Jeu de la vie

1.0

Généré par Doxygen 1.12.0

1 Index des espaces de nommage

1.1 Liste des espaces de nommage

Liste de tous les espaces de nommage avec une brève description:

2 Index hiérarchique

2.1 Hiérarchie des classes

Cette liste d'héritage est classée approximativement par ordre alphabétique :

3 Index des classes

3.1 Liste des classes

Liste des classes, structures, unions et interfaces avec une brève description :

4 Index des fichiers

4.1 Liste des fichiers

Liste de tous les fichiers avec une brève description :

5 Documentation des espaces de nommage

5.1 Référence de l'espace de nommage dependances

Espaces de nommage

- namespace constantes
- namespace jdlv
- namespace overload

5.2 Référence de l'espace de nommage dependances.constantes

Espaces de nommage

- namespace constantes

5.3 Référence de l'espace de nommage dependances.constantes.constantes

Classes

- class Direction
- class Etat

5.4 Référence de l'espace de nommage dependances.jdlv

Espaces de nommage

- namespace cellule
- namespace scene
- namespace tableau

5.5 Référence de l'espace de nommage dependances.jdlv.cellule

Classes

- class Cellule

5.6 Référence de l'espace de nommage dependances.jdlv.scene

Classes

- class Scene

5.7 Référence de l'espace de nommage dependances.jdlv.tableau

Fonctions

— list[list[Any]] construit (int h, int l, Any valeur_defaut)

— tuple[bool, str] est_template_valide (Any tableau)

5.7.1 Documentation des fonctions

construit()

```
list[list[Any]] dependances.jdlv.tableau.construit (
            int h,
            int l,
            Any valeur_defaut)
```

```
Entrée:
    h: int (hauteur)
    l: int (largeur)
Sortie:
    list[list[int]]
Rôle:
    Construit un tableau de taille taille avec comme valeur valeur_defaut
```

Définition à la ligne 5 du fichier tableau.py.

```
00005 def construit(h: int, l: int, valeur_defaut: Any) -> list[list[Any]]:
00006     """
00007     Entrée:
00008         h: int (hauteur)
00009         l: int (largeur)
00010     Sortie:
00011         list[list[int]]
00012     Rôle:
00013         Construit un tableau de taille taille avec comme valeur valeur_defaut
00014     """
00015     # Déclaration d'un tableau vide
00016     plateau = []
00017
00018     # Pour chaque ligne de la matrice (de hauteur h)
00019     for i in range(h):
00020         # On ajoute une nouvelle ligne au tableau
00021         plateau.append([])
00022         # Pour chaque élément de la ligne (de longeur l)
00023         for _ in range(l):
00024             # On ajoute la valeur par défaut à la ligne du tableau
00025             plateau[i].append(valeur_defaut)
00026
00027     # On retourne le tableau
00028     return plateau
00029
00030
```

est_template_valide()

```
tuple[bool, str] dependances.jdlv.tableau.est_template_valide (
            Any tableau)
```

```
Entrée:
    tableau: list[str]  (attendu)
Sortie:
    tuple[
        bool,   (validité)
        str     (message d'erreur)
    ]
Rôle:
    Retourne True si le tableau respecte le format d'un template. En cas
    d'erreur, le deuxième élément sera le message qui explique l'erreur.
```

Définition à la ligne 31 du fichier tableau.py.

```
00031 def est_template_valide(tableau: Any) -> tuple[bool, str]:
00032     """
00033     Entrée:
00034         tableau: list[str]  (attendu)
00035     Sortie:
00036         tuple[
00037             bool,   (validité)
00038             str     (message d'erreur)
00039         ]
00040     Rôle:
00041         Retourne True si le tableau respecte le format d'un template. En cas
00042         d'erreur, le deuxième élément sera le message qui explique l'erreur.
00043     """
00044     message = ""
00045     # Si le tableau est de type list
00046     if type(tableau) is list:
00047         # Déclare taille et l'initialise à -1
00048         taille = -1
00049         # Déclaration d'une variable d'arrêt
00050         lignes_valide = True
00051         # Déclaration d'un itérateur
00052         i = 0
00053         # Pour chaque ligne du tableau tant que les lignes sont valides
00054         while i < len(tableau) and lignes_valide:
00055             # Si la ligne du tableau est une chaîne de caractères
00056             if type(tableau[i]) is str:
00057                 # On cherche l'expression régulière suivante dans la ligne
00058                 # ^ : on part du premier caractère
00059                 # $ : on s'arrête au dernier caractère
00060                 # 0-1 : l'élément est soit 0 soit 1
00061                 # , : il est suivit d'une virgule
00062                 # * : il est présent 0 ou plus fois
00063                 # \\n : la ligne finit par \n
00064                 correspondance = search("^[0-1,]*\\n$", tableau[i])
00065 
00066                 # S'il y a une correspondance
00067                 if correspondance:
00068                     # Si la taille n'a pas encore été définit
00069                     if taille == -1:
00070                         # On définit la taille d'une ligne
00071                         taille = len(tableau[i])
00072                     # Si la ligne ne fait pas la même taille que la première
00073                     if len(tableau[i]) != taille:
00074                         # On définit un message d'erreur
00075                         message = "Les lignes doivent être de même taille"
00076                         # On arrête la boucle
00077                         lignes_valide = False
00078                 # S'il n'y a pas de correspondance
00079                 else:
00080                     # On définit un message d'erreur
00081                     message = "Les éléments doivent être 0 ou 1. Ils " +\
00082                         "doivent être séparés par des virgules."
00083                     # On arrête la boucle
00084                     lignes_valide = False
00085             else:
00086                 # On définit un message d'erreur
00087                 message = "Chaque ligne doit être une chaîne de caractère " +\
00088                     "encodant la valeurs des cellules."
00089                 # On arrête la boucle
00090                 lignes_valide = False
00091 
00092             # Incrémentation de l'itérateur
00093             i += 1
00094 
00095         # On retourne True si toutes les lignes sont valides, False sinon
00096         return lignes_valide, message
00097 
00098     # On retourne False sinon
00099     return False, message
```

5.8 Référence de l'espace de nommage dependances.overload

Espaces de nommage

— namespace overload

5.9 Référence de l'espace de nommage dependances.overload.overload

Classes

— class Overload

Fonctions

— callable signature (*types)

— tuple[str] get_signature_complete (obj, tuple sig=())

5.9.1 Documentation des fonctions

get_signature_complete()

```
tuple[str] dependances.overload.overload.get_signature_complete (
              obj,
             tuple  sig = ())


Entrées:
    obj: Any
        obj est une variable quelconque
    sig: tuple
        signature de obj jusqu'alors
Sortie:
    tuple[str]
Rôle:
    Retourne l'arbre de signature de obj
Exemple:
    >>> a: int = 1
    >>> get_inherited_signature(a)
    ['int', 'object']
Explications:
    Le but de cette fonction est de savoir de quelleS classeS hérite obj.
    Plus obj est abstrait, plus sont arbre de signature sera grand.
    L'arbre ne possède qu'une dimension pour facilité son traitement.
Addendum:
    Cette fonction est la raison pour laquelle j'ai recodé ce module.
    Ce dernier ne permet pas de faire de surcharge 'intelligente'.
    Il est évident les parmatères suivants de type:
        list, int, int
    correspond à la signature suivante:
        list, int, object
    car int hérite de object. Or l'héritage doit respecter le principe de
    substitution de Liskov.
```

Définition à la ligne 61 du fichier overload.py.

```
00061 def get_signature_complete(obj, sig: tuple = ()) -> tuple[str]:
00062     """
00063     Entrées:
00064         obj: Any
00065             obj est une variable quelconque
00066         sig: tuple
00067             signature de obj jusqu'alors
00068     Sortie:
00069         tuple[str]
00070     Rôle:
00071         Retourne l'arbre de signature de obj
00072     Exemple:
00073         »> a: int = 1
00074         »> get_inherited_signature(a)
00075         ['int', 'object']
00076     Explications:
00077         Le but de cette fonction est de savoir de quelleS classeS hérite obj.
00078         Plus obj est abstrait, plus sont arbre de signature sera grand.
00079         L'arbre ne possède qu'une dimension pour facilité son traitement.
00080     Addendum:
00081         Cette fonction est la raison pour laquelle j'ai recodé ce module.
00082         Ce dernier ne permet pas de faire de surcharge 'intelligente'.
00083         Il est évident les parmatères suivants de type:
00084             list, int, int
00085         correspond à la signature suivante:
00086             list, int, object
00087         car int hérite de object. Or l'héritage doit respecter le principe de
00088         substitution de Liskov.
00089     """
00090     # Si c'est le premier appelle
00091     if len(sig) == 0:
00092         # On redéfinit obj par sa classe
00093         obj = obj.__class__
00094         # on attribut le singleton du nom de la classe d'obj à sig
00095         sig = (obj.__name__,)
00096         # On retourne get_inherited_signature(obj, sig)
00097         return get_signature_complete(obj, sig)
00098     # Si obj est object
00099     elif obj.__name__ == object.__name__:
00100         # On retourne la signature complète plus object
00101         return sig + (obj.__name__,)
00102     # Sinon
00103     else:
00104         # parent_sig est initialisé avec les classes directement parentes d'obj
00105         parent_sig = obj.__bases__
00106         # Si obj à plus d'un parent directe
00107         if type(parent_sig) is tuple:
00108             # Pour chaque parent
00109             for par in parent_sig:
00110                 # Si le parent est object
00111                 if par.__name__ == object.__name__:
00112                     # on ajoute ('object',) à sig
00113                     sig += (par.__name__,)
00114                 # Sinon
00115                 else:
00116                     # on ajoute l'arbre de signature du parent
00117                     sig += get_signature_complete(par, (par.__name__,))
00118             # On retourne l'arbre de signature de obj
00119             return sig
00120         # Si sig n'a qu'un parent directe
00121         else:
00122             # On retourne sig plus son type plus son arbre de signature
00123             return sig  + (obj.__name__,) + (
00124                 get_signature_complete(parent_sig, sig),)
00125
00126
```

signature()

```
callable dependances.overload.overload.signature (
            * types)


Entrée(s):
    types: str
Sortie:
    callable
Rôle:
    Retourne la fonction dont signature est un décorateur avec une
```

```
    propriété signature qui est un tuple des types
Addendum:
    Mes excuses pour les explications et commentaires, mais le code n'étant
    pas le mien, non commenté et parfois hors de ma porté, il a été
    complexe à comprendre. La notion de décoration en python n'est déjà pas
    facile à implémenter, alors à mettre en oeuvre, c'est une autre
    histoire.
```

Définition à la ligne 8 du fichier overload.py.

```
00008 def signature(*types) -> callable:
00009     """
00010     Entrée(s):
00011         types: str
00012     Sortie:
00013         callable
00014     Rôle:
00015         Retourne la fonction dont signature est un décorateur avec une
00016         propriété signature qui est un tuple des types
00017     Addendum:
00018         Mes excuses pour les explications et commentaires, mais le code n'étant
00019         pas le mien, non commenté et parfois hors de ma porté, il a été
00020         complexe à comprendre. La notion de décoration en python n'est déjà pas
00021         facile à implémenter, alors à mettre en oeuvre, c'est une autre
00022         histoire.
00023     """
00024     def func(f: callable) -> callable:
00025         """
00026         Entrée:
00027             f: callable (fonction qui est décorée)
00028         Sortie:
00029             callable
00030         Rôle:
00031             Retourne la fonction dont signature est un décorateur avec une
00032             propriété signature qui est un tuple des types
00033         """
00034         def inner_func(callingObj, *args, **kwargs) -> callable:
00035             """
00036             Entrées:
00037                 callingObj: self (puisque f est une méthode Cf Overload)
00038                 *args: arguments passé à f lors de son appelle
00039                 **kwargs: couples (clé;argument) passé à f lors de son appelle
00040             Sortie:
00041                 Sortie de f(callingObj, *args, **kwargs)
00042             Rôle:
00043                 Retourne la valeur de f(callingObj, *args, **kwargs)
00044             """
00045             # retourne la valeur de retour de f, la fonction décorée, en lui
00046             # passant en paramètre les arguments qui lui sont passés en
00047             # paramètre
00048             return f(callingObj, *args, **kwargs)
00049 
00050         # Créer une propriété signature pour la fonction décorée et lui assigne
00051         # le tuple de types
00052         inner_func.signature = types  # types -> tuple
00053 
00054         # retourn l'objet inner_func
00055         return inner_func
00056 
00057     # retourne l'objet func
00058     return func
00059 
00060 
```

5.10 Référence de l'espace de nommage jeu_de_la_vie_cli

Classes

— class JeuDeLaVieCLI

Variables

— jeu = JeuDeLaVieCLI()

5.10.1 Documentation des variables

jeu

```
jeu_de_la_vie_cli.jeu = JeuDeLaVieCLI()
```

Définition à la ligne 426 du fichier jeu_de_la_vie_cli.py.

5.11 Référence de l'espace de nommage jeu_de_la_vie_gui

Classes

— class JeuDeLaVieGUI

Variables

— reponse

— proc = run("pip install PySide6", stdout=PIPE)

— app = QApplication(argv)

— my_window = JeuDeLaVieGUI()

5.11.1 Documentation des variables

app

```
jeu_de_la_vie_gui.app = QApplication(argv)
```

Définition à la ligne 717 du fichier jeu_de_la_vie_gui.py.

my_window

```
jeu_de_la_vie_gui.my_window = JeuDeLaVieGUI()
```

Définition à la ligne 719 du fichier jeu_de_la_vie_gui.py.

proc

```
jeu_de_la_vie_gui.proc = run("pip install PySide6", stdout=PIPE)
```

Définition à la ligne 17 du fichier jeu_de_la_vie_gui.py.

reponse

```
jeu_de_la_vie_gui.reponse
```

Valeur initiale :

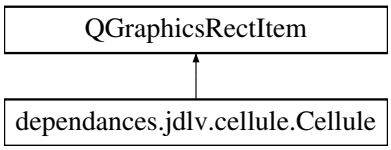
```
00001 =  input("Le module PySide6 n'est pas installé, souhaitez-vous " +
00002                  "l'installer automatiquement ? (o/n)")[0]
```

Définition à la ligne 7 du fichier jeu_de_la_vie_gui.py.

6 Documentation des classes

6.1 Référence de la classe dependances.jdlv.cellule.Cellule

Graphe d'héritage de dependances.jdlv.cellule.Cellule:



Fonctions membres publiques

- None __init__ (self, QGraphicsItem|None parent=None, Etat etat=Etat.Vivant)
- None peint (self)
- None mousePressEvent (self, QGraphicsSceneMouseEvent even)
- None hoverEnterEvent (self, QGraphicsSceneHoverEvent even)
- None set_etat (self, Etat etat)
- Etat get_etat (self)
- str __repr__ (self)
- object __deepcopy__ (self, dict memo)
- bool __eq__ (self, Any valeur)

Attributs publics

- etat = etat

6.1.1 Description détaillée

```
Hérite de:
    QGraphicsRectItem
Rôle:
    Représente une cellule
```

Définition à la ligne 16 du fichier cellule.py.

