) .

### 3.1.4 Sujet 4 : traitement d'images au format PGM/PBM/PPM (*facile*)

Les formats d'images PGM/PBM/PPM sont des formats d'images qui sont des fichiers textes où chaque pixel est décrit par une valeur. L'en-tête du fichier fixe des paramètres comme la largeur, la hauteur et le type pixel : noir et blanc, en niveaux de gris ou couleur (R,G,B).

Traiter les questions du document TraitementImage . pdf inclus dans [ressources-pf-2025.zip](#).

## 3.2 Sujets non guidés

- Chiffrer comme Enigma ⇒ Facile

Vous devez résoudre ce défi Codin Game :

<https://www.codingame.com/training/easy/encryptiondecryption-of-enigma-machine>

Dans ce projet vous développerez vos compétences en manipulation de chaînes de caractères et découvrirez le fonctionnement de la machine Enigma, utilisée pendant la seconde guerre mondiale par les nazis pour chiffrer leurs messages.

- **Valideur de Sudoku**  $\Rightarrow$  Facile

Vous devez résoudre ce défi Codin Game : <https://www.codingame.com/training/easy/sudoku-validation>.

*Un joueur vous remet une grille de sudoku et vous devez vérifier si elle a été correctement remplie.*

*Une grille de sudoku se compose de  $9 \times 9 = 81$  cases divisées en 9 sous-grilles de  $3 \times 3 = 9$  cases. Pour que la grille soit correcte, chaque ligne doit contenir une occurrence de chaque chiffre (1 à 9), chaque colonne doit contenir une occurrence de chaque chiffre (1 à 9) et chaque sous-grille doit contenir une occurrence de chaque chiffre (1 à 9).*

*Vous devez répondre par vrai si la grille est correcte ou par faux si elle ne l'est pas.*

- **Décodeur de code préfixe**  $\Rightarrow$  Facile

Vous devez résoudre ce défi Codin Game : <https://www.codingame.com/training/easy/prefix-code>.

*Étant donné un ensemble fixe de caractères, un code est une table qui indique le codage à utiliser pour chaque caractère.*

*Un code préfixe est un code qui possède la propriété de préfixe, c'est-à-dire qu'il n'existe aucun caractère dont le codage est un préfixe (segment initial) du codage d'un autre caractère.*

*Votre objectif est de décoder une chaîne encodée en utilisant le code préfixe donné, ou de dire que ce n'est pas possible.*

*Exemple de codage. Étant donné la chaîne « abracadabra » et le code préfixe :*

a -> 1  
b -> 001  
c -> 011  
d -> 010  
r -> 000

*L'encodage résultant est : 10010001011101010010001*

*Ainsi, si l'on vous donne le code ci-dessus et l'entrée 10010001011101010010001, vous devriez produire la chaîne « abracadabra ».*

*Avec le même code préfixe, si l'entrée est 0000, alors vous devriez dire qu'il y a une erreur à l'index 3. En effet, les trois premiers caractères de cette entrée peuvent être décodés pour donner un « r », mais il reste le 0, qui ne peut pas être décodé.*

- **L'Awalé un jeu africain**  $\Rightarrow$  Facile

Vous devez résoudre ce défi Codin Game : <https://www.codingame.com/training/easy/simple-awale>.

*L'Awalé est un jeu africain pour 2 joueurs qui consiste à déplacer des grains dans des bols. Chaque joueur dispose de 7 bols indexés de 0 à 6. Le dernier bol est la réserve.*

*A chaque tour, un joueur choisit un de ses bols à l'exception de la réserve, y ramasse tous les grains et les redistribue un par un dans les bols commençant juste après celui choisi. Si le*

*nombre de grains en main est suffisant, après avoir ajouté un grain à sa réserve, le joueur continue la distribution dans les bols de l'adversaire, à l'exclusion de sa réserve, puis dans ses propres bols, jusqu'à ce que sa main soit vide.*

*Si le dernier grain est distribué dans la réserve du joueur, celui-ci est autorisé à rejouer.*

