Projet guidé – SUTOM

---

1. Préambule

Objectif général

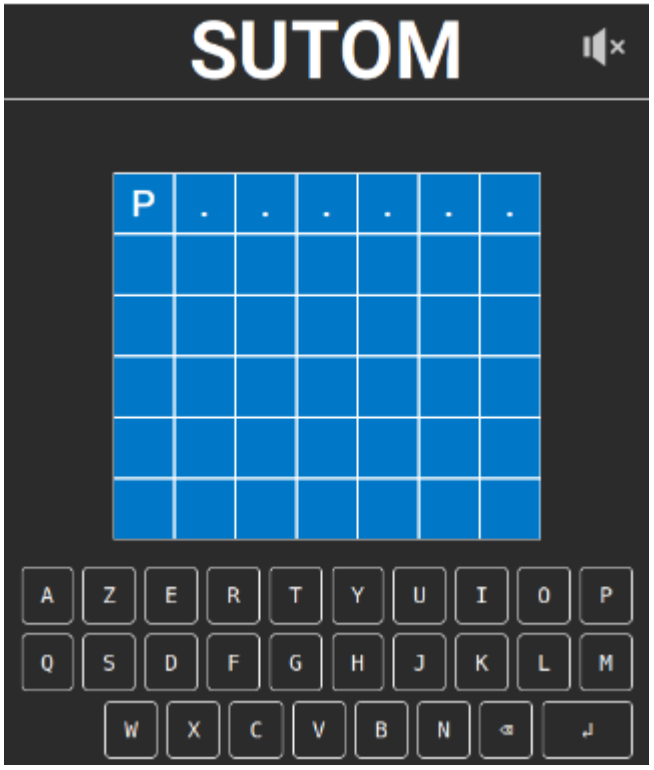
L'objectif de ce projet est de réaliser un **simulateur du jeu SUTOM**, version française du jeu **Wordle**, lui-même inspiré du jeu télévisé **Motus**.

Le projet se décline en plusieurs parties :

- une **bibliothèque de fonctions**,
- une **interface textuelle**,
- une **interface graphique**,
- un **solveur automatique** et un **assistant de résolution**.

Présentation du jeu

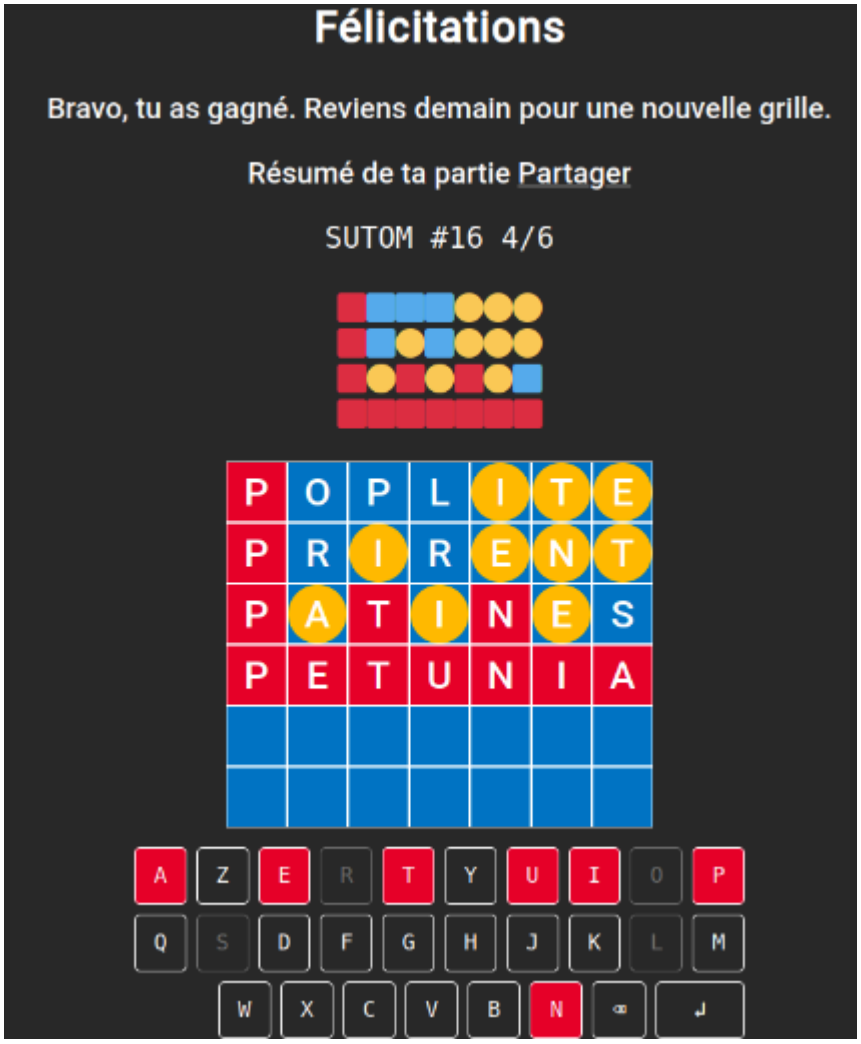
L'interface du jeu SUTOM se présente sous la forme d'une grille permettant de saisir des mots.