6.1.2 Documentation des constructeurs et destructeur

__init__()

```
None dependances.jdlv.cellule.Cellule.__init__ (
              self,
             QGraphicsItem | None  parent = None,
             Etat  etat = Etat.Vivant)
```

```
Entrée:
    self: Cellule
    parent QGraphicsItem | None
        valeur par défaut: None
    etat: Etat
Sortie:
    None (ctor)
Rôle:
    Construit un nouvel objet Cellule
```

Définition à la ligne 23 du fichier cellule.py.

```
00024                 etat: Etat = Etat.Vivant) -> None:
00025         """
00026         Entrée:
00027             self: Cellule
00028             parent QGraphicsItem | None
00029                 valeur par défaut: None
00030             etat: Etat
00031         Sortie:
00032             None (ctor)
00033         Rôle:
00034             Construit un nouvel objet Cellule
00035         """
00036         # Initialise la classe mère
00037         QGraphicsRectItem.__init__(self, parent)
00038         # L'objet accepte de recevoir les événements de survole de la souris
00039         self.setAcceptHoverEvents(True)
00040         # on définit l'état
00041         self.etat = etat
00042         # On peint la cellule en fonction de l'état
00043         self.peint()
00044         # Définit le rectangle d'affichage (x, y, w, h)
00045         self.setRect(0, 0, 50, 50)
00046         # Définit la couleur et taille du contour sur transparent et 0
00047         # car le contour agrandit le rectangle
00048         self.setPen(QPen(Qt.GlobalColor.transparent, 0))
00049
```

6.1.3 Documentation des fonctions membres

__deepcopy__()

```
object dependances.jdlv.cellule.Cellule.__deepcopy__ (
              self,
             dict memo)
```

```
Entrées:
    self: Cellule
    memo: dict
Sortie:
    object (copie de self)
Rôle:
    Méthode magique du module copy pour effectuer des copies de
    l'object.
Addendum:
    Ne connaissant pas le module copy, j'ai utilisé la réponse
    suivante : https://stackoverflow.com/a/15774013/15793884
```

Définition à la ligne 167 du fichier cellule.py.

```
00167     def __deepcopy__(self, memo: dict) -> object:
00168         """
00169         Entrées:
00170             self: Cellule
00171             memo: dict
00172         Sortie:
00173             object (copie de self)
00174         Rôle:
00175             Méthode magique du module copy pour effectuer des copies de
00176             l'object.
00177         Addendum:
00178             Ne connaissant pas le module copy, j'ai utilisé la réponse
00179             suivante : https://stackoverflow.com/a/15774013/15793884
00180         """
00181         # classe de self (Cellule)
00182         cls = self.__class__
00183         # Créer une nouvelle instance de Cellule
00184         result = cls.__new__(cls)
00185         # ajoute au dictionnaire memo le couple identifant du présent object et
00186         # nouvel object
00187         memo[id(self)] = result
00188         # Pour chaque nom d'attribut et sa valeur de self (le présent object)
00189         for k, v in self.__dict__.items():
00190             # On affecte à l'attribut k de result une copie de la valeur de
00191             # l'attribut k
00192             setattr(result, k, deepcopy(v, memo))
00193 
00194         # Revoie l'object copié
00195         return result
00196 
```

__eq__()

```
bool dependances.jdlv.cellule.Cellule.__eq__ (
              self,
             Any valeur)
```

```
Entrées:
    self: Cellule
    valeur: Any
Sortie:
    bool
Rôle:
    Méthode magique appelée par l'opérateur d'égalité ==.
Explications:
    Une explication complète du fonctionnement de __eq__ est disponible
    via le lien suivant: https://stackoverflow.com/a/3588809/15793884
```

Définition à la ligne 197 du fichier cellule.py.

```
00197     def __eq__(self, valeur: Any) -> bool:
00198         """
00199         Entrées:
00200             self: Cellule
00201             valeur: Any
00202         Sortie:
00203             bool
00204         Rôle:
00205             Méthode magique appelée par l'opérateur d'égalité ==.
00206         Explications:
00207             Une explication complète du fonctionnement de __eq__ est disponible
00208             via le lien suivant: https://stackoverflow.com/a/3588809/15793884
00209         """
00210         # Retoure True si etat est égal à valeur
00211         return self.etat == valeur
```

__repr__()

```
str dependances.jdlv.cellule.Cellule.__repr__ (
              self)
```

```
Entrée:
    self: Cellule
Sortie:
    str
Rôle:
    Méthode magique __repr__, représentation de la cellule en str.
```

Définition à la ligne 149 du fichier cellule.py.

```
00149     def __repr__(self) -> str:
00150         """
00151         Entrée:
00152             self: Cellule
00153         Sortie:
00154             str
00155         Rôle:
00156             Méthode magique __repr__, représentation de la cellule en str.
00157         """
00158         # Si la cellule est vivante
00159         if self.etat is Etat.Vivant:
00160             # Retourne ■
00161             return "■"
00162         # Si la cellule est morte
00163         else:
00164             # Retourne □
00165             return "□"
00166
```

get_etat()

```
Etat dependances.jdlv.cellule.Cellule.get_etat (
              self)
```

```
Entrée:
    self: Cellule
Sortie:
    Etat
Rôle:
    Retourne l'état de la cellule.
```

Définition à la ligne 137 du fichier cellule.py.

```
00137     def get_etat(self) -> Etat:
00138         """
00139         Entrée:
00140             self: Cellule
00141         Sortie:
00142             Etat
00143         Rôle:
00144             Retourne l'état de la cellule.
00145         """
00146         # On retourne la valeur de l'attribut est_vivant
00147         return self.etat
00148
```

hoverEnterEvent()

```
None dependances.jdlv.cellule.Cellule.hoverEnterEvent (
              self,
             QGraphicsSceneHoverEvent even)
```

```
Réimplémentation de hoverEnterEvent hérité de QGraphicsScene
Entrées:
    self: Cellule
    even: QEvent (et les classes qui en hérite)
Sortie:
    bool
Rôle:
    Capture les événements de QGraphicsRectItem, les traites, puis les
    rends.
```

Définition à la ligne 92 du fichier cellule.py.

```
00092     def hoverEnterEvent(self, even: QGraphicsSceneHoverEvent) -> None:
00093         """
00094         Réimplémentation de hoverEnterEvent hérité de QGraphicsScene
00095         Entrées:
00096             self: Cellule
00097             even: QEvent (et les classes qui en hérite)
00098         Sortie:
00099             bool
00100         Rôle:
00101             Capture les événements de QGraphicsRectItem, les traites, puis les
00102             rends.
00103         """
00104         # Si la cellule est vivante
00105         if self.etat is Etat.Vivant:
00106             # Envoie un évènement QStatusTipEvent à la scène mère
00107             self.scene().event(QStatusTipEvent(
00108                 "Cliquez droit pour changer l'état. Pour modifier l'état de " +
00109                 "plusieurs cellules, maintenez le clique droit en déplaçant " +
00110                 "la souris."))
00111         # Si la cellule est morte
00112         else:
00113             # Envoie un évènement QStatusTipEvent à la scène mère
00114             self.scene().event(QStatusTipEvent(
00115                 "Cliquez gauche pour changer l'état. Pour modifier l'état " +
00116                 "de plusieurs cellules, maintenez le clique gauche en " +
00117                 "déplaçant la souris."))
00118 
00119         # Rend l'évènement
00120         return super().hoverEnterEvent(even)
00121 
```

mousePressEvent()

```
None dependances.jdlv.cellule.Cellule.mousePressEvent (
              self,
             QGraphicsSceneMouseEvent even)
```

```
Réimplémentation de mousePressEvent hérité de QGraphicsRectItem
Entrées:
    self: Cellule
    even: QGraphicsSceneMouseEvent (événement)
Sortie:
    None (modification en place)
Rôle:
    Détecte les événements de pression de souris sur la cellule.
```

Définition à la ligne 68 du fichier cellule.py.

```
00068     def mousePressEvent(self, even: QGraphicsSceneMouseEvent) -> None:
00069         """
00070         Réimplémentation de mousePressEvent hérité de QGraphicsRectItem
00071         Entrées:
00072             self: Cellule
00073             even: QGraphicsSceneMouseEvent (événement)
00074         Sortie:
00075             None (modification en place)
00076         Rôle:
00077             Détecte les événements de pression de souris sur la cellule.
00078         """
00079         # Si le click est un click gauche
00080         if even.button() is Qt.MouseButton.LeftButton:
00081             # Inverse l'état de la cellule
00082             self.etat = Etat.Vivant
00083         elif even.button() is Qt.MouseButton.RightButton:
00084             # Inverse l'état de la cellule
00085             self.etat = Etat.Mort
00086         # On peint la cellule en fonction de l'état
00087         self.peint()
00088 
00089         # On rend l'événement à la classe mère
00090         return super().mousePressEvent(even)
00091 
```

peint()

```
None dependances.jdlv.cellule.Cellule.peint (
              self)


Entrée:
    self: Cellule
Sortie:
    None (modification en place)
Rôle:
    Définit la couleur de la cellule
```

Définition à la ligne 50 du fichier cellule.py.

```
00050     def peint(self) -> None:
00051         """
00052         Entrée:
00053             self: Cellule
00054         Sortie:
00055             None (modification en place)
00056         Rôle:
00057             Définit la couleur de la cellule
00058         """
00059         # Si la cellule est vivante
00060         if self.etat is Etat.Vivant:
00061             # On remplit en noir
00062             self.setBrush(QBrush(Qt.GlobalColor.black))
00063         # si la cellule est morte
00064         else:
00065             # On remplit en blanc
00066             self.setBrush(QBrush(Qt.GlobalColor.white))
00067
```

set_etat()

```
None dependances.jdlv.cellule.Cellule.set_etat (
              self,
             Etat etat)


Entrées:
    self: Cellule
    etat: Etat
Sortie:
    None (modification en place)
Rôle:
    Redéfinit l'état de la cellule.
```

Définition à la ligne 122 du fichier cellule.py.

```
00122     def set_etat(self, etat: Etat) -> None:
00123         """
00124         Entrées:
00125             self: Cellule
00126             etat: Etat
00127         Sortie:
00128             None (modification en place)
00129         Rôle:
00130             Redéfinit l'état de la cellule.
00131         """
00132         # On met à jour l'attribut est_vivant
00133         self.etat = etat
00134         # On peint la cellule en fonction de l'état
00135         self.peint()
00136
```

6.1.4 Documentation des données membres

etat

```
dependances.jdlv.cellule.Cellule.etat = etat
```

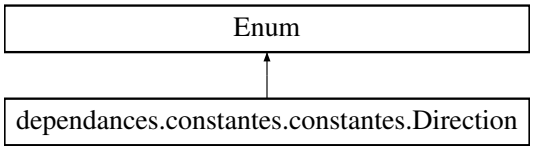
Définition à la ligne 41 du fichier cellule.py.

La documentation de cette classe a été générée à partir du fichier suivant :

— D:/Jeu de la vie/source/dependances/jdlv/cellule.py

6.2 Référence de la classe dependances.constantes.constantes.Direction

Graphe d'héritage de dependances.constantes.constantes.Direction:



Attributs publics statiques

— int Nord = 0

— int Est = 1

— int Sud = 2

— int Ouest = 3

6.2.1 Description détaillée

```
Hérite de:
    Enum
Rôle:
    Abstaitiste les directions pour simplifier la lecture du code
```

Définition à la ligne 18 du fichier constantes.py.

6.2.2 Documentation des données membres

Est

```
int dependances.constantes.constantes.Direction.Est = 1  [static]
```

Définition à la ligne 28 du fichier constantes.py.

Nord

```
int dependances.constantes.constantes.Direction.Nord = 0  [static]
```

Définition à la ligne 26 du fichier constantes.py.

Ouest

```
int dependances.constantes.constantes.Direction.Ouest = 3  [static]
```

Définition à la ligne 32 du fichier constantes.py.

Sud

```
int dependances.constantes.constantes.Direction.Sud = 2  [static]
```

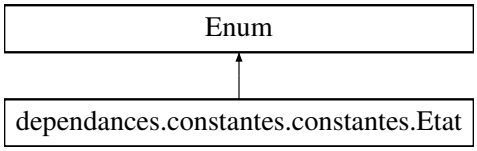
Définition à la ligne 30 du fichier constantes.py.

La documentation de cette classe a été générée à partir du fichier suivant :

— D:/Jeu de la vie/source/dependances/constantes/constantes.py

6.3 Référence de la classe dependances.constantes.constantes.Etat

Graphe d'héritage de dependances.constantes.constantes.Etat:



Attributs publics statiques

— bool Vivant = True

— bool Mort = False

6.3.1 Description détaillée

```
Hérite de:
    Enum
Rôle:
    Abstaitiste les états de la cellule pour simplifier la lecture du code
```

Définition à la ligne 5 du fichier constantes.py.

6.3.2 Documentation des données membres

Mort

```
bool dependances.constantes.constantes.Etat.Mort = False  [static]
```

Définition à la ligne 15 du fichier constantes.py.

Vivant

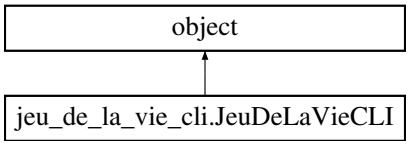
```
bool dependances.constantes.constantes.Etat.Vivant = True  [static]
```

Définition à la ligne 13 du fichier constantes.py.

La documentation de cette classe a été générée à partir du fichier suivant :
— D:/Jeu de la vie/source/dependances/constantes/constantes.py

6.4 Référence de la classe jeu_de_la_vie_cli.JeuDeLaVieCLI

Graphe d'héritage de jeu_de_la_vie_cli.JeuDeLaVieCLI:



Fonctions membres publiques

— None __init__ (self, list[list[Literal[1]|Literal[0]]] plateau=[ ])
— None set_symbole_mort (self, Any symb_mort)
— None set_symbole_vivant (self, Any symb_vivant)
— bool set_tableau (self, Any plateau, Any vivant=1)
— str __repr__ (self)
— None affiche (self)
— Literal[0]|Literal[1] valeur_case (self, int i, int j)
— int total_voisins (self, int i, int j)
— Literal[0]|Literal[1] resultat (self, int i, int j)
— None tour (self)
— bool arret_automatique (self)
— None run (self, int nombre_tours, float delai)
— bool importe_template (self, str fichier)

Fonctions membres publiques statiques

- bool est_tableau_valide (Any plateau)
- bool meurt (int nb_voisins)
- bool nait (int nb_voisins)

Attributs publics

- list tableau_precedent = [ ]
- list[list[Literal[1]|Literal[0]]] tableau = plateau
- str symbole_mort = "□"
- str symbole_vivant = "■"
- list tableau = vivant:
- str tableau = str(cellule) + " "
- list[list[Literal[1]|Literal[0]]] tableau_precedent = deepcopy(self.tableau)
- int tableau = 0

6.4.1 Description détaillée

```
Hérite de:
    object
Rôle:
    Représente le jeu de la vie.
```

Définition à la ligne 19 du fichier jeu_de_la_vie_cli.py.

6.4.2 Documentation des constructeurs et destructeur

__init__()

```
None jeu_de_la_vie_cli.JeuDeLaVieCLI.__init__ (
              self,
             list[list[Literal[1] | Literal[0]]]  plateau = [])
```

```
Entrées:
    self: JeuDeLaVieCLI
    tableau: list[list[Literal[1] | Literal[0]]]
        valeur par défaut: []
Sortie:
    None (modification en place)
Rôle:
    Construit un nouvel objet JeuDeLaVieCLI.
```

Définition à la ligne 27 du fichier jeu_de_la_vie_cli.py.

```
00028         -> None:
00029         """
00030         Entrées:
00031             self: JeuDeLaVieCLI
00032             tableau: list[list[Literal[1] | Literal[0]]]
00033                 valeur par défaut: []
00034         Sortie:
00035             None (modification en place)
00036         Rôle:
00037             Construit un nouvel objet JeuDeLaVieCLI.
00038         """
00039         # Déclaration d'un tableau au cycle n-1, initialisé vide
00040         self.tableau_precedent: list[list[Literal[1] | Literal[0]]] = []
00041         # Déclaration de tableau et l'initialise à plateau
00042         self.tableau: list[list[Literal[1] | Literal[0]]] = plateau
00043         # Déclaration de symbol_mort et l'initialise à □
00044         self.symbole_mort = "□"
00045         # Déclaration de symbole_vivant et l'initialise à ■
00046         self.symbole_vivant = "■"
00047
```

6.4.3 Documentation des fonctions membres

__repr__()

```
str jeu_de_la_vie_cli.JeuDeLaVieCLI.__repr__ (
                self)
```

```
Entrées:
    self: JeuDeLaVieCLI
Sortie:
    str
Rôle:
    Retourne une représentation en chaîne de caractères de manière
    simple (matrice de 1 et de 0).
```

Définition à la ligne 144 du fichier jeu_de_la_vie_cli.py.

```
00144     def __repr__(self) -> str:
00145         """
00146         Entrées:
00147             self: JeuDeLaVieCLI
00148         Sortie:
00149             str
00150         Rôle:
00151             Retourne une représentation en chaîne de caractères de manière
00152             simple (matrice de 1 et de 0).
00153         """
00154         reponse = ""
00155         # Pour chaque ligne du tableau
00156         for ligne in self.tableau:
00157             # Pour chaque cellule de la ligne
00158             for cellule in ligne:
00159                 # On ajoute la valeur de cellule en chaîne de caractères plus
00160                 # un espace
00161                 reponse += str(cellule) + " "
00162             # On ajoute un retour à la ligne
00163             reponse += "\n"
00164
00165         # On retourne la chaîne de caractère
00166         return reponse
00167
```

affiche()

```
None jeu_de_la_vie_cli.JeuDeLaVieCLI.affiche (
                self)
```

```
Entrées:
    self: JeuDeLaVieCLI
Sortie:
    None (affichage)
Rôle:
    Afficher de manière complexe un tableau de cellules remplacées
    par des caractères.
```

Définition à la ligne 168 du fichier jeu_de_la_vie_cli.py.

```
00168     def affiche(self) -> None:
00169         """
00170         Entrées:
00171             self: JeuDeLaVieCLI
00172         Sortie:
00173             None (affichage)
00174         Rôle:
00175             Afficher de manière complexe un tableau de cellules remplacées
00176             par des caractères.
00177         """
00178         # On efface le terminal
00179         system("cls")
00180         # crée la variable temporaire tab
00181         tab = self.tableau
00182         # pour chaque ligne du tableau
00183         for ligne in tab:
00184             # on crée une variable temporaire pour chaque ligne
00185             ligne_affiche = []
00186             for valeur in ligne:
00187                 # si la valeur dans le tableau est 0 on ajoute
00188                 # self.symbole_mort à la variable temporaire
00189                 if valeur == 0:
00190                     ligne_affiche.append(self.symbole_mort)
00191                 # si la valeur dans le tableau est 1 on ajoute
00192                 # self.symbole_vivant à la variable temporaire
00193                 else:
00194                     ligne_affiche.append(self.symbole_vivant)
00195             # On affiche la variable temporaire en enlevant les ""
00196             print(" ".join(ligne_affiche))
00197         # On affiche une ligne vide
00198         print()
00199
```

arret_automatique()

```
bool jeu_de_la_vie_cli.JeuDeLaVieCLI.arret_automatique (
              self)
```

```
Entrée:
    self: JeuDeLaVieCLI
Sortie:
    bool
Rôle:
    Renvoie True si le plateau est identique deux tours de suite, False
    sinon.
```