*Exemples :*

```
nombre de bols : 0 1 2 3 4 5 6
-----
bols opp : 5 1 0 6 2 2 [3]
mes bols : 3 4 0 3 3 2 [2]
```

Je joue le bol 0 : distribuer 3 grains dans les bols 1, 2 et 3

```
nombre de bols : 0 1 2 3 4 5 6
-----
bols opp : 5 1 0 6 2 2 [3]
mes bols : 0 5 1 4 3 2 [2]
```

Je joue le bol 5 : je distribue 2 grains (1 dans ma réserve  
et 1 dans le premier bol adverse)

```
nombre de bols : 0 1 2 3 4 5 6
-----
bols adverses : 6 1 0 6 2 2 [3]
mes boules : 3 4 0 3 3 0 [3]
```

Si je termine dans ma réserve, je peux rejouer :  
Je joue la boule 3 :

```
nombre de bols : 0 1 2 3 4 5 6
-----
boule opp : 5 1 0 6 2 2 [3]
mes boules : 3 4 0 0 4 3 [3]
```

On rejoue

*Votre but est de simuler votre tour de jeu. Etant donné les nombres de grains dans chaque bol et le numéro du bol choisi, votre programme doit afficher la nouvelle situation et la chaîne REPLAY si la partie a été jouée.*

- **Fix the network** ⇒ Facile

Vous devez résoudre ce défi Codin Game :

<https://www.codingame.com/training/easy/fix-the-networks> qui porte sur les chaînes de caractère, le codage binaire des entiers sur 8 bits, et les **adresses IPV4 au format CIDR**.



- sp = space
- bS = backSlash \
- sQ = singleQuote '
- nl = NewLine

- **Le problème du voyageur de commerce**  $\Rightarrow$  Moyen / Facile

Vous devez résoudre ce défi Codin Game à l'aide d'un algorithme glouton : <https://www.codingame.com/ide/puzzle/the-travelling-salesman-problem>.

*Le problème du voyageur de commerce (TSP) pose la question suivante : « Étant donné une liste de villes et les distances entre chaque paire de villes, quel est l'itinéraire le plus court possible qui visite chaque ville exactement une fois et retourne à la ville d'origine ? »*

*Dans cette énigme, la réponse n'est pas nécessairement le chemin le plus court, mais une approximation à l'aide d'un algorithme gourmand (qui, en fait, pourrait aussi être le chemin le plus court).*

*Cet algorithme gourmand part de la première entrée donnée et choisit toujours le point le plus proche du point actuel. Il continue ainsi jusqu'à ce qu'il ne reste plus aucun point et que le dernier point soit relié au premier.*

*Utilisez la distance euclidienne, c'est-à-dire  $\sqrt{\Delta x^2 + \Delta y^2}$ , comme distance entre deux villes. Si plusieurs points ont la même distance, choisissez toujours celui qui apparaît en premier dans la liste.*

*En général, l'algorithme glouton ne trouve pas la solution optimale, mais une heuristique gourmande peut néanmoins produire des solutions localement optimales qui se rapprochent d'une solution optimale globale en un temps raisonnable.*

- **Asteroids**  $\Rightarrow$  Moyen / Facile

Vous devez résoudre ce défi Codin Game : <https://www.codingame.com/ide/puzzle/asteroids>.

*Vous avez été chargé d'étudier une région de l'espace afin de détecter des astéroïdes potentiellement dangereux. On vous donne deux images du ciel nocturne de dimensions  $W \times H$ , prises à deux moments différents  $t_1$  et  $t_2$ . Pour votre commodité, les astéroïdes ont été marqués par les lettres majuscules A à Z, le reste étant de l'espace vide représenté par un point (.) . En utilisant les informations contenues dans ces deux images, déterminez la position des astéroïdes à  $t_3$ , et produisez une image de la même région du ciel.*