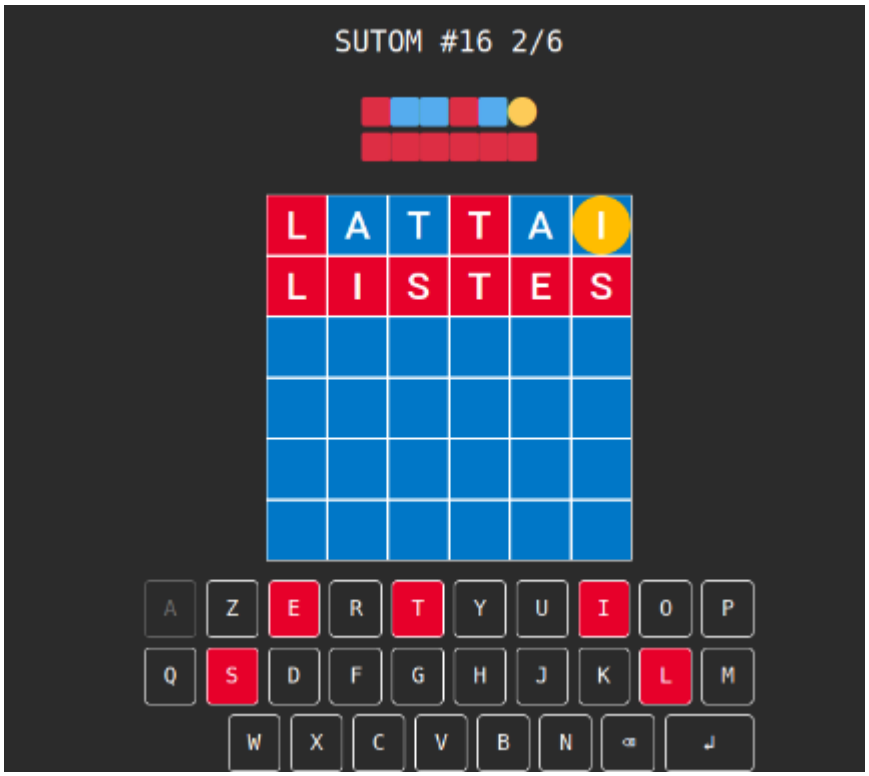
Règles du jeu

- Le joueur dispose de **6 essais** pour deviner le mot secret.

- Le mot secret :
    - contient entre **6 et 9 lettres**,
    - est **identique pour tous les joueurs le même jour**.
- Chaque proposition :
    - doit commencer par la **même lettre** que le mot secret,
    - doit appartenir au **dictionnaire fourni**.
- Codage des lettres :
    - 🟥 **Lettre bien placée**
    - 🟡 **Lettre présente mais mal placée**
    - 🟦 **Lettre absente du mot**



On donne un autre exemple avec la lettre ‘T’ qui n’apparaît qu’une seule fois dans le mot secret, on considère que le ‘T’ surnuméraire n’appartient pas au mot secret.