Définition à la ligne 334 du fichier jeu_de_la_vie_cli.py.

```
00334     def arret_automatique(self) -> bool:
00335         """
00336         Entrée:
00337             self: JeuDeLaVieCLI
00338         Sortie:
00339             bool
00340         Rôle:
00341             Renvoie True si le plateau est identique deux tours de suite, False
00342             sinon.
00343         """
00344         # retourne True si le tableau n'a pas changé entre deux cycles
00345         return self.tableau_precedent == self.tableau
00346
```

est_tableau_valide()

```
bool jeu_de_la_vie_cli.JeuDeLaVieCLI.est_tableau_valide (
             Any plateau)  [static]
```

```
Entrée:
    plateau: list[list[Any]]
Sortie:
    bool (validité)
Rôle:
    Vérifie la capacité du plateau à être un tableau.
```

Définition à la ligne 75 du fichier jeu_de_la_vie_cli.py.

```
00075     def est_tableau_valide(plateau: Any) -> bool:
00076         """
00077         Entrée:
00078             plateau: list[list[Any]]
00079         Sortie:
00080             bool (validité)
00081         Rôle:
00082             Vérifie la capacité du plateau à être un tableau.
00083         """
00084         # Déclaration de valide et initialisation sur False
00085         valide = True
00086 
00087         # Si plateau est une liste
00088         if type(plateau) is list:
00089             # Déclaration d'un itérateur
00090             i = 0
00091             # Pour chaque valeur de plateau tant que valide vaut True
00092             while i < len(plateau) and valide:
00093                 # Si plateau à l'indice i n'est pas une liste
00094                 if type(plateau[i]) is not list:
00095                     # On passe valide à False
00096                     valide = False
00097                 # Incrémentation de l'itérateur
00098                 i += 1
00099         # Si plateau n'est pas une liste
00100         else:
00101             # On passe valide à False
00102             valide = False
00103 
00104         # Retourne la valeur de valide
00105         return valide
00106 
```

importe_template()

```
bool jeu_de_la_vie_cli.JeuDeLaVieCLI.importe_template (
              self,
             str fichier)
```

```
Entrées:
    self: JeuDeLaVieCLI
    fichier: str
Sortie:
    bool (validité)
Rôle:
    Importe le template si celui-ci est valide.
```

Définition à la ligne 381 du fichier jeu_de_la_vie_cli.py.

```
00381     def importe_template(self, fichier: str) -> bool:
00382         """
00383         Entrées:
00384             self: JeuDeLaVieCLI
00385             fichier: str
00386         Sortie:
00387             bool (validité)
00388         Rôle:
```

```
00389             Importe le template si celui-ci est valide.
00390         """
00391         # Si le fichier existe
00392         if exists(fichier):
00393             # On ouvre le fichier en mode lecture
00394             with open(file=fichier, mode='r', encoding='utf8') as f:
00395                 # On extrait les données
00396                 template = f.readlines()
00397                 # On ferme le fichier
00398                 f.close()
00399             # Si le template est valide
00400             valide, erreur = est_template_valide(template)
00401             if valide:
00402                 # Attribut à tableau une matrice vide
00403                 self.tableau = []
00404                 # Pour chaque ligne
00405                 for i, ligne in enumerate(reader(template)):
00406                     # On ajoute une nouvelle ligne à la matrice
00407                     self.tableau.append([])
00408                     # Pour chaque élément de la ligne
00409                     for element in ligne:
00410                         # On ajoute l'élément converti en int
00411                         self.tableau[i].append(int(element))
00412                 # On retoure True
00413                 return True
00414             else:
00415                 print(erreur)
00416         # On retourne False
00417         return False
00418
00419
00420 # Si le présent fichier est executé avec python 3.10 ou plus
```

meurt()

```
bool jeu_de_la_vie_cli.JeuDeLaVieCLI.meurt (
             int nb_voisins)  [static]
```

```
Entrée:
    nb_voisins: int
Sortie:
    bool
Rôle:
    Retourne True si la cellule meurt
```

Définition à la ligne 252 du fichier jeu_de_la_vie_cli.py.

```
00252     def meurt(nb_voisins: int) -> bool:
00253         """
00254         Entrée:
00255             nb_voisins: int
00256         Sortie:
00257             bool
00258         Rôle:
00259             Retourne True si la cellule meurt
00260         """
00261         # Retourne True si le nombre de voisins est différent de 2 ou 3
00262         return nb_voisins < 2 or nb_voisins > 3
00263
```

nait()

```
bool jeu_de_la_vie_cli.JeuDeLaVieCLI.nait (
             int nb_voisins)  [static]
```

```
Entrée:
    nb_voisins: int
Sortie:
    bool
Rôle:
    Retourne True si la cellule naît
```

Définition à la ligne 265 du fichier jeu_de_la_vie_cli.py.

```
00265     def nait(nb_voisins: int) -> bool:
00266         """
00267         Entrée:
00268             nb_voisins: int
00269         Sortie:
00270             bool
00271         Rôle:
00272             Retourne True si la cellule naît
00273         """
00274         # Retourne True si le nombre de voisins est égal à 3
00275         return nb_voisins == 3
00276
```

resultat()

```
Literal[0] | Literal[1] jeu_de_la_vie_cli.JeuDeLaVieCLI.resultat (
              self,
             int i,
             int j)
```

```
Entrées:
    self: JeuDeLaVieCLI
    i: int (ordonnée)
    j: int (abscisse)
Sortie:
    Literal[0] | Literal[1] (Etat de la cellule)
Rôle:
    Retourne l'état de la cellule en fonction du nombre de voisins
```

Définition à la ligne 277 du fichier jeu_de_la_vie_cli.py.

```
00277     def resultat(self, i: int, j: int) -> Literal[0] | Literal[1]:
00278         """
00279         Entrées:
00280             self: JeuDeLaVieCLI
00281             i: int (ordonnée)
00282             j: int (abscisse)
00283         Sortie:
00284             Literal[0] | Literal[1] (Etat de la cellule)
00285         Rôle:
00286             Retourne l'état de la cellule en fonction du nombre de voisins
00287         """
00288         # On calcul le nombre de voisin
00289         nb_voisins = self.total_voisins(i, j)
00290         # si la cellule == 1 (vivante)
00291         if self.tableau[i][j]:
00292             # si la cellule meurt
00293             if self.meurt(nb_voisins):
00294                 # On met à jour la valeur dans le faux tableau
00295                 return 0
00296             # Sinon
00297             else:
00298                 # On retourne la valeur de (i;j)
00299                 return 1
00300         # si la cellule == 0 (morte)
00301         else:
00302             # si la cellule naît
00303             if self.nait(nb_voisins):
00304                 # On met à jour la valeur dans le faux tableau
00305                 return 1
00306             # Sinon
00307             else:
00308                 # On retourne la valeur de (i;j)
00309                 return 0
00310
```

run()

```
None jeu_de_la_vie_cli.JeuDeLaVieCLI.run (
              self,
             int nombre_tours,
             float delai)
```

```
Entrées:
    self: JeuDeLaVieCLI
    nombre_tours: int
    delai: float
Sortie:
    None (modification en place)
Rôle:
    Effectue nombre_tours cycles de delai seconde(s).
```

Définition à la ligne 347 du fichier jeu_de_la_vie_cli.py.

```
00347     def run(self, nombre_tours: int, delai: float) -> None:
00348         """
00349         Entrées:
00350             self: JeuDeLaVieCLI
00351             nombre_tours: int
00352             delai: float
00353         Sortie:
00354             None (modification en place)
00355         Rôle:
00356             Effectue nombre_tours cycles de delai seconde(s).
00357         """
00358         # Si le tableau n'est pas vide
00359         if self.tableau:
00360             # On déclare un itérateur i
00361             i = 0
00362             # se répète en fonction du nombre de tour
00363             while i < nombre_tours:
00364                 # si 2 tours à la suite sont identique
00365                 if self.arret_automatique():
00366                     # on stop la boucle
00367                     i = nombre_tours - 1
00368                 # sinon la boucle réactualise eu prochain tour
00369                 else:
00370                     # affiche la matrice de JeuDeLaVieCLI
00371                     self.affiche()
00372                     # actualise la matrice
00373                     self.tour()
00374                     # laisse un temps d'attente entre chaque tour
00375                     sleep(delai)
00376                 # on incrémente n pour que la boucle ait une terminaison
00377                 i += 1
00378                 # Affiche le dernier tour de boucle
00379             self.affiche()
00380
```

set_symbole_mort()

```
None jeu_de_la_vie_cli.JeuDeLaVieCLI.set_symbole_mort (
              self,
             Any symb_mort)
```

```
Entrées:
    self: JeuDeLaVieCLI
    symb_mort: Any
Sortie:
    None (modification en place)
Rôle:
    Redéfinit le symbole des cases mortes (affichage_complexe).
```

Définition à la ligne 48 du fichier jeu_de_la_vie_cli.py.

```
00048     def set_symbole_mort(self, symb_mort: Any) -> None:
00049         """
00050         Entrées:
00051             self: JeuDeLaVieCLI
00052             symb_mort: Any
00053         Sortie:
00054             None (modification en place)
00055         Rôle:
00056             Redéfinit le symbole des cases mortes (affichage_complexe).
00057         """
00058         # définie le nouveau symbole d'une cellule morte
00059         self.symbole_mort = str(symb_mort)
00060
```

set_symbole_vivant()

```
None jeu_de_la_vie_cli.JeuDeLaVieCLI.set_symbole_vivant (
              self,
             Any symb_vivant)


Entrées:
    self: JeuDeLaVieCLI
    symb_vivant: Any
Sortie:
    None (modification en place)
Rôle:
    Redéfinit le symbole des cases vivantes (affichage_complexe).
```

Définition à la ligne 61 du fichier jeu_de_la_vie_cli.py.

```
00061     def set_symbole_vivant(self, symb_vivant: Any) -> None:
00062         """
00063         Entrées:
00064             self: JeuDeLaVieCLI
00065             symb_vivant: Any
00066         Sortie:
00067             None (modification en place)
00068         Rôle:
00069             Redéfinit le symbole des cases vivantes (affichage_complexe).
00070         """
00071         # définie le nouveau symbole d'une cellule morte
00072         self.symbole_vivant = str(symb_vivant)
00073
```

set_tableau()

```
bool jeu_de_la_vie_cli.JeuDeLaVieCLI.set_tableau (
              self,
             Any plateau,
             Any  vivant = 1)


Entrées:
    self: JeuDeLaVieCLI
    plateau: list[list[Any]] (attendu)
    vivant: Any
        valeur par défaut: 1
Sortie:
    bool (validité)
Rôle:
    Redéfinit l'attribut tableau de même dimension que le matrice. Si
    l'élément vaut vivant, son état sera Vivant, sinon Mort.
```

Définition à la ligne 107 du fichier jeu_de_la_vie_cli.py.

```
00107     def set_tableau(self, plateau: Any, vivant: Any = 1) -> bool:
00108         """
00109         Entrées:
00110             self: JeuDeLaVieCLI
00111             plateau: list[list[Any]] (attendu)
00112             vivant: Any
00113                 valeur par défaut: 1
00114         Sortie:
00115             bool (validité)
00116         Rôle:
00117             Redéfinit l'attribut tableau de même dimension que le matrice. Si
00118             l'élément vaut vivant, son état sera Vivant, sinon Mort.
00119         """
00120         # Si le plateau est apte à devenir tableau
00121         if self.est_tableau_valide(plateau):
00122             # On vide tableau
00123             self.tableau = []
00124             # Pour chaque indice de plateau
00125             for i in range(len(plateau)):
00126                 # On ajoute une ligne à tableau
```

```
00127                 self.tableau.append([])
00128                 # Pour chaque indice de plateau à l'indice i
00129                 for j in range(len(plateau[i])):
00130                     # Si l'élément (i;j) de plateau est vivant
00131                     if plateau[i][j] == vivant:
00132                         # On ajoute 1 à tableau (vivant)
00133                         self.tableau[i].append(1)
00134                     # Sinon
00135                     else:
00136                         # On ajoute 0 à tableau (mort)
00137                         self.tableau[i].append(0)
00138             # On retourne True car la procédure c'est bien déroulée
00139             return True
00140 
00141         # On retourne False car le plateau ne peut être utilisé comme tableau
00142         return False
00143 
```

total_voisins()

```
int jeu_de_la_vie_cli.JeuDeLaVieCLI.total_voisins (
              self,
             int i,
             int j)


Entrées:
    self: JeuDeLaVieCLI
    i: int
    j: int
Sortie:
    int
Rôle:
    Retourne le total de voisins de la cellule (i;j)
```

Définition à la ligne 220 du fichier jeu_de_la_vie_cli.py.

```
00220     def total_voisins(self, i: int, j: int) -> int:
00221         """
00222         Entrées:
00223             self: JeuDeLaVieCLI
00224             i: int
00225             j: int
00226         Sortie:
00227             int
00228         Rôle:
00229             Retourne le total de voisins de la cellule (i;j)
00230         """
00231         # Récupère le voisin du bas
00232         b = self.valeur_case(i + 1, j)
00233         # Récupère le voisin du bas droit
00234         bd = self.valeur_case(i + 1, j + 1)
00235         # Récupère le voisin du bas gauche
00236         bg = self.valeur_case(i + 1, j - 1)
00237         # Récupère le voisin du haut
00238         h = self.valeur_case(i - 1, j)
00239         # Récupère le voisin du haut droit
00240         hd = self.valeur_case(i - 1, j + 1)
00241         # Récupère le voisin du haut gauche
00242         hg = self.valeur_case(i - 1, j - 1)
00243         # Récupère le voisin de droite
00244         d = self.valeur_case(i, j + 1)
00245         # Récupère le voisin de gauche
00246         g = self.valeur_case(i, j - 1)
00247 
00248         # retourne la somme des voisins
00249         return b + bd + bg + h + hd + hg + d + g
00250 
```

tour()

```
None jeu_de_la_vie_cli.JeuDeLaVieCLI.tour (
              self)
```

```
Entrée:
    self: JeuDeLaVieCLI
Sortie:
    None (modification en place)
Rôle:
    Execute un tour du jeu.
```

Définition à la ligne 311 du fichier jeu_de_la_vie_cli.py.

```
00311     def tour(self) -> None:
00312         """
00313         Entrée:
00314             self: JeuDeLaVieCLI
00315         Sortie:
00316             None (modification en place)
00317         Rôle:
00318             Execute un tour du jeu.
00319         """
00320         # On copie le tableau pour pouvoir geler le vrai pour faire les modifs
00321         self.tableau_precedent = deepcopy(self.tableau)
00322         # On déclare tableau comme une copie de l'attribut tableau (gèle)
00323         tableau = deepcopy(self.tableau)
00324 
00325         # on itère dans les lignes
00326         for i in range(len(self.tableau)):
00327             # on itère dans la ligne i
00328             for j in range(len(self.tableau[i])):
00329                 tableau[i][j] = self.resultat(i, j)
00330 
00331         # On attribut le tableau gelé au vrai tableau
00332         self.tableau = tableau
00333 
```

valeur_case()

```
Literal[0] | Literal[1] jeu_de_la_vie_cli.JeuDeLaVieCLI.valeur_case (
               self,
              int i,
              int j)
```

```
Entrées:
    self: JeuDeLaVieCLI
    i: int
    j: int
Sortie:
    Literal[0] | Literal[1]
Rôle:
    Donner l'état d'une case (1 ou 0).
```

Définition à la ligne 200 du fichier jeu_de_la_vie_cli.py.

```
00200     def valeur_case(self, i: int, j: int) -> Literal[0] | Literal[1]:
00201         """
00202         Entrées:
00203             self: JeuDeLaVieCLI
00204             i: int
00205             j: int
00206         Sortie:
00207             Literal[0] | Literal[1]
00208         Rôle:
00209             Donner l'état d'une case (1 ou 0).
00210         """
00211         # si les indices décrivent une valeur du tableau
00212         if 0 <= i < len(self.tableau) and 0 <= j < len(self.tableau[0]):
00213             # On retourne la valeur de la case (i;j)
00214             return self.tableau[i][j]
00215         # si l'indice i ou j est trop grand ou négatif
00216         else:
00217             # on retourne 0
00218             return 0
00219 
```

6.4.4 Documentation des données membres

symbole_mort

```
jeu_de_la_vie_cli.JeuDeLaVieCLI.symbole_mort = "□"
```

Définition à la ligne 44 du fichier jeu_de_la_vie_cli.py.

symbole_vivant

```
jeu_de_la_vie_cli.JeuDeLaVieCLI.symbole_vivant = "■"
```

Définition à la ligne 46 du fichier jeu_de_la_vie_cli.py.

tableau [1/4]

```
list[list[Literal[1] | Literal[0]]] jeu_de_la_vie_cli.JeuDeLaVieCLI.tableau = plateau
```

Définition à la ligne 42 du fichier jeu_de_la_vie_cli.py.

tableau [2/4]

```
list jeu_de_la_vie_cli.JeuDeLaVieCLI.tableau = vivant:
```

Définition à la ligne 123 du fichier jeu_de_la_vie_cli.py.

tableau [3/4]

```
str jeu_de_la_vie_cli.JeuDeLaVieCLI.tableau = str(cellule) + " "
```

Définition à la ligne 156 du fichier jeu_de_la_vie_cli.py.

tableau [4/4]

```
int jeu_de_la_vie_cli.JeuDeLaVieCLI.tableau = 0
```

Définition à la ligne 359 du fichier jeu_de_la_vie_cli.py.

tableau_precedent [1/2]

```
list jeu_de_la_vie_cli.JeuDeLaVieCLI.tableau_precedent = []
```

Définition à la ligne 40 du fichier jeu_de_la_vie_cli.py.

tableau_precedent [2/2]

```
list[list[Literal[1] | Literal[0]]] jeu_de_la_vie_cli.JeuDeLaVieCLI.tableau_precedent = deepcopy(self.↩
tableau)
```

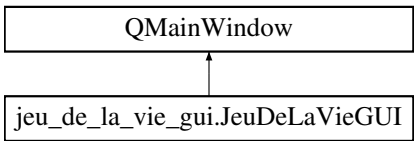
Définition à la ligne 321 du fichier jeu_de_la_vie_cli.py.

