Découpage Fonctionnel

Pour le projet nous allons créer trois classes différentes pour les compiler dans un seul programme principal.

- class Affichage (anciennement class Main):
  Cette classe est la classe d'affichage qui permet de mettre en forme l'interface graphique et permet à l'utilisateur d'interagir avec le jeu.

  - def __init__()
    - Initialise une petite fenêtre principale comme une simple "barre". Appelle la fonction "fenetre_principale" qui permet de créer une fenêtre top level utilisée comme fenêtre d'accueil
      - param: None
      - returns: None
  - def fenetre_principale(event):
    - Créer une fenêtre top level utilisée comme page d'accueil, appelle la méthode widget_fen_accueil et initialise les variables utiles à la fenêtre d'accueil
      - param: event
      - returns: None
  - def widget_fen_accueil():
    - Créé les différents widgets de la fenêtre principale (affichage des stats, boutons difficulté et lancer la partie, boutons login/sign up)
      - param: None
      - returns: None
  - def login(event):
    - Ouvre une fenêtre permettant d'entrer l'id et le mot de passe de l'utilisateur
      - param: None
      - returns: None
  - def connecter(event):
    - Utilise la classe User pour reconnaître un utilisateur et récupérer ses informations
      - param: event
      - returns: None
  - def sign_up(event):
    - Ouvre une fenêtre permettant d'entrer son identifiant et son mot de passe

        - param: Event
        - returns: None
- def creerCompte(event):
    - Utilise la classe User pour créer un compte si toutes les conditions sont remplies
        - param: None
        - returns: None

- def statistiques(event):
    - Cette fonction permet de gerer l'affichage pour les statistiques du joueur enregistré au préalable
        - param: None
        - returns: None
- def afficherInfoUser():
    - Permet de modifier l'affichage de la fenêtre d'accueil lorsque l'utilisateur est connecté
        - param: None
        - returns: None
- def afficherClassement():
    - Affiche une fenêtre de classement lorsqu'on appui sur le bouton
        - param: None
        - returns: None
- def createBarreInfo():
    - Affiche une barre d'info lorsqu'on passe la souris sur un widget étant lié à cette fonction. Pour lier un widget, il suffit de la passer en paramètre avec le texte à afficher dans la barre d'info. Utilise la méthode showBarreInfo
        - param: Texte à afficher, Widget sur lequel il faut l'afficher
        - returns None

- def fenetre_jeu(event):
    - Permet de créer la fenêtre de jeu et de récupérer une grille de sudoku de la classe sudoku
        - param: Event, Difficulté
        - returns: None

- def remplirCases():
    - Fais le lien entre la classe sudoku et affichage pour charger les différents chiffres présents dans la grille.
        - param: None
        - return: None

- def indexMatrice():
  - Méthode permettant de renvoyer l'indice (ligne, colonne) d'une valeur dans une matrice 2D
    - param: Élément recherché dans la matrice, Matrice en question
    - return: None

- def validerCases():
  - Méthode validate des entry des cases de sudoku, permet de vérifier si l'utilisateur a bien entré un chiffre entre 1 et 9.
    - param: None
    - return: None

- def envoyerCase():
  - Permet d'envoyer les informations à la classe Sudoku quand l'utilisateur modifie une case de la grille
    - param: None
    - return: None

- def indice():
  - Permet d'ajouter un indice à la grille en cours en utilisant la méthode indice de la classe Sudoku
    - param: None
    - return: None

- def updateChrono():
  - Met en place le chrono dans la fenêtre de jeu
    - param: None
    - returns: None

- <u>class Soduko:</u>

  Cette classe remplace la classe “graph” présente dans notre découpage fonctionnel initial.

  Cette classe va créer notre grille de jeu et va permettre d’agir grâce à différentes fonctions pour donner des indices, …

  - def __init__():
    - initialise les différentes variables (tableaux, difficulté…)
      - param: Int difficultée

        - returns: None

- def parcoursLigne():
    - Parcourt la ligne d'une case spécifique et renvoie toutes les autres cases présentes sur cette ligne
        - param: Instance de type Case, grille
        - returns: Liste d'éléments de type Case

- def parcoursRegion():
    - Parcourt la région 3x3 d'une case spécifique et renvoie toutes les autres cases présentes dans cette région
        - param: Instance de type Case, grille
        - returns:Liste d'éléments de type Case

- def parcoursColonne():
    - Parcourt la colonne d'une case spécifique et renvoie toutes les autres cases présentes sur cette colonne
        - param: Instance de type Case, grille
        - returns: Liste d'éléments de type Case