Convention simplifiée utilisée dans le projet

Pour les interfaces pédagogiques développées dans ce projet :

Situation	Symbole
Lettre absente	*
Lettre mal placée	+
Lettre cachée	?

```
Saisir l'affichage de l'ordi : P?????
Mot secret : P?????
Proposition du solveur :  POPLITE
Saisir l'affichage de l'ordi : P***+++
Proposition du solveur :  PRIRENT
Saisir l'affichage de l'ordi : P*+*+++
Proposition du solveur :  PATINES
Saisir l'affichage de l'ordi : P+T+N+*
Proposition du solveur :  PETUNIA
Saisir l'affichage de l'ordi : PETUNIA
```

---

2. Contenu fourni

Une archive `materiel.zip` est fournie et contient :

1. `dico.txt` → dictionnaire des mots autorisés (sans accents)

2. `outils_sutom.py` → bibliothèque de fonctions à compléter
3. `test_outils_sutom.py` → tests unitaires associés
4. `sutom_cli.py` → interface **textuelle interactive**
5. `sutom_gui.py` → interface **graphique** (basée sur `nsi_ui.py`)
6. `sutom_solver.py` → solveur automatique et assistant
7. `test_sutom_solver.py` → tests unitaires + tests de performance

---

3. Cahier des charges

- Les fichiers doivent être complétés à partir des **squelettes fournis**
- Le rendu final est une archive :

`Eleve1_Eleve2_projet_sutom.zip`

- Les questions signalées doivent être répondues dans un fichier :

`reponse.odt`

Qualité du code

- Chaque fonction doit contenir une **docstring**
- Les parties complexes doivent être **commentées**
- Le code doit être validé avec les **tests unitaires**
- Les tests échoués doivent être signalés en commentaire

---

4. Thèmes du programme NSI abordés

- Types construits : tableaux, chaînes, dictionnaires
- Structures fondamentales : boucles, conditions, fonctions
- Interfaces Homme-Machine (CLI / GUI)
- Modularité et séparation logique / interface
- Tests unitaires

---

5. Partie 1 – Bibliothèque d'outils (`outils_sutom.py`)

Objectif

Créer une bibliothèque de fonctions utilisées par :

- l'interface textuelle,

- l'interface graphique,
- le solveur.

Fonctions à implémenter

```
def charger_dico(dico_path, taille_secret):
    """Charge les mots du dictionnaire de longueur donnée"""

def histogramme_dico(dico_path):
    """Histogramme du nombre de mots par taille"""

def tirage_mot(tab):
    """Tirage aléatoire d'un mot"""

def rang_alpha(c):
    """Rang alphabétique (0–25) d'un caractère"""

def histogramme(mot):
    """Histogramme des lettres d'un mot"""

def verif_proposition(prop_joueur, secret):
    """
    Compare proposition et secret
    Retourne :
    - booléen (gagné ou non)
    - chaîne de caractères (* + lettres)
    """

def copie_tab(tab):
    """Copie superficielle d'un tableau"""
```

---

6. Partie 2 – Interface textuelle (`sutom_cli.py`)

Objectif

Permettre à un joueur humain de jouer à SUTOM via le **terminal**.

Structure générale

- Variables globales stockées dans un dictionnaire `jeu`
- Fonctions principales :
    - `interface()`
    - `partie()`

- `validation_joueur()`
- `reponse_ordi()`

Exemple d'exécution

```
Nouvelle partie (o/n) ? o
Taille du mot secret ? 6
Mot secret : P?????
Essai 1/6
Proposition : PASSER
Réponse : P*+*+*
Essai 2/6
Proposition du joueur : PSAUME
Réponse de l'ordinateur : P+***+
Essai 3/6
Proposition du joueur : PENSER
Réponse de l'ordinateur : PE*+**
Essai 4/6
Proposition du joueur : PEINES
Réponse de l'ordinateur : PE+**S
Essai 5/6
Proposition du joueur : PERDIS
Réponse de l'ordinateur : PE**+S
Essai 6/6
Proposition du joueur : PETITS
Réponse de l'ordinateur : PETITS
Gagné en 6 essais
Le mot secret était PETITS
Nouvelle partie (o/n) ? n
```

Questions

1. On définit comme constante toute variable globale dont le nom est en majuscules et dont la valeur ne sera pas modifiée lors de l'exécution du programme.
   Lister toutes les constantes disponibles lorsqu'on exécute `sutom_cli.py`.