La documentation de cette classe a été générée à partir du fichier suivant :

— D:/Jeu de la vie/source/jeu_de_la_vie_cli.py

6.5 Référence de la classe jeu_de_la_vie_gui.JeuDeLaVieGUI

Graphe d'héritage de jeu_de_la_vie_gui.JeuDeLaVieGUI:



Fonctions membres publiques

— None __init__ (self)

— bool event (self, QEvent even)

— None enregistrer (self)

— None enregistrer_sous (self)

— None importe_template (self)

— None set_auto_grandissement (self, Qt.CheckState etat)

— None set_auto_stop (self, Qt.CheckState etat)

— None set_periode (self, float valeur)

— None lance_anim (self)

— None stop_anim (self)

— None zoom_in (self)

— None zoom_out (self)

Attributs publics

— str chemin_absolu = __file__[:-len(basename(__file__))]

— bool est_fichier_ouvert = False

— str fichier = ""

— int nb_cycle = 0

— widget_central = QWidget(self)

— affichage = QHBoxLayout()

— QMenuBar menu = self.menuBar()

— QMenu menu_fichier = self.menu.addMenu("&Fichier")

— importer_template = QAction(self)

— importe_template

— QAction enregistrer_template = QAction(self)

— enregistrer

— enregistrer_template_sous = QAction(self)

— enregistrer_sous

— scene = Scene(self, QSize(10, 10))

— vue = QGraphicsView(self)

— outils_layout = QVBoxLayout()

— affichage_cycle = QLabel(self)

— controles_titre = QLabel(self)

— controles_layout = QHBoxLayout()

— jouer = QPushButton(self)

— lance_anim

— pause = QPushButton(self)

— stop_anim

— tour_par_tour = QPushButton(self)

— periode_layout = QGridLayout()

— periode_label = QLabel(self)

— periode_entree = QDoubleSpinBox(self)

— set_periode

— ligne_post_controles = QFrame(self)

— auto_grandissement_entree = QCheckBox(self)

— set_auto_grandissement

— auto_stop_entree = QCheckBox(self)

— set_auto_stop

— ligne_post_autos = QFrame(self)

- zoom = QGridLayout()
- zoom_label = QLabel(self)
- zoom_in_entree = QPushButton(self)
- zoom_in
- zoom_out_entree = QPushButton(self)
- zoom_out
- bar_info_titre = QLabel(self)
- bar_info = QLabel(self)

6.5.1 Description détaillée

```
Hérite de:
    QMainWindow
Rôle:
    Représente l'application du jeu de la vie
```

Définition à la ligne 54 du fichier jeu_de_la_vie_gui.py.

6.5.2 Documentation des constructeurs et destructeur

__init__()

```
None jeu_de_la_vie_gui.JeuDeLaVieGUI.__init__ (
              self)


Entrée:
    self: JeuDeLaVieGUI
Sortie:
    None (ctor)
Rôle:
    Construit un nouvel objet JeuDeLaVieGUI.
```

Définition à la ligne 61 du fichier jeu_de_la_vie_gui.py.

```
00061     def __init__(self) -> None:
00062         """
00063         Entrée:
00064             self: JeuDeLaVieGUI
00065         Sortie:
00066             None (ctor)
00067         Rôle:
00068             Construit un nouvel objet JeuDeLaVieGUI.
00069         """
00070         # Initialisation de la classe mère
00071         QMainWindow.__init__(self)
00072 
00073         # Attributs
00074 
00075         # Déclaration de chemin_absolu pour accéder aux ressources (svg, ...)
00076         self.chemin_absolu: str = __file__[:-len(basename(__file__))]
00077         # Déclaration d'un booléan est_fichier_ouvert
00078         self.est_fichier_ouvert: bool = False
00079         self.fichier: str = ""
00080         # Déclaration d'un attribut nb_cycle
00081         self.nb_cycle: int = 0
00082 
00083         # Fenêtre
```

```
00084
00085         # Définit une taille minimum pour la fenêtre
00086         self.setMinimumSize(710, 400)
00087         # Redimensionne la fenêtre
00088         self.resize(self.screen().geometry().width() - 400,
00089                     self.screen().geometry().height() - 300)
00090         # Replace la fenêtre
00091         self.move(
00092             (self.screen().geometry().width() - self.width()) // 2,
00093             (self.screen().geometry().height() - self.height()) // 2 - 50)
00094         # On créer widget_central
00095         self.widget_central = QWidget(self)
00096         # On définit widget_central comme le widget principal
00097         # (celui qui prend toute la place)
00098         self.setCentralWidget(self.widget_central)
00099         # On créer le layout principal
00100         self.affichage = QHBoxLayout()
00101         # On associe le layout au widget car setCentralLayout n'existe pas
00102         self.widget_central.setLayout(self.affichage)
00103
00104         # Menu bar
00105
00106         # Déclaration d'un attribut menuBar
00107         self.menu: QMenuBar = self.menuBar()
00108         # Déclaration d'un attribut menu fichier
00109         self.menu_fichier: QMenu = self.menu.addMenu("&Fichier")
00110         # Déclaration de importer_template
00111         self.importer_template = QAction(self)
00112         # Définition du texte de importer_template
00113         self.importer_template.setText("Importer un template (.csv)")
00114         # Définition d'un raccourci clavier
00115         self.importer_template.setShortcut(QKeySequence.StandardKey.Open)
00116         # Relie le signal à la méthode importe
00117         self.importer_template.triggered.connect(self.importe_template)
00118         # Ajoute l'action au menu
00119         self.menu_fichier.addAction(self.importer_template)
00120
00121         # Déclaration de enregistrer_template
00122         self.enregistrer_template: QAction = QAction(self)
00123         # Définition du texte de enregistrer_template
00124         self.enregistrer_template.setText("Enregistrer le template")
00125         # Définition d'un raccourci clavier
00126         self.enregistrer_template.setShortcut(QKeySequence.StandardKey.Save)
00127         # Relie le signal à la méthode enregistrer
00128         self.enregistrer_template.triggered.connect(self.enregistrer)
00129         # Ajoute l'action au menu
00130         self.menu_fichier.addAction(self.enregistrer_template)
00131
00132         # Déclaration de enregistrer_template_sous
00133         self.enregistrer_template_sous = QAction(self)
00134         # Définition du texte de enregistrer_template
00135         self.enregistrer_template_sous.setText("Enregistrer sous le template")
00136         # Définition d'un raccourci clavier
00137         self.enregistrer_template_sous.setShortcut(
00138             QKeySequence.StandardKey.SaveAs)
00139         # Relie le signal à la méthode enregistrer
00140         self.enregistrer_template_sous.triggered.connect(self.enregistrer_sous)
00141         # Ajoute l'action au menu
00142         self.menu_fichier.addAction(self.enregistrer_template_sous)
00143
00144         # Scène - vue
00145
00146         # On créer une scène du jeu de la vie
00147         self.scene = Scene(self, QSize(10, 10))
00148         # On créer une vue
00149         self.vue = QGraphicsView(self)
00150         # Définition d'un conseil pour l'utilisateur
00151         self.vue.setStatusTip(
00152             "Scène du jeu de la vie. Vous pouvez zommer, vous y déplacer et " +
00153             "modifier l'état des cellules en cliquant dessus")
00154         self.vue.enterEvent
00155         # On dit à la vue quelle scène afficher
00156         # Fonctionnement -> https://doc.qt.io/qt-6/graphicsview.html
00157         self.vue.setScene(self.scene)
00158         # On définit l'échelle (fonctionne par pourcentage d'agrandissement)
00159         self.vue.scale(0.5, 0.5)
00160         # On ajoute la vue au layout principal
00161         self.affichage.addWidget(self.vue)
00162
00163         # Bar d'outils
00164
00165         # Déclaration d'un layout vertical outils_layout, layout ≃ div en html
00166         self.outils_layout = QVBoxLayout()
00167         # On définit les options d'alignement du layout
00168         self.outils_layout.setAlignment(
00169             Qt.AlignmentFlag.AlignHCenter | Qt.AlignmentFlag.AlignTop)
00170         # On ajoute outils_layout au layout principal
```

```
00171         self.affichage.addLayout(self.outils_layout)
00172
00173         # Déclaration d'un label
00174         self.affichage_cycle = QLabel(self)
00175         # Définition d'un conseil pour l'utilisateur
00176         self.affichage_cycle.setStatusTip("Compte le nombre de cycle effectué")
00177         # Modifie le texe du label
00178         self.affichage_cycle.setText("Cycle n°" + str(self.nb_cycle))
00179         # Définit l'alignement du texte
00180         self.affichage_cycle.setAlignment(Qt.AlignmentFlag.AlignRight)
00181         # Ajoute le label dans le layout
00182         self.outils_layout.addWidget(self.affichage_cycle)
00183         # On ajoute un peu d'espace entre le haut de la fenêtre et les boutons
00184         self.outils_layout.addSpacing(20)
00185
00186         # Déclaration d'un label
00187         self.controles_titre = QLabel(self)
00188         # Définition d'un conseil pour l'utilisateur
00189         self.controles_titre.setStatusTip("Contrôle du jeu de l'animation.")
00190         # Définitons du texte du label
00191         self.controles_titre.setText("Contrôles :")
00192         # Définition du style du texte et de sa taille
00193         self.controles_titre.setStyleSheet(
00194             "font-style: bold; font-size: 11pt;")
00195         # Ajoute le titre à outils_layout
00196         self.outils_layout.addWidget(self.controles_titre)
00197         # Déclaration d'un layout horizontal jouer/pause
00198         self.controles_layout = QHBoxLayout()
00199         # Déclaration d'un bouton jouer
00200         self.jouer = QPushButton(self)
00201         # Définition d'un conseil pour l'utilisateur
00202         self.jouer.setStatusTip("Joue l'animation.")
00203         # Définition de l'icone de jouer
00204         icone_jouer = QIcon(self.chemin_absolu +
00205                             "assets\\icones\\play-button.svg")
00206         # Attribution de l'icone au bouton
00207         self.jouer.setIcon(icone_jouer)
00208         # Relie le signal à la méthode run
00209         self.jouer.clicked.connect(self.lance_anim)
00210         # Modification du style de jouer
00211         self.jouer.setStyleSheet("padding-top: 7px; padding-bottom: 7px")
00212         # Modification automatique de la taille
00213         self.jouer.adjustSize()
00214         # On ajoute jouer à controles_layout
00215         self.controles_layout.addWidget(self.jouer)
00216         # Déclaration d'un bouton pause
00217         self.pause = QPushButton(self)
00218         # Définition d'un conseil pour l'utilisateur
00219         self.pause.setStatusTip("Pause l'animation")
00220         # Définition de l'icone de pause
00221         icone_pause = QIcon(self.chemin_absolu +
00222                             "assets\\icones\\pause-button.svg")
00223         # Attribution de l'icone au bouton
00224         self.pause.setIcon(icone_pause)
00225         # Relie le signal à la méthode stop
00226         self.pause.clicked.connect(self.stop_anim)
00227         # Désactive le bouton pause
00228         self.pause.setEnabled(False)
00229         # Modification du style de pause
00230         self.pause.setStyleSheet("padding-top: 7px; padding-bottom: 7px")
00231         # Modification automatique de la taille
00232         self.pause.adjustSize()
00233         # On ajoute pause à controles_layout
00234         self.controles_layout.addWidget(self.pause)
00235         # On ajoute controles_layout à outils_layout
00236         self.outils_layout.addLayout(self.controles_layout)
00237
00238         # Déclaration du bouton tour par tour
00239         self.tour_par_tour = QPushButton(self)
00240         # Définition d'un conseil pour l'utilisateur
00241         self.tour_par_tour.setStatusTip("Exécute un tour de l'animation")
00242         # On définit le texte à afficher
00243         self.tour_par_tour.setText("Tour par tour")
00244         # Relie le signal à la méthode tour de la scène
00245         self.tour_par_tour.clicked.connect(self.scene.tour)
00246         # On ajoute tour_par_tour au layout du menu
00247         self.outils_layout.addWidget(self.tour_par_tour)
00248         # Ajoute de l'espace après
00249
00250         # Déclaration de periode_layout
00251         self.periode_layout = QGridLayout()
00252         # Déclaration de periode_label
00253         self.periode_label = QLabel(self)
00254         # Définition d'un conseil pour l'utilisateur
00255         self.periode_label.setStatusTip(
00256             "La période est la durée d'un cycle de l'animation.")
00257         # Définition du texte de periode_label
```

```
00258         self.periode_label.setText("Durée d'une période :")
00259         # Déclaration de periode_entree
00260         self.periode_entree = QDoubleSpinBox(self)
00261         # Définition d'un conseil pour l'utilisateur
00262         self.periode_entree.setStatusTip(
00263             "Une période trop courte peut causer des ralentissements.")
00264         # On définit la valeur minimal de periode_entree
00265         self.periode_entree.setMinimum(0.01)
00266         # Définition de la valeur de periode_entree
00267         self.periode_entree.setValue(0.5)
00268         # Définition du pas de periode_entree
00269         self.periode_entree.setSingleStep(0.1)
00270         # On définit le suffix de periode_entree
00271         self.periode_entree.setSuffix(" sec")
00272         # On définit la largeur minimal de periode_entree
00273         self.periode_entree.setMinimumWidth(120)
00274         # Relie le signal à la méthode set_periode
00275         self.periode_entree.valueChanged.connect(self.set_periode)
00276         # Ajoute periode_label dans la cellule (0;0) du layout
00277         self.periode_layout.addWidget(self.periode_label, 0, 0)
00278         # Ajoute periode_entree dans la cellule (0;1) du layout
00279         self.periode_layout.addWidget(self.periode_entree, 0, 1)
00280         # Ajoute periode_layout à outils_layout
00281         self.outils_layout.addLayout(self.periode_layout)
00282         # Ajoute de l'espace après
00283         self.outils_layout.addSpacing(7)
00284         # Déclaration d'une ligne de séparation
00285         self.ligne_post_controles = QFrame(self)
00286         # Définition de la forme
00287         self.ligne_post_controles.setFrameShape(QFrame.Shape.HLine)
00288         # Définiton des ombres
00289         self.ligne_post_controles.setFrameShadow(QFrame.Shadow.Sunken)
00290         # Ajoute la ligne à outils_layout
00291         self.outils_layout.addWidget(self.ligne_post_controles)
00292         # Ajoute de l'espace après
00293         self.outils_layout.addSpacing(7)
00294
00295         # Déclaration d'une case à cocher
00296         self.auto_grandissement_entree = QCheckBox(self)
00297         # Définition d'un conseil pour l'utilisateur
00298         self.auto_grandissement_entree.setStatusTip("Quand est cochée, " +
00299             "agrandit le plateau pour s'adapter à l'animation")
00300         # Définition du texte de la case à cocher
00301         self.auto_grandissement_entree.setText("Auto grandissement")
00302         # On définit son état sur coché
00303         self.auto_grandissement_entree.setCheckState(Qt.CheckState.Checked)
00304         # Relie le signal à la fonction set_auto_grandissement
00305         self.auto_grandissement_entree.checkStateChanged.connect(
00306             self.set_auto_grandissement)
00307         # On ajoute la case à cocher au layout
00308         self.outils_layout.addWidget(self.auto_grandissement_entree)
00309
00310         # Déclaration d'une case à cocher
00311         self.auto_stop_entree = QCheckBox(self)
00312         # Définition d'un conseil pour l'utilisateur
00313         self.auto_stop_entree.setStatusTip(
00314             "Quand est cochée, arrête automatiquement l'animation quand il " +
00315             "n'y a plus de changement.")
00316         # Définition du texte de la case à cocher
00317         self.auto_stop_entree.setText("Auto stop")
00318         # On définit son état sur coché
00319         self.auto_stop_entree.setCheckState(Qt.CheckState.Checked)
00320         # Relie le signal à la fonction set_auto_stop
00321         self.auto_stop_entree.checkStateChanged.connect(
00322             self.set_auto_stop)
00323         # On ajoute la case à cocher au layout
00324         self.outils_layout.addWidget(self.auto_stop_entree)
00325         # Ajoute de l'espace après
00326         self.outils_layout.addSpacing(7)
00327         # Déclaration d'une ligne de séparation
00328         self.ligne_post_autos = QFrame(self)
00329         # Définition de la forme
00330         self.ligne_post_autos.setFrameShape(QFrame.Shape.HLine)
00331         # Définiton des ombres
00332         self.ligne_post_autos.setFrameShadow(QFrame.Shadow.Sunken)
00333         # Ajoute la ligne à outils_layout
00334         self.outils_layout.addWidget(self.ligne_post_autos)
00335         # Ajoute de l'espace après
00336         self.outils_layout.addSpacing(7)
00337
00338         # Déclaration d'un layout horizontal
00339         self.zoom = QGridLayout()
00340         # Déclaration d'un label zoom
00341         self.zoom_label = QLabel(self)
00342         # Définitions du text du label
00343         self.zoom_label.setText("Zoom :")
00344         # Définition du style du texte et de sa taille
```

```
00345         self.zoom_label.setStyleSheet("font-style: bold; font-size: 11pt;")
00346         # Définition d'un conseil pour l'utilisateur
00347         self.zoom_label.setStatusTip("Interface de zoom")
00348         # On ajoute zoom_label dans la cellule (0;0) sur un espace de 1 ligne
00349         # et 2 colonnes
00350         self.zoom.addWidget(self.zoom_label, 0, 0, 1, 2)
00351         # Déclaration d'un bouton zoom_in_entree (zoom in)
00352         self.zoom_in_entree = QPushButton(self)
00353         # Définition d'un conseil pour l'utilisateur
00354         self.zoom_in_entree.setStatusTip("Zoom dans la scène")
00355         # Définition du texte du bouton
00356         self.zoom_in_entree.setText("+")
00357         # Relie le signal à la méthode zoom_in
00358         self.zoom_in_entree.clicked.connect(self.zoom_in)
00359         # On ajoute zoom_in_entree au layout zoom
00360         self.zoom.addWidget(self.zoom_in_entree, 1, 0)
00361         # Déclaration d'un bouton zoom_out_entree (zoom out)
00362         self.zoom_out_entree = QPushButton(self)
00363         # Définition d'un conseil pour l'utilisateur
00364         self.zoom_out_entree.setStatusTip("Dézoom dans la scène")
00365         # Définition du texte du bouton
00366         self.zoom_out_entree.setText("-")
00367         # Relie le signal à la méthode zoom_out
00368         self.zoom_out_entree.clicked.connect(self.zoom_out)
00369         # On ajoute zoom_out_entree au layout zoom
00370         self.zoom.addWidget(self.zoom_out_entree, 1, 1)
00371         # On ajoute zoom au layout du menu
00372         self.outils_layout.addLayout(self.zoom)
00373         # On ajoute de l'étirement qui à pour effet de prendre le plus de place
00374         # possible, ainsi, les prochains éléments seront le plus bas possible
00375         self.outils_layout.addStretch()
00376 
00377         # Déclaration d'un titre pour bar_info
00378         self.bar_info_titre = QLabel(self)
00379         # Définition du texte
00380         self.bar_info_titre.setText("Vue Info :")
00381         # Définition du style du texte et de sa taille
00382         self.bar_info_titre.setStyleSheet("font-style: bold; font-size: 11pt;")
00383         # Ajoute le titre à outils_layout
00384         self.outils_layout.addWidget(self.bar_info_titre)
00385         # Déclaration de bar_info
00386         self.bar_info = QLabel(self)
00387         # Définit la taille minimum de bar_info en hauteur
00388         self.bar_info.setMinimumHeight(175)
00389         # On précise les marges du texte
00390         self.bar_info.setMargin(10)
00391         # Déclaration d'une Politique de taille et d'agrandissement
00392         bar_info_policy = QSizePolicy()
00393         # Spécifit que les demandes agrandissement sont ignorées
00394         # demandes effectuées à cause des marges
00395         bar_info_policy.setHorizontalPolicy(QSizePolicy.Policy.Ignored)
00396         # Attribution de la politique de taille à bar_info
00397         self.bar_info.setSizePolicy(bar_info_policy)
00398         # Définiton de l'attribut wordwrap sur True, pour les retous à la ligne
00399         self.bar_info.setWordWrap(True)
00400         # Précision de l'alignement du texte en haut à gauche
00401         self.bar_info.setAlignment(Qt.AlignmentFlag.AlignLeft |
00402                                    Qt.AlignmentFlag.AlignTop)
00403         # Définition de la taille du texte
00404         self.bar_info.setStyleSheet("font-size: 10.5pt;")
00405         # Définition du format du texte comme étant Markdown
00406         self.bar_info.setTextFormat(Qt.TextFormat.MarkdownText)
00407         # Sélection du style du cadre (classe mère que QLabel)
00408         self.bar_info.setFrameStyle(QFrame.Shape.StyledPanel)
00409         # Définition du statusTip (Cf méthode event)
00410         self.bar_info.setStatusTip(
00411             "La *Vue Info* fournit une brève description de l'élément " +
00412             "de l'interface utilisateur sur lequel la souris se trouve " +
00413             "actuellement.")
00414         # Ajoute bar_info à outils_layout
00415         self.outils_layout.addWidget(self.bar_info)
00416 
00417         # Initialisation de toutes les variables reliées aux champs d'entrées
00418 
00419         # Initialise auto_grandissement sur l'état de la case à cocher
00420         self.scene.auto_grandissement = bool(
00421             self.auto_grandissement_entree.checkState().value)
00422         # Initialise auto_stop sur l'état de la case à cocher
00423         self.scene.auto_stop = bool(self.auto_stop_entree.checkState().value)
00424         # Initialise periode sur la valeur de periode_entree
00425         self.scene.periode = int(self.periode_entree.value() * 1000)
00426 
00427         # Affiche la fenêtre
00428         self.show()
00429 
```