*Si nécessaire, les coordonnées finales doivent être arrondies à la baisse (partie entière `floor` à importer du module `math`). Les astéroïdes voyagent à des altitudes différentes (A étant le plus proche et Z le plus éloigné de votre point d'observation) et ne peuvent donc pas entrer en collision les uns avec les autres pendant leur transit. Si deux astéroïdes ou plus ont les mêmes coordonnées finales, seul l'astéroïde le plus proche sera pris en compte. Il est garanti que tous les astéroïdes à  $t_1$  seront encore présents à  $t_2$ , qu'aucun astéroïde n'est caché dans les images données, et qu'il n'y a qu'un seul astéroïde par altitude.*

*NB : En raison de l'opération `floor`, il est important de choisir un système de coordonnées dont l'origine se trouve dans le coin supérieur gauche et dont l'axe des  $y$  augmente dans le sens descendant.*



- **Déchiffrer un message par Analyse fréquentielle** ⇒ *Moyen*

Il s'agit de résoudre le problème *Frequency-Based Decryption* sur la plateforme Codin Game : <https://www.codingame.com/ide/puzzle/frequency-based-decryption> :

*Alice travaille depuis plusieurs années dans un bureau de renseignement secret. Experte en cryptographie, elle est souvent chargée de décoder des messages interceptés.*

*Un jour, Alice reçoit un étrange message sur son bureau. Il s'agit d'une chaîne de caractères codée avec un code à décalage, mais aucune clé n'a été fournie pour la décoder. Alice sait qu'elle doit décoder le message rapidement pour découvrir les plans de l'ennemi.*

*Heureusement, Alice dispose d'une méthode fiable pour décoder les messages sans clé. Elle utilise l'analyse de fréquence des lettres pour déterminer le décalage utilisé dans le processus d'encodage.*

*Cependant, Alice sait que le décalage n'est appliqué qu'aux caractères alphabétiques. Les caractères non alphabétiques, tels que les chiffres et les symboles, ne sont pas affectés par le décalage.*

*En comparant la fréquence des lettres du message codé avec celles de la langue anglaise, Alice finit par trouver le décalage utilisé dans le processus de codage.*

*En outre, Alice souhaite préserver les caractéristiques du texte original, y compris les majuscules. Cela signifie que le programme doit conserver la casse du texte original lorsqu'il génère le message décodé en texte clair.*

*Votre tâche consiste maintenant à aider Alice en écrivant un programme qui prend la chaîne codée en entrée, utilise l'analyse de la fréquence des lettres pour décoder le message en appliquant le décalage uniquement aux caractères alphabétiques, et génère le message décodé en texte clair tout en préservant la casse du texte original.*

- **Jeu pixel art : pong, snake, angry birds, space invaders ...** ⇒ *Moyen à difficile*

Écrire une version Python du jeu Pong (ou Angry Birds, Space Invaders, Snake, Tetris ...) On utilisera le module **pyxel** de Python.

- **Étude de la percolation dans une grille** ⇒ *Moyen*

La **percolation** désigne le passage d'un fluide dans une grille de sites qui peuvent être ouverts (perméables au fluide) ou fermés (imperméables au fluide).

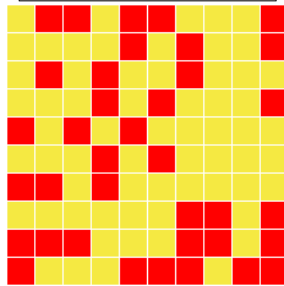
On modélise l'ensemble des sites par une grille de cases ouvertes ou fermées et on s'intéresse au passage d'un fluide du bord supérieur vers le bord inférieur. Lorsque le fluide remplit une case, il peut s'écouler vers l'une des 4 voisines en croix (si elle existe) : case au dessus, en dessous, à gauche ou à droite. S'il existe un chemin d'un site ouvert du bord supérieur de la grille vers un site ouvert du bord inférieur, on dit que la grille percole.

Dans les graphiques ci-dessous, les cases jaunes sont ouvertes, les rouges sont fermées et les bleues sont remplies par le fluide.

