## La classe QSolutionToSolvePanel

C’est une classe **abstraite** qui sert de base pour définir les panneaux de résolution des différents problèmes dans l'application.

Les éléments de paramétrisation du problème sont dans les sous-classes de la classe **QSolutionToSolvePanel**.

Chaque sous-classe doit impérativement implémenter les propriétés et méthodes abstraites suivantes pour assurer une intégration correcte :

        - name (propriété) : Un nom court et descriptif du problème.

        - summary (propriété) : Un résumé concis du problème, généralement de 1 à 3 phrases.

        - description (propriété) : Un texte détaillé expliquant le problème en profondeur.

        - problem\_definition (propriété) : Un objet définissant le problème à résoudre.

        - default\_parameters (propriété) : Les paramètres par défaut pour l'algorithme génétique.

        - \_update\_from\_simulation (méthode) : Une méthode protégée pour mettre à jour l'interface utilisateur en fonction de l'état actuel de la simulation.

Les deux problèmes proviennent de la même classe abstraite et permettent au code principal de rester **générique** et **flexible**. En utilisant des méthodes avec des **noms identiques** et en tirant parti du **polymorphisme.** La seule ligne à ajouter dans gamain.py est celle qui ajoute le problème à l’application. Ce problème peut avoir des paramètres ou non et on passe une instance, pas une classe. La classe hérite de **QSolutionToSolvePanel**.

Ex : ga\_app.add\_solution\_panel(problemeClasse())

## 3 chaînes de caractères demandés

**name** :  Un nom court et descriptif du problème

**summary** : Un résumé concis du problème, généralement de 1 à 3 phrases

**description** : Un texte détaillé expliquant le problème en profondeur

## Définition du problème

La définition du problème inclut les domaines des chromosomes et la fonction objective.

**domaine** :

Espace de recherche dans lequel on cherche la solution. Pour D dimensions, le domaine est définir par un intervalle di [… , …] qui limite les valeurs possibles de chaque dimension.

i représentant quelle dimension.

**fonction objective (fitness)** :

La fonction objective consiste à déterminer à quel point une solution performe bien. Dans ce cas-ci, on   
considère la distance 1d entre la solution courante et la valeur recherchée.

Puisque cette valeur oriente la résolution vers une **minimisation**, indiquant que la plus petite valeur est la   
meilleure, on inverse la réponse en fonction de la plage maximum des données.

Si le problème est de minimisation, on peut transformer cela en maximisation en prenant **1 divisé par la valeur**.

### Classe Domain

**ranges (numpy.ndarray):** Un tableau 2D contenant les intervalles [min, max] pour chaque dimension.

**names (tuple of str):** Noms des dimensions.

**dimension (int):** Le nombre de dimensions du problème, dérivé de la taille de `ranges`.

**Note:**

            - `ranges` doit :

                - être un numpy.ndarray avec un dtype de float (float16, float32, float64, etc.)

                - avoir deux colonnes pour les valeurs minimales et maximales

                - avoir n lignes représentant n dimensions

                - chaque dimension doit avoir un nom associé dans `names`

            - 'names' doit :

                - être un tuple de string

                - avoir n lignes représentant n dimensions

                - pour un problème à une seule dimension, `names` doit être dans un tuple (un tuple avec une virgule, par exemple ('x',))

### Fonction objective

La fonction de fitness évalue un chromosome et retourne une valeur de performance sous forme de float. Cette valeur indique à quel point la solution est bonne par rapport à l'objectif du problème. Plus la valeur est élevée, meilleure est la solution.

C’est ici qu’on maximise la valeur trouvée.

## Paramètres par défaut de l'algorithme génétique

Ces paramètres sont utilisés pour initialiser les paramètres de l'interface graphique

**maximum\_epoch** : **nombre maximal d'itérations** que l'algorithme génétique effectuera

**population\_size** : **nombre d'individus** (ou solutions potentielles) dans chaque génération de l'algorithme

**elitism\_rate** : **proportion des meilleures solutions** de chaque génération qui seront **conservées** pour la génération suivante, sans modifications

**selection\_rate** : **proportion d'individus** qui seront choisis pour se reproduire et **former la nouvelle génération**

**mutation\_rate** : probabilité qu'une **mutation** se produise lors de la reproduction (pour chaque nouvel individu)

## Problème géométrique :

chromosome = [x, y, theta, scale]

Domaine : Dimensionalité et intervalles

* dimensions = 4
* x = [0, largeur du canevas]
* y = [0, hauteur du canevas]
* theta = [0, 2π] (0, 360 degrés)
* scale = [0, facteur de mise à l’échelle max]

Fonction objective :

* Surface valide : surface de la forme à l'intérieur du canevas.
* Si la forme dépasse le canevas ou intersecte des obstacles, fitness return 0
* Le plus grand la surface, le meilleur est la solution

Paramètres d’entrée qui seront configurable par l’utilisateur :

* Nombre d’obstacles (0 - 100)
* Liste de formes à configurer

Rétroaction visuelle à produire :

* Canevas noir avec les obstacles d’un pixel blanc
* Forme actuelle est d’une couleur autre que blanc et noir
* Essais de formes précédentes sont blanche et non-remplies
* Graphique linéaire

## Problème des caméras dans un champ :

chromosome = [x, y, theta]

Domaine : Dimensionalité et intervalles

dimensions = 3

x = [0, largeur du champ]

y = [0, hauteur du champ]

theta = [0, 2π] (0, 360 degrés)

Fonction objective :

Surface du champ total couverte par les caméras

Le plus grand la surface, le meilleur est la solution

Paramètres d’entrée qui seront configurable

* Dimensions du champ
* Nombre de caméras (1 - 3)
* L’angle du champ des caméras ?
* Nombre d’obstacles (pour l’instant 0)

Rétroaction visuelle à produire :

* Surface 2D qui représente le champ
* Carré sur la surface qui représente les caméras
* Angle de la surface couvert par chaque caméra

## Stratégies de mutation

* Stratégie de mutation qui mute tous les gènes d'un individu à la fois.

La mutation est effectuée sur chaque gène d'un individu avec la probabilité donnée.

* Stratégie de mutation générale qui donnera un taux de probabilité pour effectuer une stratégie de mutation parmi plusieurs
* Stratégie de mutation spécifique qui va seulement muter 2 gènes (x et y des caméras)