6.5.3 Documentation des fonctions membres

enregistrer()

```
None jeu_de_la_vie_gui.JeuDeLaVieGUI.enregistrer (
              self)
```

```
Entrée:
    self: JeuDeLaVieGUI
Sortie:
    None (modification en place)
Rôle:
    Enregistre le plateau au format csv.
```

Définition à la ligne 460 du fichier jeu_de_la_vie_gui.py.

```
00460     def enregistrer(self) -> None:
00461         """
00462         Entrée:
00463             self: JeuDeLaVieGUI
00464         Sortie:
00465             None (modification en place)
00466         Rôle:
00467             Enregistre le plateau au format csv.
00468         """
00469         # Si un fichier est "ouvert"
00470         if self.est_fichier_ouvert:
00471 
00472             # On ouvre le fichier (s'il n'existe plus, il est recréé)
00473             with open(self.fichier, 'w', encoding='utf-8') as f:
00474                 # Récupère le plateau avec 1 pour vivant et 0 pour mort
00475                 plateau: list[list[str]] = self.scene.get_tableau("1", "0")
00476                 # Pour chaque ligne
00477                 for i in range(len(plateau)):
00478                     # Pour chaque élément
00479                     for j in range(len(plateau[i])):
00480                         # On écrit l'élément
00481                         f.write(plateau[i][j])
00482                         # Si l'élément n'est pas le dernier
00483                         if j < len(plateau[i]) - 1:
00484                             # on écrit une virgule
00485                             f.write(",")
00486                     # Ecriture d'un retour chariot
00487                     f.write("\n")
00488                 # Fermeture du fichier
00489                 f.close()
00490             # On prévient l'utilisateur que le fichier a été enregistré
00491             self.event(QStatusTipEvent("Le fichier a été enregistré !"))
00492         # S'il n'y a pas de fichier ouvert
00493         else:
00494             # Appelle la méthode enregistrer_sous
00495             self.enregistrer_sous()
00496 
```

enregistrer_sous()

```
None jeu_de_la_vie_gui.JeuDeLaVieGUI.enregistrer_sous (
              self)
```

```
Entrée:
    self: JeuDeLaVieGUI
Sortie:
    None (modification en place)
Rôle:
    Enregistre sous le plateau au format csv.
```

Définition à la ligne 497 du fichier jeu_de_la_vie_gui.py.

```
00497     def enregistrer_sous(self) -> None:
00498         """
00499         Entrée:
00500             self: JeuDeLaVieGUI
00501         Sortie:
00502             None (modification en place)
00503         Rôle:
00504             Enregistre sous le plateau au format csv.
00505         """
00506         # Déclaration d'un nouveau QFileDialog
00507         enregistrer_fichier = QFileDialog(self, caption="Enregistrer sous")
00508         # Définition du type de fichier attendu
00509         enregistrer_fichier.setNameFilter("Templates (*.csv)")
00510         # Définition du filtre
00511         enregistrer_fichier.setFilter(
00512             QDir.Filter.Readable | QDir.Filter.Files | QDir.Filter.NoSymLinks)
00513         # Définition du mode du fichier sur fichier existant
00514         enregistrer_fichier.setFileMode(QFileDialog.FileMode.ExistingFile)
00515         # Définition du suffix par défaut
00516         enregistrer_fichier.setDefaultSuffix(".csv")
00517         # Définition du mode
00518         enregistrer_fichier.setAcceptMode(QFileDialog.AcceptMode.AcceptSave)
00519 
00520         # Si l'utilisateur sélectionne un fichier
00521         if enregistrer_fichier.exec():
00522             # On met à jour l'attribut est_fichier_ouvert
00523             self.est_fichier_ouvert = True
00524             # Mise à jour du nom du fichier
00525             self.fichier = enregistrer_fichier.selectedFiles()[0]
00526             # On enregistre le template
00527             self.enregistrer()
00528 
```

event()

```
bool jeu_de_la_vie_gui.JeuDeLaVieGUI.event (
              self,
             QEvent even)
```

```
Réimplémentation de event hérité de QMainWindow
Entrées:
    self: JeuDeLaVieGUI
    even: QEvent (et les classes qui en hérite)
Sortie:
    bool
Rôle:
    Capture les événements de QMainWindow, les traites, puis les rends.
```

Définition à la ligne 430 du fichier jeu_de_la_vie_gui.py.

```
00430     def event(self, even: QEvent) -> bool:
00431         """
00432         Réimplémentation de event hérité de QMainWindow
00433         Entrées:
00434             self: JeuDeLaVieGUI
00435             even: QEvent (et les classes qui en hérite)
00436         Sortie:
00437             bool
00438         Rôle:
00439             Capture les événements de QMainWindow, les traites, puis les rends.
00440         """
00441         # Si le type de l'évènement est StatusTipEvent
00442         if even.type() is QEvent.Type.StatusTip:
00443             # On précise le type de l'évènement pour être sûr de pouvoir
00444             # accéder aux méthodes (optionnel)
00445             even: QStatusTipEvent = even
00446             # Définit le texte de bar_info sur le conseil de l'évènement
00447             self.bar_info.setText(even.tip())
00448             """
00449             Fonctionnement:
00450             Chaque objet possédant l'attribut statusTip émet un signal
00451             QStatusTipEvent lorsque la souris le survole. Ce signal comporte un
00452             tip, un conseil, concernant le dit objet. Il est par défaut
00453             réceptionné par statusBar de QMainWindow, mais il peut être
00454             intercepté et affiché autre part.
00455             """
00456 
00457         # On rend l'évènement
00458         return super().event(even)
00459 
```

importe_template()

```
None jeu_de_la_vie_gui.JeuDeLaVieGUI.importe_template (
              self)

Entrée:
    self: JeuDeLaVieGUI
Sortie:
    None (modification en place)
Rôle:
    Importe un template d'un fichier csv.
```

Définition à la ligne 529 du fichier jeu_de_la_vie_gui.py.

```
00529     def importe_template(self) -> None:
00530         """
00531         Entrée:
00532             self: JeuDeLaVieGUI
00533         Sortie:
00534             None (modification en place)
00535         Rôle:
00536             Importe un template d'un fichier csv.
00537         """
00538         # Déclaration d'un nouveau QFileDialog
00539         obtenir_fichier = QFileDialog(self, caption="Importer un template")
00540         # Définition du type de fichier attendu
00541         obtenir_fichier.setNameFilter("Templates (*.csv)")
00542         # Définition du filtre
00543         obtenir_fichier.setFilter(
00544             QDir.Filter.Readable | QDir.Filter.Files | QDir.Filter.NoSymLinks)
00545         # Définition du mode du fichier sur fichier existant
00546         obtenir_fichier.setFileMode(QFileDialog.FileMode.ExistingFile)
00547         # Définition du suffix par défaut
00548         obtenir_fichier.setDefaultSuffix(".csv")
00549         # Définition du mode
00550         obtenir_fichier.setAcceptMode(QFileDialog.AcceptMode.AcceptOpen)
00551
00552         # Si l'utilisateur sélectionne un fichier
00553         if obtenir_fichier.exec():
00554             # Récupère le chemin du fichier
00555             fichier = obtenir_fichier.selectedFiles()[0]
00556
00557             # On ouvre le fichier en mode lecture
00558             with open(file=fichier, mode='r', encoding='utf8') as f:
00559                 # On extrait les données
00560                 donnees = f.readlines()
00561                 # On ferme le fichier
00562                 f.close()
00563
00564             # On teste la validité des données
00565             tableau_valide, message_erreur = est_template_valide(donnees)
00566
00567             # si le tableau est valide
00568             if tableau_valide:
00569                 # On arrêtre l'animation (economie de ressources)
00570                 self.stop_anim()
00571                 # Déclaration d'une matrice vide
00572                 tableau = []
00573                 # Pour chaque ligne
00574                 for i, ligne in enumerate(reader(donnees)):
00575                     # On ajoute une nouvelle ligne à la matrice
00576                     tableau.append([])
00577                     # Pour chaque élément de la ligne
00578                     for element in ligne:
00579                         # On ajoute l'élément converti en int
00580                         tableau[i].append(int(element))
00581                 # On définit le plateau selon la matrice, sachant que l'élément
00582                 # vivant est 1
00583                 self.scene.set_tableau(tableau, 1)
00584                 # Remet le nombre de cycle à 0
00585                 self.nb_cycle = 0
00586                 # Modifie le texe du label
00587                 self.affichage_cycle.setText("Cycle n°" + str(self.nb_cycle))
00588                 # On met à jour l'attribut est_fichier_ouvert
00589                 self.est_fichier_ouvert = True
00590                 # Mise à jour du nom du fichier
00591                 self.fichier = fichier
00592             # Si le tableau n'est pas valide
00593             else:
00594                 # Affichage du message d'erreur
00595                 QMessageBox.warning(
00596                     self, "Erreur", message_erreur,
00597                     QMessageBox.StandardButton.Ok,
00598                     QMessageBox.StandardButton.Ok)
00599
```

lance_anim()

```
None jeu_de_la_vie_gui.JeuDeLaVieGUI.lance_anim (
              self)
```

```
Entrées:
    self: JeuDeLaVieGUI
Sortie:
    None (modification en place)
Rôle:
    Lance l'animation
```

Définition à la ligne 646 du fichier jeu_de_la_vie_gui.py.

```
00646     def lance_anim(self) -> None:
00647         """
00648         Entrées:
00649             self: JeuDeLaVieGUI
00650         Sortie:
00651             None (modification en place)
00652         Rôle:
00653             Lance l'animation
00654         """
00655         # désactive le bouton jouer
00656         self.jouer.setEnabled(False)
00657         # active le bouton pause
00658         self.pause.setEnabled(True)
00659         # On enlève focus de periode_entree, puisqu'en se désactivant, le jouer
00660         # lui donne le focus
00661         self.periode_entree.clearFocus()
00662
00663         # Démarre le jeu
00664         self.scene.run()
00665
```

set_auto_grandissement()

```
None jeu_de_la_vie_gui.JeuDeLaVieGUI.set_auto_grandissement (
              self,
             Qt.CheckState etat)
```

```
Entrées:
    self: JeuDeLaVieGUI
    etat: Qt.CheckState
Sortie:
    None (modification en place)
Rôle:
    Redéfinit la valeur de auto_grandissement selon etat
```

Définition à la ligne 600 du fichier jeu_de_la_vie_gui.py.

```
00600     def set_auto_grandissement(self, etat: Qt.CheckState) -> None:
00601         """
00602         Entrées:
00603             self: JeuDeLaVieGUI
00604             etat: Qt.CheckState
00605         Sortie:
00606             None (modification en place)
00607         Rôle:
00608             Redéfinit la valeur de auto_grandissement selon etat
00609         """
00610         # | etat             | etat.value
00611         # | Unchecked        | 0
00612         # | Checked          | 2
00613         # bool(0) -> False, bool(2) -> True
00614         # Attribut une nouvelle valeur à auto_grandissement
00615         self.scene.auto_grandissement = bool(etat.value)
00616
```

set_auto_stop()

```
None jeu_de_la_vie_gui.JeuDeLaVieGUI.set_auto_stop (
              self,
             Qt.CheckState etat)


Entrées:
    self: JeuDeLaVieGUI
    etat: Qt.CheckState
Sortie:
    None (modification en place)
Rôle:
    Redéfinit la valeur de auto_stop selon etat
```