Pensez à utiliser la fonction `dir()` pour afficher les variables accessibles dans le script.

2. Décrire précisément le fonctionnement de la boucle d'interface textuelle exécutée par la fonction `interface()`.

Préciser en particulier les conditions d'arrêt des différentes boucles imbriquées, le sens des interactions textuelles (entrée ou sortie) et les fonctions Python qui les rendent possibles.

3. `verif_proposition(prop_joueur, secret)` est appelée dans `reponse_ordi(prop_joueur, secret)` alors que `verif_proposition` n'est pas définie dans `sutom_cli.py`.
   Comment est-ce possible ?
4. Que faut-il modifier dans `sutom_cli.py` si le chemin d'accès au fichier contenant le dictionnaire des mots proposables a changé ?
5. Compléter le bloc de la fonction `validation_joueur()` en respectant sa spécification.
6. Compléter le bloc de la fonction `partie()` en respectant sa spécification

---

7. Partie 3 – Interface graphique (`sutom_gui.py`)

Objectif

Créer une interface graphique interactive à l'aide de `nsi_ui`.

Concepts abordés

- Interacteurs graphiques
- Programmation **événementielle**
- Fonctions de rappel (callbacks)
- Verrouillage du jeu

Fonctions clés

- `interface()`
- `partie()`
- `validation_joueur()`
- `reponse_ordi()`

Questions

1. Lister les différents types d'interacteurs utilisés dans cette interface graphique.
2. Qu'est-ce qu'une fonction de rappel ? Donner des exemples dans le code de `sutom_gui.py`.
3. Combien de boucles `for` ou `while` le code `sutom_gui.py` contient-il ?
4. Pour le déroulement d'une partie, quelle fonction permet cependant une boucle d'interaction entre les interacteurs de l'interface ?
5. Pour les interfaces graphiques, on parle de programmation guidée par les événements. Expliquez cette expression.

Avez-vous déjà écrit des programmes guidés par les événements dans un contexte scolaire ?

6. Quel est le rôle de la valeur `jeu["verrou"]` ?

7. Compléter le code de la fonction `reponse_ordi(prop_joueur, secret)` pour gérer la fin de partie :
   • Affichage du message « Vous avez gagné en .. essais » dans l'interacteur `jeu["bloc_gagne"]` si le joueur a gagné. • Affichage du message « Vous avez perdu en .. essais » dans l'interacteur `jeu["bloc_gagne"]` si le joueur a perdu. • Affichage du mot secret dans l'interacteur `jeu["bloc_annonce"]`.

8. Écrire une fonction `interface2()` où la disposition des blocs est modifiée.

---

8. Partie 4 – Solveur et assistant (`sutom_solver.py`)

Objectif

Développer un solveur automatique fonctionnant selon deux modes :

Mode assistant (`ASSISTANCE = True`)

- L'utilisateur saisit les réponses du site officiel
- Le programme propose le meilleur mot suivant

Mode solveur (`ASSISTANCE = False`)

- Le programme joue **automatiquement**
- Juge et solveur intégrés

Fonctions à compléter

```
def tirage_mot_sans_remise(tab)
def bien_place(caractere)
def compatible(mot, prop_solveur, rep_ordi)
def proposition_solveur(affichage)
def interface_assistant()
```

Tests de performance

- Fonction `test_performance_solveur`
- Mesure :
  - taux de réussite

    - nombre moyen d’essais

- Comparaison avec le corrigé (~3–4 essais)

---

9. Conclusion

Ce projet permet de :

- consolider les bases de Python,
- pratiquer la **modularité**,
- comprendre la programmation événementielle,
- découvrir les **tests unitaires**,
- aborder des problématiques d’**algorithmique avancée**.