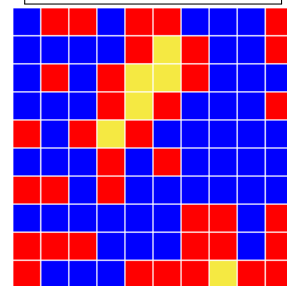
- Une grille qui percole :



Avant écoulement

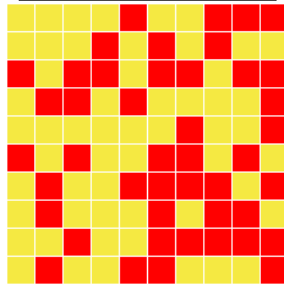


Après écoulement

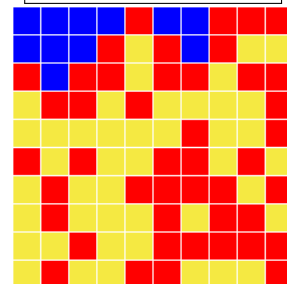


### – Une grille qui ne percole pas :

Avant écoulement



Après écoulement



On commencera par résoudre ce problème <https://e-nsi.gitlab.io/pratique/N2/666-percolation/sujet/>.

Ensuite avec le module `ipynbblocks` dans un notebook ou `pyxel` dans un script Python, réaliser une petite application graphique qui simule un écoulement de fluide depuis le bord supérieure d'une grille remplie aléatoirement de sites ouverts avec une probabilité  $p$  paramétrable.

On pourra capturer l'écoulement dans un gif animé à l'aide du module `imageio`. Lien vers un tutoriel :

<https://www.tutorialsexample.com/python-create-gif-with-images-using-imageio-a-complete-guide-python-tutorial/>

En complément on pourra étudier le seuil de percolation sur une grille  $n \times n$  en mesurant pour chaque proportion  $p$  de sites ouverts, la probabilité que la grille percole à l'aide d'une **méthode de Monte-Carlo** : on fixe  $p$  et sur un échantillon de 100 grilles aléatoires on mesure la fréquence de grilles qui percolent.

### • Nombre de Bacon $\Rightarrow$ Moyen

Vous devez résoudre ce défi Codin Game à l'aide d'un parcours de graphe en largeur : <https://www.codingame.com/ide/puzzle/six-degrees-of-kevin-bacon>. Avec ce projet vous découvrirez une nouvelle structure de données pour modéliser des relations entre objets : les graphes et un algorithme qui permet de déterminer le nombre minimal de sauts dans un graphe. Vous pourrez vous appuyer sur **mon cours de TNSI**.

Pour comprendre le fonctionnement d'un parcours en largeur vous pouvez résoudre d'abord ce défi <https://www.codingame.com/ide/puzzle/moves-in-maze>.

### • ASCII-ART $\Rightarrow$ Moyen

Vous devez résoudre ce défi Codin Game :

<https://www.codingame.com/training/easy/ascii-art-the-drunken-bishop-algorithm>

Dans ce projet vous devrez créer un ASCII-ART en appliquant un algorithme. Vous développerez vos compétences en manipulation de chaînes de caractères et codage.

- **Vérification de la validité d'un numéro de carte**  $\Rightarrow$  *Moyen*

L'algorithme de Luhn, ou code de Luhn, ou encore formule de Luhn est aussi connu comme l'algorithme « modulo 10 » ou « mod 10 ». C'est une formule de somme de contrôle utilisée pour valider une variété de numéros de comptes, comme les numéros de cartes bancaires, les numéros d'assurance sociale canadiens, les numéros IMEI des téléphones mobiles ainsi que pour le calcul de validité d'un numéro SIRET. Vous devez résoudre ce défi Codin Game : <https://www.codingame.com/training/easy/credit-card-verifier-luhns-algorithm> et proposer une interface graphique minimaliste permettant la saisie et la vérification d'un numéro avec cet algorithme.

- **Sortir d'un labyrinthe**  $\Rightarrow$  *Moyen*

Vous devez résoudre ce défi Codin Game à l'aide d'un parcours de graphe en profondeur : <https://www.codingame.com/ide/puzzle/maze>. Avec ce projet vous découvrirez une nouvelle structure de données pour modéliser des relations entre objets : les graphes et un algorithme qui permet d'atteindre la sortie dans un labyrinthe. Vous pourrez vous appuyer sur [mon cours de TNSI](#).