Définition à la ligne 617 du fichier jeu_de_la_vie_gui.py.

```
00617     def set_auto_stop(self, etat: Qt.CheckState) -> None:
00618         """
00619         Entrées:
00620             self: JeuDeLaVieGUI
00621             etat: Qt.CheckState
00622         Sortie:
00623             None (modification en place)
00624         Rôle:
00625             Redéfinit la valeur de auto_stop selon etat
00626         """
00627         # | etat              | etat.value
00628         # | Unchecked         | 0
00629         # | Checked           | 2
00630         # bool(0) -> False, bool(1) -> True
00631         # Attribut une nouvelle valeur à auto_stop
00632         self.scene.auto_stop = bool(etat.value)
00633
```

set_periode()

```
None jeu_de_la_vie_gui.JeuDeLaVieGUI.set_periode (
              self,
             float valeur)


Entrées:
    self: JeuDeLaVieGUI
    valeur: double
Sortie:
    None (modification en place)
Rôle:
    Redéfinit la période/durée d'un cycle de l'animation.
```

Définition à la ligne 634 du fichier jeu_de_la_vie_gui.py.

```
00634     def set_periode(self, valeur: float) -> None:
00635         """
00636         Entrées:
00637             self: JeuDeLaVieGUI
00638             valeur: double
00639         Sortie:
00640             None (modification en place)
00641         Rôle:
00642             Redéfinit la période/durée d'un cycle de l'animation.
00643         """
00644         self.scene.periode = int(valeur * 1000)
00645
```

stop_anim()

```
None jeu_de_la_vie_gui.JeuDeLaVieGUI.stop_anim (
              self)
```

```
Entrées:
    self: JeuDeLaVieGUI
Sortie:
    None (modification en place)
Rôle:
    Pause l'animation
```

Définition à la ligne 666 du fichier jeu_de_la_vie_gui.py.

```
00666     def stop_anim(self) -> None:
00667         """
00668         Entrées:
00669             self: JeuDeLaVieGUI
00670         Sortie:
00671             None (modification en place)
00672         Rôle:
00673             Pause l'animation
00674         """
00675         # Arrête le chrono
00676         self.scene.chrono.stop()
00677         # active le bouton jouer
00678         self.jouer.setEnabled(True)
00679         # désactive le bouton pause
00680         self.pause.setEnabled(False)
00681         # On enlève focus de periode_entree, puisqu'en se désactivant, le pause
00682         # lui donne le focus
00683         self.periode_entree.clearFocus()
00684
```

zoom_in()

```
None jeu_de_la_vie_gui.JeuDeLaVieGUI.zoom_in (
              self)
```

```
Entrées:
    self: JeuDeLaVieGUI
Sortie:
    None (modification en place)
Rôle:
    Zoom dans la vue
Addendum:
    Le nom est en anglais car le terme n'existe pas en français.
```

Définition à la ligne 685 du fichier jeu_de_la_vie_gui.py.

```
00685     def zoom_in(self) -> None:
00686         """
00687         Entrées:
00688             self: JeuDeLaVieGUI
00689         Sortie:
00690             None (modification en place)
00691         Rôle:
00692             Zoom dans la vue
00693         Addendum:
00694             Le nom est en anglais car le terme n'existe pas en français.
00695         """
00696         # Augmente les grandeurs de 10%
00697         self.vue.scale(1.1, 1.1)
00698
```

zoom_out()

```
None jeu_de_la_vie_gui.JeuDeLaVieGUI.zoom_out (
              self)


Entrées:
    self: JeuDeLaVieGUI
Sortie:
    None (modification en place)
Rôle:
    Dézoom dans la vue
Addendum:
    Le nom est en anglais car le terme n'existe pas en français.
```

Définition à la ligne 699 du fichier jeu_de_la_vie_gui.py.

```
00699     def zoom_out(self) -> None:
00700         """
00701         Entrées:
00702             self: JeuDeLaVieGUI
00703         Sortie:
00704             None (modification en place)
00705         Rôle:
00706             Dézoom dans la vue
00707         Addendum:
00708             Le nom est en anglais car le terme n'existe pas en français.
00709         """
00710         # Diminue les grandeurs de 10%
00711         self.vue.scale(0.9, 0.9)
00712
00713
00714 # Si le présent fichier est executé avec python 3.10 ou plus
```

6.5.4 Documentation des données membres

affichage

```
jeu_de_la_vie_gui.JeuDeLaVieGUI.affichage = QHBoxLayout()
```

Définition à la ligne 100 du fichier jeu_de_la_vie_gui.py.

affichage_cycle

```
jeu_de_la_vie_gui.JeuDeLaVieGUI.affichage_cycle = QLabel(self)
```

Définition à la ligne 174 du fichier jeu_de_la_vie_gui.py.

auto_grandissement_entree

```
jeu_de_la_vie_gui.JeuDeLaVieGUI.auto_grandissement_entree = QCheckBox(self)
```

Définition à la ligne 296 du fichier jeu_de_la_vie_gui.py.

auto_stop_entree

```
jeu_de_la_vie_gui.JeuDeLaVieGUI.auto_stop_entree = QCheckBox(self)
```

Définition à la ligne 311 du fichier jeu_de_la_vie_gui.py.

bar_info

```
jeu_de_la_vie_gui.JeuDeLaVieGUI.bar_info = QLabel(self)
```

Définition à la ligne 386 du fichier jeu_de_la_vie_gui.py.

bar_info_titre

```
jeu_de_la_vie_gui.JeuDeLaVieGUI.bar_info_titre = QLabel(self)
```

Définition à la ligne 378 du fichier jeu_de_la_vie_gui.py.

chemin_absolu

```
str jeu_de_la_vie_gui.JeuDeLaVieGUI.chemin_absolu = __file__[:-len(basename(__file__))]
```

Définition à la ligne 76 du fichier jeu_de_la_vie_gui.py.

controles_layout

```
jeu_de_la_vie_gui.JeuDeLaVieGUI.controles_layout = QHBoxLayout()
```

Définition à la ligne 198 du fichier jeu_de_la_vie_gui.py.

controles_titre

```
jeu_de_la_vie_gui.JeuDeLaVieGUI.controles_titre = QLabel(self)
```

Définition à la ligne 187 du fichier jeu_de_la_vie_gui.py.

enregistrer

```
jeu_de_la_vie_gui.JeuDeLaVieGUI.enregistrer
```

Définition à la ligne 128 du fichier jeu_de_la_vie_gui.py.

enregistrer_sous

```
jeu_de_la_vie_gui.JeuDeLaVieGUI.enregistrer_sous
```

Définition à la ligne 140 du fichier jeu_de_la_vie_gui.py.

enregistrer_template

```
jeu_de_la_vie_gui.JeuDeLaVieGUI.enregistrer_template = QAction(self)
```

Définition à la ligne 122 du fichier jeu_de_la_vie_gui.py.

enregistrer_template_sous

```
jeu_de_la_vie_gui.JeuDeLaVieGUI.enregistrer_template_sous = QAction(self)
```

Définition à la ligne 133 du fichier jeu_de_la_vie_gui.py.

est_fichier_ouvert

```
bool jeu_de_la_vie_gui.JeuDeLaVieGUI.est_fichier_ouvert = False
```

Définition à la ligne 78 du fichier jeu_de_la_vie_gui.py.

fichier

```
str jeu_de_la_vie_gui.JeuDeLaVieGUI.fichier = ""
```

Définition à la ligne 79 du fichier jeu_de_la_vie_gui.py.

importe_template

```
jeu_de_la_vie_gui.JeuDeLaVieGUI.importe_template
```

Définition à la ligne 117 du fichier jeu_de_la_vie_gui.py.

importer_template

```
jeu_de_la_vie_gui.JeuDeLaVieGUI.importer_template = QAction(self)
```

Définition à la ligne 111 du fichier jeu_de_la_vie_gui.py.

jouer

```
jeu_de_la_vie_gui.JeuDeLaVieGUI.jouer = QPushButton(self)
```

Définition à la ligne 200 du fichier jeu_de_la_vie_gui.py.

lance_anim

```
jeu_de_la_vie_gui.JeuDeLaVieGUI.lance_anim
```

Définition à la ligne 209 du fichier jeu_de_la_vie_gui.py.

ligne_post_autos

```
jeu_de_la_vie_gui.JeuDeLaVieGUI.ligne_post_autos = QFrame(self)
```

Définition à la ligne 328 du fichier jeu_de_la_vie_gui.py.

ligne_post_controles

```
jeu_de_la_vie_gui.JeuDeLaVieGUI.ligne_post_controles = QFrame(self)
```

Définition à la ligne 285 du fichier jeu_de_la_vie_gui.py.

menu

```
QMenuBar jeu_de_la_vie_gui.JeuDeLaVieGUI.menu = self.menuBar()
```

Définition à la ligne 107 du fichier jeu_de_la_vie_gui.py.

menu_fichier

```
QMenu jeu_de_la_vie_gui.JeuDeLaVieGUI.menu_fichier = self.menu.addMenu("&Fichier")
```

Définition à la ligne 109 du fichier jeu_de_la_vie_gui.py.

nb_cycle

```
int jeu_de_la_vie_gui.JeuDeLaVieGUI.nb_cycle = 0
```

Définition à la ligne 81 du fichier jeu_de_la_vie_gui.py.

outils_layout

```
jeu_de_la_vie_gui.JeuDeLaVieGUI.outils_layout = QVBoxLayout()
```

Définition à la ligne 166 du fichier jeu_de_la_vie_gui.py.

pause

```
jeu_de_la_vie_gui.JeuDeLaVieGUI.pause = QPushButton(self)
```

Définition à la ligne 217 du fichier jeu_de_la_vie_gui.py.

periode_entree

```
jeu_de_la_vie_gui.JeuDeLaVieGUI.periode_entree = QDoubleSpinBox(self)
```

Définition à la ligne 260 du fichier jeu_de_la_vie_gui.py.

periode_label

```
jeu_de_la_vie_gui.JeuDeLaVieGUI.periode_label = QLabel(self)
```

Définition à la ligne 253 du fichier jeu_de_la_vie_gui.py.

periode_layout

```
jeu_de_la_vie_gui.JeuDeLaVieGUI.periode_layout = QGridLayout()
```

Définition à la ligne 251 du fichier jeu_de_la_vie_gui.py.

scene

```
jeu_de_la_vie_gui.JeuDeLaVieGUI.scene = Scene(self, QSize(10, 10))
```

Définition à la ligne 147 du fichier jeu_de_la_vie_gui.py.

set_auto_grandissement

```
jeu_de_la_vie_gui.JeuDeLaVieGUI.set_auto_grandissement
```

Définition à la ligne 306 du fichier jeu_de_la_vie_gui.py.

set_auto_stop

```
jeu_de_la_vie_gui.JeuDeLaVieGUI.set_auto_stop
```

Définition à la ligne 322 du fichier jeu_de_la_vie_gui.py.

set_periode

```
jeu_de_la_vie_gui.JeuDeLaVieGUI.set_periode
```

Définition à la ligne 275 du fichier jeu_de_la_vie_gui.py.

stop_anim

```
jeu_de_la_vie_gui.JeuDeLaVieGUI.stop_anim
```

Définition à la ligne 226 du fichier jeu_de_la_vie_gui.py.

tour_par_tour

```
jeu_de_la_vie_gui.JeuDeLaVieGUI.tour_par_tour = QPushButton(self)
```

Définition à la ligne 239 du fichier jeu_de_la_vie_gui.py.

vue

```
jeu_de_la_vie_gui.JeuDeLaVieGUI.vue = QGraphicsView(self)
```

Définition à la ligne 149 du fichier jeu_de_la_vie_gui.py.

widget_central

```
jeu_de_la_vie_gui.JeuDeLaVieGUI.widget_central = QWidget(self)
```

Définition à la ligne 95 du fichier jeu_de_la_vie_gui.py.

zoom

```
jeu_de_la_vie_gui.JeuDeLaVieGUI.zoom = QGridLayout()
```

Définition à la ligne 339 du fichier jeu_de_la_vie_gui.py.

zoom_in

```
jeu_de_la_vie_gui.JeuDeLaVieGUI.zoom_in
```

Définition à la ligne 358 du fichier jeu_de_la_vie_gui.py.

zoom_in_entree

```
jeu_de_la_vie_gui.JeuDeLaVieGUI.zoom_in_entree = QPushButton(self)
```

Définition à la ligne 352 du fichier jeu_de_la_vie_gui.py.

zoom_label

```
jeu_de_la_vie_gui.JeuDeLaVieGUI.zoom_label = QLabel(self)
```

Définition à la ligne 341 du fichier jeu_de_la_vie_gui.py.

zoom_out

```
jeu_de_la_vie_gui.JeuDeLaVieGUI.zoom_out
```

Définition à la ligne 368 du fichier jeu_de_la_vie_gui.py.

zoom_out_entree

```
jeu_de_la_vie_gui.JeuDeLaVieGUI.zoom_out_entree = QPushButton(self)
```

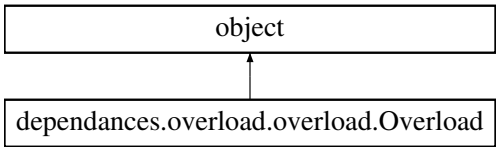
Définition à la ligne 362 du fichier jeu_de_la_vie_gui.py.

La documentation de cette classe a été générée à partir du fichier suivant :

— D:/Jeu de la vie/source/jeu_de_la_vie_gui.py

6.6 Référence de la classe dependances.overload.overload.Overload

Graphe d'héritage de dependances.overload.overload.Overload:



Fonctions membres publiques

- None __init__ (self, callable func)
- object __get__ (self, object|None owner, Any ownerType=None)
- __call__ (self, *args, **kwargs)
- overload (self, callable func)

Attributs publics

- owner = None
- list signatures = [ ]
- list methods = [ ]

6.6.1 Description détaillée

```
Hérite de:
    object
Rôle:
    Permet de surcharger une (uniquement) méthode (uniquement) et met en
    oeuvre le principe de substitution de Liskov.
Exemple:
    >>> @Overload  # (il est préférable de l'utiliser en décorateur)
    >>> @signature(int)  # (obligatoire !!!)
    >>> def double(valeur: int) -> int:
    >>>     return int(valeur * 2)
    >>> @double.overload  # Pas de majuscule à overload
    >>> @signature(float)  # (obligatoire !!!)
    >>> def double(valeur: float) -> float:
    >>>     return float(valeur * 2)
    >>> double(.2)  # .2 est de type float
    0.4  # float
    >>> double(5)  # 5 est de type int
    10  # int
Addendum:
    Overload est dérivée, comme signature, de pythonlangutil.overload donc
    les variables sont en englais.
```

Définition à la ligne 127 du fichier overload.py.