- def CreateAdjacence():
    - Créer la liste d'adjacence qui va nous permettre d'implémenter la règle du sudoku
        - param: Grille
        - returns: dictionnaire d'adjacence

- def nombreValide():
    - Vérifie si un nombre peut être placé dans une case selon les règles du sudoku
        - param: case, nombre
        - returns: Booléen

- def grilleRemplie():
    - Vérifie si une grille est complète (plus de case à 0)
        - param: Tableau
        - returns: Booléen

- def grilleSolution():
  - Algorithme de backtracking permettant de créer une grille de solution. Elle prend en paramètre une grille solution vide d'abord. Le principe de cet algorithme est que pour chaque case, on choisit une valeur qui pourrait convenir, on explore jusqu'au bout si cette valeur convient vraiment, et si elle ne convient pas, on remonte pour la changer.
    - param: Grille qui deviendra la grille solution
    - returns: False tant qu'aucune solution n'est trouvé, True quand il n'y a plus de case vide (solution trouvée)

- def resoudreSudoku():
  - Algorithme de backtracking permettant de résoudre une grille donnée. Elle prend en paramètre une grille partiellement remplie. Le principe est le même que l'algorithme de génération de sudoku. Cette méthode sera utilisée pour vérifier l'unicité des solutions d'une grille donnée avec la variable self.counter
    - param: Grille
    - returns: False dans tous les cas (seul la variable self.counter nous importe)

- def enleverCases():
  - Méthode permettant d'enlever des cases de la grille solution tout en vérifiant l'unicité des solutions de la grille proposée. Pour limiter le nombre de calculs et le temps d'exécution, on teste en prenant des cases vides aléatoirement si avec une grille solution on peut trouver une unique solution correspondant aux critères de difficulté. Sinon on change de grille solution en reprenant toutes les autres fonctions (ce qui est rapide). Cette méthode utilise d'autres méthodes secondaires.
    - param: None
    - returns: None

- def indice():
  - Permet de modifier la grille en cours pour remplacer une de ses cases vides par une case de la grille solution
    - param: None
    - returns: Ligne, colonne, valeur

- def erreurSudoku():

        - Vérifie si il y a une erreur dans la liste
            - param: None
            - returns : None si pas d'erreur, la première case d'erreur si il y en a une

- Class data_user:

Cette classe permet de stocker les données des utilisateurs et de les traiter. Cette classe nous permettra par exemple de vérifier si l'utilisateur rentre le bon mot de passe et le bon pseudonyme lors de sa connexion.

    - def __init__():
        - Initialise les principales données utilisateurs (pseudonyme, mdp, nombre de parties, temps total de jeu, record de temps, meilleure partie)
            - param: None
            - returns: None
    - def lire_donnees_json():
        - Lit le fichier json et le met dans un dictionnaire data
            - param: None
            - returns: Dictionnaire des utilisateurs
    - def _ajouter_utilisateur_json():
        - Ajoute un nom d'utilisateur et son mot de passe au fichier json
            - param: Nom d'utilisateur, mot de passe
            - returns: None
    - def lire_data_user():
        - Accède aux données de l'utilisateur quand il se connecte
            - param: Nom d'utilisateur, mot de passe
            - returns: None
    - def actualiser_data_user():
        - met à jour le fichier à chaque fin de partie en actualisant le json avec les données de la dernière partie. Cette méthode utilise d'autres méthodes secondaires pour updater les différentes statistiques
            - param: Temps, difficultée
            - returns: Points de la partie (pour l'affichage des points gagnés quand la partie est finie)

    - def updatePoints():
        - On définit le nombre de points à ajouter en fonction de la difficulté

        - param: temps, difficulté
        - returns: Nombre de points à ajouter pour cette partie

    - def updateClassement():
        - Permet de redéfinir le classement des utilisateurs
            - param: None
            - returns: None
    - def getClassement():
        - Permet de récupérer une liste triée des utilisateurs dans l'ordre du classement

- Class Case:

Cette classe est utilisée dans la classe sudoku et va permettre de récupérer, et de vérifier les valeurs possibles que prennent les cases de la grille

- def __init__():
    - initialise les différents paramètres de la classe (ligne, colonne, valeur)
        - param : ligne, colonne
        - returns : None

- def definir_region():
    - Définit la région associée à la case. Les régions sont numérotées de 1 à 9 en partant d'en haut à gauche et en allant de gauche à droite
        - param : None
        - Returns : None