- **Chemin le plus court dans une grille**  $\Rightarrow$  *Moyen*

Vous devez résoudre ce défi Codin Game à l'aide d'un parcours de graphe en largeur : <https://www.codingame.com/ide/puzzle/the-lost-child-episode-1>. Avec ce projet vous découvrirez une nouvelle structure de données pour modéliser des relations entre objets : les graphes et un algorithme qui permet de déterminer le nombre minimal de sauts dans un graphe. Vous pourrez vous appuyer sur [mon cours de TNSI](#).

- **Convertisseur binaire/décimal**  $\Rightarrow$  *Moyen*

En utilisant le module graphique `nsi_ui` décrit dans le manuel Hachette (pages 53 et 54) et fourni dans [ressources-pf-2024.zip](#) ou le module `tkinter`, créer une interface graphique de conversion d'un entier de la base dix à la base deux. L'interface devra proposer la conversion pour les entiers non signés ou signés (complément à deux).

- **Jeu de Simon**  $\Rightarrow$  *Moyen*

- **Niveau 1 :** Programmer le [jeu de société Simon](#) en version texte. L'utilisateur joue contre le programme qui choisit de manière aléatoire une suite de caractères. Au début, le programme propose un caractère que le joueur doit reproduire. Ensuite, à chaque itération, un caractère supplémentaire est ajouté

*Cahier des charges :*

- \* Le programme choisit 10 caractères au hasard.
- \* À chaque tour N :
  - le programme affiche N caractères de la suite,
  - le joueur propose une reproduction de ces N caractères,
  - En cas d'erreur, le programme s'arrête et le score du joueur est affiché.
- \* Quatre niveaux de difficulté en fonction du temps laissé au joueur.
- **Niveau 2 :** Programmer le [jeu de société Simon](#) en version couleur et son. Utiliser le module `pyxel`.

- **Fractions égyptiennes**  $\Rightarrow$  *Moyen*

Une fraction  $\frac{p}{q}$  comprise entre 0 et 1 peut se décomposer en une somme de fractions de numérateur 1 et l'algorithme qui permet de construire une telle décomposition est un algorithme glouton déjà connu des Égyptiens! L'algorithme est décrit dans ce [sujet d'Olympiades](#). Vous devez implémenter cet algorithme et proposer une petite interface graphique où l'on peut saisir une fraction pour obtenir l'affichage de sa décomposition.

- **Jeu de Morpion et Min-Max**  $\Rightarrow$  *Moyen à Difficile*

Programmation d'un jeu de morpion avec ou sans interface graphique qui propose deux modes :

1. Joueur humain 1 contre joueur humain 2;
2. Joueur humain contre ordinateur qui choisit son coup avec **l'algorithme Min-Max**;

Ressource sur l'algorithme Min-Max : [https://www.fil.univ-lille1.fr/~L2S3API/CoursTP/Projets/minmax/algo\\_minmax.html](https://www.fil.univ-lille1.fr/~L2S3API/CoursTP/Projets/minmax/algo_minmax.html).

- **Réalisation d'une calculatrice en notation Polonaise inverse**  $\Rightarrow$  *Difficile*

Écrire un programme d'émulation de calculatrice qui effectue des calculs élémentaires (addition, soustraction, multiplication, division) en notation *postfixée* ou *polonaise inverse*. Proposer une petite interface graphique en le module **nsi\_ui** fourni dans le manuel Hachette (pages 53 et 54) ou le module **tkinter**. Commencer par résoudre ce problème : <https://www.codingame.com/ide/puzzle/reverse-polish-notation>.

- **Jeu de puissance 4**  $\Rightarrow$  *Difficile*

Réaliser une version du jeu de Puissance 4 pour 2 joueurs avec ou sans interface graphique. On pourra utiliser les modules **pyxel** ou **tkinter** de Python.