6.6.2 Documentation des constructeurs et destructeur

__init__()

```
None dependances.overload.overload.Overload.__init__ (
              self,
             callable func)


Entrées:
    self: Overload
    func: callable (fonction à surcharger)
Sortie:
    None (ctor)
Rôle:
    Construit un nouvel objet Overload
```

Définition à la ligne 151 du fichier overload.py.

```
00151     def __init__(self, func: callable) -> None:
00152         """
00153         Entrées:
00154             self: Overload
00155             func: callable (fonction à surcharger)
00156         Sortie:
00157             None (ctor)
00158         Rôle:
00159             Construit un nouvel objet Overload
00160         """
00161         # Déclaration de l'attribut owner pour la méhtode magique __get__
00162         self.owner = None
00163         # Déclaration d'une liste de tuple où les tuples sont les signatures
00164         # des méthodes qui se surchargent
00165         self.signatures: list[tuple] = []
00166         # Liste des méthodes qui se surchargent
00167         self.methods: list[callable] = []
00168         # On ajoute la méthode surchargée à methods
00169         self.methods.append(func)
00170         # On ajoute la signature de la méthode à signatures
00171         self.signatures.append(func.signature)
00172
```

6.6.3 Documentation des fonctions membres

__call__()

```
dependances.overload.overload.Overload.__call__ (
              self,
             * args,
             ** kwargs)


Entrée:
    self: Overload
    args: Any  (arguments passés à la fonction surchargée)
    kwargs: Any (couples (clé;arg) passés à la fonction surchargée)
Sortie:
    Résultat de la fonction surchargé qui correspond à la signature des
    arguments passés en paramètre.
Rôle:
    Trouve la fonction correspondant à la signature des arguments
    passés en paramètre, l'execute et retourne sa valeur de retour.
```

Définition à la ligne 192 du fichier overload.py.

```
00192     def __call__(self, *args, **kwargs):
00193         """
00194         Entrée:
00195             self: Overload
00196             args: Any  (arguments passés à la fonction surchargée)
00197             kwargs: Any (couples (clé;arg) passés à la fonction surchargée)
00198         Sortie:
00199             Résultat de la fonction surchargé qui correspond à la signature des
00200             arguments passés en paramètre.
00201         Rôle:
00202             Trouve la fonction correspondant à la signature des arguments
00203             passés en paramètre, l'execute et retourne sa valeur de retour.
00204         """
00205         # Signature des arguments
00206         signature = []
00207         # Pour chaque argument sans clé
00208         for arg in args:
00209             # On ajoute le type des arguments sans clé
00210             signature.append(arg.__class__.__name__)
00211         # Pour chaque clé (_), argument (v)
00212         for _, v in kwargs:
00213             # On ajoute le type des arguments qui ont une clé
00214             signature.append(v.__class__.__name__)
00215         # Signature doit être un tuple car les signatures sont stockée sous
00216         # forme de tuple
00217         signature = tuple(signature)
00218         # si la signature exacte est enregistrée
00219         if signature in self.signatures:
00220             # On trouve l'indice correspondant à cet signature
00221             index = self.signatures.index(signature)
00222         # si la signature exacte est n'est pas enregistrée
00223         else:
00224             # extrait l'arbre d'héritage pour chaque argument de la signature
00225             inherited_signature = []
00226             # Pour chaque argument sans clé
00227             for arg in args:
00228                 # On ajoute l'arbre d'héritage des arguments sans clé
00229                 inherited_signature.append(get_signature_complete(arg))
00230             # Pour chaque clé (_), argument (v)
00231             for _, v in kwargs:
00232                 # On ajoute l'arbre d'héritage des arguments qui ont une clé
00233                 signature.append(get_signature_complete(v))
00234 
00235             # signature éligibles
00236             sig_eligibles = []
00237             # Pour chaque signature des signatures enregistrées
00238             for sig in self.signatures:
00239                 # Si son nombre d'argument correspond à la signature des args
00240                 if len(inherited_signature) == len(sig):
00241                     # On l'ajoute aux signature éligibles
00242                     sig_eligibles.append(sig)
00243 
00244             # Index négatif -> erreur
00245             index = -1
00246             # Variable d'arrêt, False s'il n'y a pas de signature éligible
00247             existe = len(sig_eligibles) > 0  # bool
00248             # Itérateur i
00249             i = 0
00250             # Pour chaque signature
00251             while i < len(sig_eligibles) and existe:
00252                 # Itérateur j
00253                 j = 0
00254                 # Variable d'arrêt de la seconde boucle
00255                 correspond = True
00256                 # Pour chaque type de la signature
00257                 while j < len(sig_eligibles[i]) and correspond:
00258                     # Si le type n'est pas dans les type hérité du paramètre
00259                     # c'est que ce n'est pas le bon type
00260                     if sig_eligibles[i][j] not in inherited_signature[j]:
00261                         # On stop la boucle
00262                         correspond = False
00263                     # Incrémentation de l'itérateur
00264                     j += 1
00265                 # Si la signature correspond
00266                 if correspond:
00267                     # On recherche l'indice car la signature est unique
00268                     # (cf self.overload)
00269                     index = self.signatures.index(sig_eligibles[i])
00270                 # Incrémentation de l'itérateur
00271                 i += 1
00272             # Si la signature ne correspond à aucune des méthodes enregistrées
00273             if not existe or index < 0:
00274                 # On lève une exception
00275                 raise Exception("There is no overload for this method with " +
00276                                 "this signature.")
00277 
```

```
00278         # On execute la méthode avec ses arguments et on retourne sa valeur de
00279         # retour
00280         return self.methods[index](self.owner, *args, **kwargs)
00281
```

__get__()

```
object dependances.overload.overload.Overload.__get__ (
              self,
             object | None owner,
             Any  ownerType = None)
```

```
Entrée:
    self: Overload
    owner: object | None
    owerType: None | Any
Sortie:
    Overload
Rôle:
    Retourne l'objet Overload et définit l'attribut owner.
Description:
https://python-reference.readthedocs.io/en/latest/docs/dunderdsc/get.html
```

Définition à la ligne 173 du fichier overload.py.

```
00173     def __get__(self, owner: object | None, ownerType: Any = None) -> object:
00174         """
00175         Entrée:
00176             self: Overload
00177             owner: object | None
00178             owerType: None | Any
00179         Sortie:
00180             Overload
00181         Rôle:
00182             Retourne l'objet Overload et définit l'attribut owner.
00183         Description:
00184     https://python-reference.readthedocs.io/en/latest/docs/dunderdsc/get.html
00185         """
00186         # owner n'est pas définit, self.owner = self, sinon, self.owner = owner
00187         self.owner = owner or self
00188
00189         # retourne l'objet Overload
00190         return self
00191
```

overload()

```
dependances.overload.overload.Overload.overload (
              self,
             callable func)
```

```
Entrée:
    self: Overload
    func: callable
Sortie:
    Overload
Rôle:
    Ajoute une nouvelle surcharge
Explications:
    Faire:
    >>> @Overload
    >>> @signature(int)
    >>> def double(valeur: int) -> int:
    >>>     return int(valeur * 2)
    C'est faire:
    >>> def double(valeur: int) -> int:
```

```
>>>     return int(valeur * 2)
>>> double = signature('int', double)
>>> double = Overload(double)

Donc quand on fait
>>> @double.overload  # appelle la méthode overload
>>> @signature(float)
>>> def double2(valeur: float) -> float:
>>>     return float(valeur * 2)
On fait:
>>> def double2(valeur: float) -> float:
>>>     return float(valeur * 2)
>>> double2 = signature(double2)
>>> double2 = double.overload(double2)# appelle la méthode overload
En pratique, il ne faut pas changer le nom de la méthode quand on
la surcharge.

https://www.geeksforgeeks.org/decorators-in-python/
```

Définition à la ligne 282 du fichier overload.py.

```
00282     def overload(self, func: callable):
00283         """
00284         Entrée:
00285             self: Overload
00286             func: callable
00287         Sortie:
00288             Overload
00289         Rôle:
00290             Ajoute une nouvelle surcharge
00291         Explications:
00292             Faire:
00293             >> @Overload
00294             >> @signature(int)
00295             >> def double(valeur: int) -> int:
00296             >>     return int(valeur * 2)
00297             C'est faire:
00298             >> def double(valeur: int) -> int:
00299             >>     return int(valeur * 2)
00300             >> double = signature('int', double)
00301             >> double = Overload(double)
00302
00303             Donc quand on fait
00304             >> @double.overload  # appelle la méthode overload
00305             >> @signature(float)
00306             >> def double2(valeur: float) -> float:
00307             >>     return float(valeur * 2)
00308             On fait:
00309             >> def double2(valeur: float) -> float:
00310             >>     return float(valeur * 2)
00311             >> double2 = signature(double2)
00312             >> double2 = double.overload(double2)# appelle la méthode overload
00313             En pratique, il ne faut pas changer le nom de la méthode quand on
00314             la surcharge.
00315
00316             https://www.geeksforgeeks.org/decorators-in-python/
00317         """
00318         # Si la signature est unique
00319         if func.signature not in self.signatures:
00320             # On ajoute la méthode à methods
00321             self.methods.append(func)
00322             # On ajoute la signature de func à signatures
00323             self.signatures.append(func.signature)
00324             # Retourne l'objet Overload
00325             return self
00326         # Si la signature n'est pas unique
00327         else:
00328             # On lève une exception
00329             raise Exception("There is no overload for this method with this" +
00330                             " signature: the signature is already taken.")
```

6.6.4 Documentation des données membres

methods

```
list dependances.overload.overload.Overload.methods = []
```

Définition à la ligne 167 du fichier overload.py.

owner

```
dependances.overload.overload.Overload.owner = None
```

Définition à la ligne 162 du fichier overload.py.

signatures

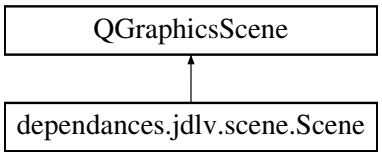
```
list dependances.overload.overload.Overload.signatures = []
```

Définition à la ligne 165 du fichier overload.py.

La documentation de cette classe a été générée à partir du fichier suivant :
— D:/Jeu de la vie/source/dependances/overload/overload.py

6.7 Référence de la classe dependances.jdlv.scene.Scene

Graphe d'héritage de dependances.jdlv.scene.Scene:



Fonctions membres publiques

— None __init__ (self, QWidget|None parent=None, QSize taille=QSize(0, 0))
— None vide_scene (self)
— None set_tableau (self, int height, int width, Etat etat)
— None set_tableau (self, list[list[Any]] tableau, Any vivant)
— None mouseMoveEvent (self, QGraphicsSceneMouseEvent event)
— bool event (self, QEvent even)
— list[list[Any]] get_tableau (self, Any vivant, Any mort)
— bool valeur_case (self, int ord, int absc)
— int total_voisins (self, int ord, int absc)
— Etat resultat (self, int ord, int absc)
— list[Direction] doit_agrandir_tableau (self)
— None extension (self, Direction direction)
— bool arret_automatique (self)
— None tour (self)
— None run (self)

Fonctions membres publiques statiques

- bool meurt (int nb_voisins)
- bool nait (int nb_voisins)

Attributs publics

- QSize dimension = taille
- bool auto_grandissement = None
- bool auto_stop = None
- int periode = None
- bool est_souris_dans_scene = False
- chrono = QTimer(self)
- run
- list tableau_precedent = [ ]
- list tableau = [ ]
- list[Direction] auto_grandissement = self.doit_agrandir_tableau()

6.7.1 Description détaillée

```
Hérite de:
    QGraphicsScene
Rôle:
    Représente la scène du jeu de la vie (Scene)
```

Définition à la ligne 22 du fichier scene.py.

6.7.2 Documentation des constructeurs et destructeur

__init__()

```
None dependances.jdlv.scene.Scene.__init__ (
             self,
            QWidget | None  parent = None,
            QSize  taille = QSize(0, 0))


Entrées:
    self: Scene
    parent QWidget (ligne d'héritage pour désallouer la mémoire)
        valeur par défaut: None
    taille: QSize
        valeur par défaut: QSize(0, 0)
Sortie:
    None (ctor)
Rôle:
    Construit un nouvel objet JDLV
```

Définition à la ligne 29 du fichier scene.py.

```
00030                 taille: QSize = QSize(0, 0)) -> None:
00031         """
00032         Entrées:
00033             self: Scene
00034             parent QWidget (ligne d'héritage pour désallouer la mémoire)
00035                 valeur par défaut: None
00036             taille: QSize
00037                 valeur par défaut: QSize(0, 0)
00038         Sortie:
00039             None (ctor)
00040         Rôle:
00041             Construit un nouvel objet JDLV
00042         """
00043         # On initialise la classe mère
00044         QGraphicsScene.__init__(self, parent)
00045 
00046         # Déclaration de dimension qui représente la dimension du plateau
00047         # (w x h)
00048         self.dimension: QSize = taille
00049         # Déclaration d'un attribut auto grandissement
00050         self.auto_grandissement: bool = None
00051         # Déclaration d'un attribut auto stop
00052         self.auto_stop: bool = None
00053         # Déclaration d'interval
00054         self.periode: int = None
00055         # Déclaration d'un attribut est_souris_dans_scene
00056         self.est_souris_dans_scene = False
00057 
00058         # Déclaration d'un chronomètre
00059         self.chrono = QTimer(self)
00060         # Relie le signal à run
00061         self.chrono.timeout.connect(self.run)
00062         # Chrono moins précis mais moins gourmant en ressources
00063         self.chrono.setTimerType(Qt.TimerType.CoarseTimer)
00064 
00065         # Déclaration d'un tableau à l'état n-1
00066         self.tableau_precedent: list[list[Cellule]] = []
00067         # Déclaration d'un tableau vide
00068         self.tableau: list[list[Cellule]] = []
00069         # On construit le plateau de jeu d'après les attributs de la fenêtre
00070         self.set_tableau(
00071             self.dimension.height(), self.dimension.width(), Etat.Mort)
00072 
```

6.7.3 Documentation des fonctions membres

arret_automatique()

```
bool dependances.jdlv.scene.Scene.arret_automatique (
              self)
```

```
Entrée:
    self: Scene
Sortie:
    bool
Rôle:
    Vérifie si deux tour de suite sont identique
```

Définition à la ligne 592 du fichier scene.py.

```
00592     def arret_automatique(self) -> bool:
00593         """
00594         Entrée:
00595             self: Scene
00596         Sortie:
00597             bool
00598         Rôle:
00599             Vérifie si deux tour de suite sont identique
00600         """
00601         # retourne True si le tableau n'a pas changé entre deux cycles
00602         return self.tableau_precedent == self.tableau
00603 
```

doit_agrandir_tableau()

```
list[Direction] dependances.jdlv.scene.Scene.doit_agrandir_tableau (
              self)
```

```
Entrée:
    self: Scene
Sortie:
    tuple[bool, Direction]
Rôle:
    Retourne la liste des directions vers lesquelles il faut agrandir
    le tableau
```

Définition à la ligne 412 du fichier scene.py.

```
00412     def doit_agrandir_tableau(self) -> list[Direction]:
00413         """
00414         Entrée:
00415             self: Scene
00416         Sortie:
00417             tuple[bool, Direction]
00418         Rôle:
00419             Retourne la liste des directions vers lesquelles il faut agrandir
00420             le tableau
00421         """
00422         # Déclaration d'une liste qui représente les directions vers lesquelles
00423         # il faut agrandir le tableau
00424         liste_directions = []
00425 
00426         # on teste la frontière nord:
00427         i = 0
00428         # Déclaration de variable d'arrêt et initialisation à False
00429         stop = False
00430         # Pour chaque cellule de la première ligne
00431         while not (i >= self.dimension.width() or stop):
00432             # Si la cellule est vivante
00433             if self.tableau[0][i].get_etat() is Etat.Vivant:
00434                 # On arrête la boucle
00435                 stop = True
00436                 # on ajoute Nord au direction
00437                 liste_directions.append(Direction.Nord)
00438             # On icrémente l'itérateur
00439             i += 1
00440 
00441         # on teste la frontière sud:
00442         i = 0
00443         # Initialisation de stop à False
00444         stop = False
00445         # On cherche l'indice de la dernière ligne
00446         derniere_ligne = self.dimension.height() - 1
00447         # Pour chaque cellule de la dernière ligne
00448         while not (i >= self.dimension.width() or stop):
00449             # Si la cellule est vivante
00450             if self.tableau[derniere_ligne][i].get_etat() is Etat.Vivant:
00451                 # On arrête la boucle
00452                 stop = True
00453                 # on ajoute Sud au direction
00454                 liste_directions.append(Direction.Sud)
00455             # On icrémente l'itérateur
00456             i += 1
00457 
00458         # on teste la frontière ouest:
00459         i = 0
00460         # Initialisation de stop à False
00461         stop = False
00462         # Pour chaque ligne
00463         while not (i >= self.dimension.height() or stop):
00464             # Si la dernière cellule est vivante
00465             if self.tableau[i][0].get_etat() is Etat.Vivant:
00466                 # On arrête la boucle
00467                 stop = True
00468                 # on ajoute Ouest au direction
00469                 liste_directions.append(Direction.Ouest)
00470             # On icrémente l'itérateur
00471             i += 1
00472 
00473         # on teste la frontière ouest:
00474         i = 0
00475         # Initialisation de stop à False
00476         stop = False
00477         # On cherche l'indice de la dernière colonne
```

```
00478         derniere_colonne = self.dimension.width() - 1
00479         # Pour chaque ligne
00480         while not (i >= self.dimension.height() or stop):
00481             # Si la dernière cellule est vivante
00482             if self.tableau[i][derniere_colonne].get_etat() is Etat.Vivant:
00483                 # On arrête la boucle
00484                 stop = True
00485                 # on ajoute Est au direction
00486                 liste_directions.append(Direction.Est)
00487             # On icrémente l'itérateur
00488             i += 1
00489
00490         # Retourne la liste
00491         return liste_directions
00492
```

event()

```
bool dependances.jdlv.scene.Scene.event (
              self,
             QEvent even)
```

```
Réimplémentation de event hérité de QGraphicsScene
Entrées:
    self: Scene
    even: QEvent (et les classes qui en hérite)
Sortie:
    bool
Rôle:
    Capture les événements de QGraphicsScene, les traites, puis les
    rends.
```

Définition à la ligne 236 du fichier scene.py.

```
00236     def event(self, even: QEvent) -> bool:
00237         """
00238         Réimplémentation de event hérité de QGraphicsScene
00239         Entrées:
00240             self: Scene
00241             even: QEvent (et les classes qui en hérite)
00242         Sortie:
00243             bool
00244         Rôle:
00245             Capture les événements de QGraphicsScene, les traites, puis les
00246             rends.
00247         """
00248         # Si le type de l'évènement est StatusTipEvent
00249         if even.type() is QEvent.Type.StatusTip:
00250             # On précise le type de l'évènement pour être sûr de pouvoir
00251             # accéder aux méthodes (optionnel)
00252             even: QStatusTipEvent = even
00253             # Relais l'évènement à la classe mère
00254             self.parent().event(even)
00255             """
00256             Fonctionnement:
00257             Chaque objet possédant l'attribut statusTip émet un signal
00258             QStatusTipEvent lorsque la souris le survole. Ce signal comporte un
00259             tip, un conseil, concernant le dit objet. Il est par défaut
00260             réceptionné par statusBar de QMainWindow, mais il peut être
00261             intercepté et affiché autre part.
00262             """
00263
00264         # On rend l'évènement à la classe mère
00265         return super().event(even)
00266
```

extension()

```
None dependances.jdlv.scene.Scene.extension (
              self,
             Direction direction)
```

```
Entrées:
    self: Scene
    direction: Direction
Sortie:
    None (modification en place)
Rôle
    Étend le tableau vers une direction donnée
```