Prolongement possible : un joueur joue contre l'ordinateur.

- **Site web de quizz sur le programme de première NSI**  $\Rightarrow$  *Difficile*

Réaliser avec le module **Flask** un mini-site web permettant de saisir un thème du programme de Première NSI et générant un quizz aléatoires de questions choisies dans un fichier csv portant sur ce thème. Les scores pourraient être enregistrés dans un fichier.

Voir ce tutoriel sur Flask :

<https://frederic-junier.gitlab.io/parc-nsi/chapitre230/php-flask-git/>.

- **Sudoku**  $\Rightarrow$  *Difficile*

Réaliser un solveur de Sudoku par backtracking, avec ou sans interface graphique. Source : le problème 1 de [https://www.apmep.fr/IMG/pdf/CAPEs\\_avril\\_2017\\_epreuve\\_info.pdf](https://www.apmep.fr/IMG/pdf/CAPEs_avril_2017_epreuve_info.pdf).

- **Seam carving : redimensionnement intelligent d'image**  $\Rightarrow$  *Difficile*

Écrire un programme avec interface textuelle qui ouvre un fichier image au format **PGM** et redimensionne sa largeur en utilisant l'algorithme décrit dans le puzzle <https://www.codingame.com/ide/puzzle/seam-carving>. Les quatre premiers tests du puzzle sont fournis dans [ressources.zip](#).

- **Othello et Min-Max** ⇒ *Difficile à très difficile*

Programmation d'un jeu d'**Othello** /**Reversi** avec ou sans interface graphique qui propose deux modes :

1. Joueur humain 1 contre joueur humain 2;
2. Joueur humain contre ordinateur qui choisit son coup avec **l'algorithme Min-Max**;

Ressource sur l'algorithme Min-Max : [https://www.fil.univ-lille1.fr/~L2S3API/CoursTP/Projets/minmax/algo\\_minmax.html](https://www.fil.univ-lille1.fr/~L2S3API/CoursTP/Projets/minmax/algo_minmax.html).

## 4 Quelques bibliothèques de Python qui pourraient vous être utiles

- 📖 Dessiner avec la tortue : **turtle**.
- 📖 Réalisation d'interfaces graphiques (suivre les liens) par ordre croissant de difficulté de prise en main, les liens pointent vers des tutoriels ou la documentation officielle :
  - le module **ipynbblocks** permet de manipuler très facilement des grilles de blocs colorés dans un notebook jupyter comme ceux utilisés sur le service Capytale de l'ENT;
  - le module **pyxel**.
  - le module **nsi\_ui** fourni dans le manuel Hachette (pages 53 et 54) et dans .
  - le module **tkinter**
- 📖 Traitement d'images : **PIL** ou son fork **Pillow** : <https://pypi.python.org/pypi/Pillow/>
- 📖 Réalisation de graphiques pour les mathématiques ou la physique (possibilité d'animation) : **matplotlib** : <http://matplotlib.org/>
- 📖 Traitement avancé de données : <https://pandas.pydata.org/>
- 📖 Collecte de données sur le Web :
  - récupérer les données avec **requests** :  
<https://realpython.com/python-requests/>
  - « parser » du HTML avec **beautiful soup** :  
<https://realpython.com/beautiful-soup-web-scraper-python/>.

## 5 Quelques outils pour la gestion de projet ou le travail collaboratif

- 📖 Cours et ressources en ligne sur <https://frederic-junier.gitlab.io/parc-nsi/projets/>.
- 📖 Partage de fichiers, pad, articles de blog, notebooks Python dans **l'ENT**.
- 📖 Partage de scripts Python et test en ligne : <https://console.basthon.fr/>. ou le service Capytale de l'ENT.
- 📖 Documents textuels collaboratifs (avec possibilité de chat) : **Framapad** accessible depuis <https://framapad.org/>.
- 📖 Feuilles de calculs collaboratives : **Framacalc** accessible depuis <https://framacalc.org/>.
- 📖 Réalisation de carte mentale : **Framindmap** accessible depuis <https://framindmap.org/c/maps/>.