Définition à la ligne 493 du fichier scene.py.

```
00493     def extension(self, direction: Direction) -> None:
00494         """
00495         Entrées:
00496             self: Scene
00497             direction: Direction
00498         Sortie:
00499             None (modification en place)
00500         Rôle
00501             Étend le tableau vers une direction donnée
00502         """
00503         # Si la direction est Nord
00504         if direction is Direction.Nord:
00505             # Déclaration de abscisse
00506             abscisse = self.tableau[0][0].pos().x()
00507             # Déclaration de ordonnée
00508             ordonnée = self.tableau[0][0].pos().y() - 50
00509             # On ajoute une ligne à l'indice 0 (déplace les éléments)
00510             self.tableau.insert(0, [])
00511             # Pour chaque colonne
00512             for i in range(self.dimension.width()):
00513                 # On créer une cellule
00514                 temp = Cellule()
00515                 # On ajuste sa position (w x h)
00516                 temp.setPos(abscisse + i * 50, ordonnée)
00517                 # On définit l'état de temp
00518                 temp.set_etat(Etat.Mort)
00519                 # On ajoute la cellule au tableau
00520                 self.tableau[0].append(temp)
00521                 # On ajoute la cellule à la scène
00522                 self.addItem(temp)
00523
00524             # On ajuste la dimension du tableau
00525             self.dimension.setHeight(self.dimension.height() + 1)
00526         # Si la direction est Sud
00527         elif direction is Direction.Sud:
00528             # Déclaration de abscisse
00529             abscisse = self.tableau[0][0].pos().x()
00530             # Déclaration de ordonnée
00531             ordonnée = self.tableau[-1][0].pos().y() + 50
00532             # On ajoute une ligne à la fin du tableau
00533             self.tableau.append([])
00534             # Pour chaque colonne
00535             for i in range(self.dimension.width()):
00536                 # On créer une cellule
00537                 temp = Cellule()
00538                 # On ajuste sa position (w x h)
00539                 temp.setPos(abscisse + i * 50, ordonnée)
00540                 # On définit l'état de temp
00541                 temp.set_etat(Etat.Mort)
00542                 # On ajoute la cellule au tableau
00543                 self.tableau[-1].append(temp)
00544                 # On ajoute la cellule à la scène
00545                 self.addItem(temp)
00546
00547             # On ajuste la dimension du tableau
00548             self.dimension.setHeight(self.dimension.height() + 1)
00549         # Si la direction est Est
00550         elif direction is Direction.Est:
00551             # Déclaration de ordonnée
00552             ordonnée = self.tableau[0][0].pos().y()
00553             # Déclaration de abscisse
00554             abscisse = self.tableau[0][-1].pos().x() + 50
00555             # Pour chaque ligne
00556             for i in range(self.dimension.height()):
00557                 # On créer une cellule
00558                 temp = Cellule()
00559                 # On ajuste sa position (w x h)
00560                 temp.setPos(abscisse, ordonnée + i * 50)
00561                 # On définit l'état de temp
00562                 temp.set_etat(Etat.Mort)
00563                 # On ajoute la cellule au tableau
00564                 self.tableau[i].append(temp)
00565                 # On ajoute la cellule à la scène
00566                 self.addItem(temp)
00567
```

```
00568             # On ajuste la dimension du tableau
00569             self.dimension.setWidth(self.dimension.width() + 1)
00570         # Si la direction est Ouest
00571         elif direction is Direction.Ouest:
00572             # Déclaration de ordonnée
00573             ordonnée = self.tableau[0][0].pos().y()
00574             # Déclaration de abscisse
00575             abscisse = self.tableau[0][0].pos().x() - 50
00576             # Pour chaque ligne
00577             for i in range(self.dimension.height()):
00578                 # On créer une cellule
00579                 temp = Cellule()
00580                 # On ajuste sa position (w x h)
00581                 temp.setPos(abscisse, ordonnée + i * 50)
00582                 # On définit l'état de temp
00583                 temp.set_etat(Etat.Mort)
00584                 # On ajoute la cellule au tableau
00585                 self.tableau[i].insert(0, temp)
00586                 # On ajoute la cellule à la scène
00587                 self.addItem(temp)
00588 
00589             # On ajuste la dimension du tableau
00590             self.dimension.setWidth(self.dimension.width() + 1)
00591 
```

get_tableau()

```
list[list[Any]] dependances.jdlv.scene.Scene.get_tableau (
              self,
             Any vivant,
             Any mort)


Entrées:
    self: Scene
    vivant: Any
    mort: Any
Sortie:
    list[list[Any]]  (consitué uniquement de vivant et mort)
Rôle:
    Retourne le plateau de jeu avec des valeurs personnalisées de
    vivant et mort.
```

Définition à la ligne 267 du fichier scene.py.

```
00267     def get_tableau(self, vivant: Any, mort: Any) -> list[list[Any]]:
00268         """
00269         Entrées:
00270             self: Scene
00271             vivant: Any
00272             mort: Any
00273         Sortie:
00274             list[list[Any]]  (consitué uniquement de vivant et mort)
00275         Rôle:
00276             Retourne le plateau de jeu avec des valeurs personnalisées de
00277             vivant et mort.
00278         """
00279         # Déclaration d'un tableau vide plateau
00280         plateau: list[list] = []
00281         # Pour chaque ligne de tableau
00282         for i in range(len(self.tableau)):
00283             # Ajout d'une nouvelle ligne dans plateau
00284             plateau.append([])
00285             # Pour chaque élément de la ligne
00286             for j in range(len(self.tableau[i])):
00287                 # Si l'élément (i;j) est d'état vivant
00288                 if self.tableau[i][j].get_etat() is Etat.Vivant:
00289                     # On ajoute la valeur vivant au plateau
00290                     plateau[i].append(vivant)
00291                 # Si l'élément (i;j) est d'état mort
00292                 else:
00293                     # On ajoute la valeur mort au plateau
00294                     plateau[i].append(mort)
00295 
00296         # On retourne le plateau
00297         return plateau
00298 
```

meurt()

```
bool dependances.jdlv.scene.Scene.meurt (
             int nb_voisins)  [static]
```

```
Entrée:
    nb_voisins: int
Sortie:
    bool
Rôle:
    Retourne True si la cellule meurt
```

Définition à la ligne 353 du fichier scene.py.

```
00353     def meurt(nb_voisins: int) -> bool:
00354         """
00355         Entrée:
00356             nb_voisins: int
00357         Sortie:
00358             bool
00359         Rôle:
00360             Retourne True si la cellule meurt
00361         """
00362         # Retourne True si nb_voisins est différent de 2 ou 3
00363         return nb_voisins < 2 or nb_voisins > 3
00364
```

mouseMoveEvent()

```
None dependances.jdlv.scene.Scene.mouseMoveEvent (
              self,
             QGraphicsSceneMouseEvent event)
```

```
Réimplémentation de mouseMoveEvent hérité de QGraphicsScene
Entrées:
    self: Scene
    even: QGraphicsSceneMouseEvent
Sortie:
    None (modification en place)
Rôle:
    Capture l'événement QGraphicsSceneMouseEvent, le traite et le rend.
```

Définition à la ligne 174 du fichier scene.py.

```
00174     def mouseMoveEvent(self, event: QGraphicsSceneMouseEvent) -> None:
00175         """
00176         Réimplémentation de mouseMoveEvent hérité de QGraphicsScene
00177         Entrées:
00178             self: Scene
00179             even: QGraphicsSceneMouseEvent
00180         Sortie:
00181             None (modification en place)
00182         Rôle:
00183             Capture l'événement QGraphicsSceneMouseEvent, le traite et le rend.
00184         """
00185         # Abscisse de la souris
00186         x = event.scenePos().x()
00187         # Ordonnée de la souris
00188         y = event.scenePos().y()
00189         # Ordonnée du haut de la scène
00190         haut = self.sceneRect().top()
00191         # Ordonnée du bas de la scène
00192         bas = self.sceneRect().bottom()
00193         # Abscisse de côté gauche de la scène
00194         gauche = self.sceneRect().left()
00195         # Abscisse de côté droit de la scène
00196         droit = self.sceneRect().right()
00197         # Si la souris est dans le rectangle de la scène
00198         if gauche <= x < droit and haut <= y < bas:
00199             # Si la souris n'est pas encore entrée dans la scène
00200             if not self.est_souris_dans_scene:
```

```
00201                 # Pour chaque vue
00202                 for vue in self.views():
00203                     # On dit à la vue que la souris est partie
00204                     vue.event(QEvent(QEvent.Type.Leave))
00205             # On passe l'attribut est_souris_dans_scene à True
00206             self.est_souris_dans_scene = True
00207             # On calcul l'indice i de la cellule dans le tableau
00208             i = int((y - haut) // 50)
00209             # On calcul l'indice j de la cellule dans le tableau
00210             j = int((x - gauche) // 50)
00211             # Si le bouton gauche de la souris est pressé
00212             if event.buttons() is Qt.MouseButton.LeftButton:
00213                 # On définit l'état de la cellule touchée par la souris sur
00214                 # Vivant
00215                 self.tableau[i][j].set_etat(Etat.Vivant)
00216             # Si le bouton droit de la souris est pressé
00217             elif event.buttons() is Qt.MouseButton.RightButton:
00218                 # On définit l'état de la cellule touchée par la souris sur
00219                 # Mort
00220                 self.tableau[i][j].set_etat(Etat.Mort)
00221         # Si la souris n'est plus dans la scène et que l'attribut
00222         # est_souris_dans_scene est encore sur True
00223         elif self.est_souris_dans_scene:
00224             # On passe l'attribut est_souris_dans_scene sur False, puisqu'elle est
00225             # sortie
00226             self.est_souris_dans_scene = False
00227             # Pour chaque vue de la scène
00228             for vue in self.views():
00229                 # On dit à la vue que la souris est sortie de la scène, donc
00230                 # entrée dans la vue
00231                 vue.event(QEvent(QEvent.Type.Enter))
00232 
00233         # Rend l'événement à la classe mère
00234         return super().mouseMoveEvent(event)
00235 
```

nait()

```
bool dependances.jdlv.scene.Scene.nait (
             int nb_voisins)  [static]
```

```
Entrée:
    nb_voisins: int
Sortie:
    bool
Rôle:
    Retourne True si la cellule naît
```

Définition à la ligne 366 du fichier scene.py.

```
00366     def nait(nb_voisins: int) -> bool:
00367         """
00368         Entrée:
00369             nb_voisins: int
00370         Sortie:
00371             bool
00372         Rôle:
00373             Retourne True si la cellule naît
00374         """
00375         # Retourne True si nb_voisins est égal à 3
00376         return nb_voisins == 3
00377 
```

resultat()

```
Etat dependances.jdlv.scene.Scene.resultat (
              self,
             int ord,
             int absc)
```

```
Entrées:
    self: Scene
    ord: int (ordonnée)
    absc: int (abscisse)
Sortie:
    Etat
Rôle:
    Retourne l'état de la cellule en fonction du nombre de voisins
```

Définition à la ligne 378 du fichier scene.py.

```
00378     def resultat(self, ord: int, absc: int) -> Etat:
00379         """
00380         Entrées:
00381             self: Scene
00382             ord: int (ordonnée)
00383             absc: int (abscisse)
00384         Sortie:
00385             Etat
00386         Rôle:
00387             Retourne l'état de la cellule en fonction du nombre de voisins
00388         """
00389         # On calcul le nombre de voisin
00390         nb_voisins = self.total_voisins(ord, absc)
00391         # si la cellule est vivante
00392         if self.tableau[ord][absc].get_etat() is Etat.Vivant:
00393             # si la cellule meurt
00394             if self.meurt(nb_voisins):
00395                 # Retourne le nouvel état de la cellule
00396                 return Etat.Mort
00397             # si la cellule ne change pas d'état
00398             else:
00399                 # Retourne l'état de la cellule
00400                 return Etat.Vivant
00401         # si la cellule est morte
00402         else:
00403             # si la cellule naît
00404             if self.nait(nb_voisins):
00405                 # Retourne le nouvel état de la cellule
00406                 return Etat.Vivant
00407             # si la cellule ne change pas d'état
00408             else:
00409                 # Retourne l'état de la cellule
00410                 return Etat.Mort
00411
```

run()

```
None dependances.jdlv.scene.Scene.run (
              self)

Entrées:
    self: Scene
Sortie:
    None (modification en place)
Rôle:
    Lance l'animation et le chrono (séquenceur).
```

Définition à la ligne 661 du fichier scene.py.

```
00661     def run(self) -> None:
00662         """
00663         Entrées:
00664             self: Scene
00665         Sortie:
00666             None (modification en place)
00667         Rôle:
00668             Lance l'animation et le chrono (séquenceur).
00669         """
00670         # Si le chrono est activé
00671         if self.chrono.isActive():
00672             # Mise à jour de l'interval de temps
00673             self.chrono.setInterval(self.periode)
00674             # Execute un tour
00675             self.tour()
00676         # Si le chrono n'est pas encore activé
00677         else:
00678             # Démarre le chrono pour un temps indéterminé
00679             self.chrono.start()
00680             # Executre le premier tour
00681             self.tour()
```

set_tableau() [1/2]

```
None dependances.jdlv.scene.Scene.set_tableau (
              self,
             int height,
             int width,
             Etat etat)
```

```
Entrée:
    self: Scene
    height: int
    width: int
    etat: Etat
Sortie:
    None (modification en place)
Rôle:
    Créer un plateau de dimension height x width dans l'état etat.
```

Définition à la ligne 91 du fichier scene.py.

```
00091     def set_tableau(self, height: int, width: int, etat: Etat) -> None:
00092         """
00093         Entrée:
00094             self: Scene
00095             height: int
00096             width: int
00097             etat: Etat
00098         Sortie:
00099             None (modification en place)
00100         Rôle:
00101             Créer un plateau de dimension height x width dans l'état etat.
00102         """
00103         # On remet le plateau en ordre
00104         self.vide_scene()
00105         # On itère dans le nombre de ligne
00106         for i in range(height):
00107             # On ajoute une ligne
00108             self.tableau.append([])
00109             # On itère dans le nombre d'élément pas ligne
00110             for j in range(width):
00111                 # On créer une cellule
00112                 temp = Cellule()
00113                 # On ajuste sa position (w x h)
00114                 temp.setPos(j * 50, i * 50)
00115                 # On définit l'état de temp
00116                 temp.set_etat(etat)
00117                 # On ajoute la cellule au tableau
00118                 self.tableau[i].append(temp)
00119                 # On ajoute la cellule à la scène
00120                 self.addItem(temp)
00121
00122         # Copie le tableau dans tableau_precedent
00123         self.tableau_precedent = deepcopy(self.tableau)
00124
```

set_tableau() [2/2]

```
None dependances.jdlv.scene.Scene.set_tableau (
              self,
             list[list[Any]] tableau,
             Any vivant)
```

```
Surcharge de set_tableau
Entrées:
    self: Scene
    tableau: list[list[Any]]
    vivant: Any
Sortie:
    None (modification en place)
Rôle:
    Créer un plateau de même dimension que le tableau. Si l'élément
    vaut vivant, son état sera Vivant, sinon Mort.
```

Définition à la ligne 127 du fichier scene.py.

```
00127     def set_tableau(self, tableau: list[list[Any]], vivant: Any) -> None:
00128         """
00129         Surcharge de set_tableau
00130         Entrées:
00131             self: Scene
00132             tableau: list[list[Any]]
00133             vivant: Any
00134         Sortie:
00135             None (modification en place)
00136         Rôle:
00137             Créer un plateau de même dimension que le tableau. Si l'élément
00138             vaut vivant, son état sera Vivant, sinon Mort.
00139         """
00140         # Redéfinit la hateur du plateau
00141         self.dimension.setHeight(len(tableau))
00142         # Redéfinit la largeur du plateau
00143         self.dimension.setWidth(len(tableau[0]))
00144         # Enlève toutes les cellules de la scène
00145         self.vide_scene()
00146         # Déclaration d'un tableau vide
00147         self.tableau = []
00148         # On itère dans le nombre de ligne
00149         for i in range(len(tableau)):
00150             # On ajoute une ligne
00151             self.tableau.append([])
00152             # On itère dans le nombre d'élément pas ligne
00153             for j in range(len(tableau[i])):
00154                 # On créer une cellule
00155                 temp = Cellule()
00156                 # On ajuste sa position (w x h)
00157                 temp.setPos(j * 50, i * 50)
00158                 # Si la cellule (i;j) est vivante
00159                 if tableau[i][j] == vivant:
00160                     # On définit l'état de temp sur vivant
00161                     temp.set_etat(Etat.Vivant)
00162                 # Si la cellule (i;j) est mort
00163                 else:
00164                     # On définit l'état de temp sur mort
00165                     temp.set_etat(Etat.Mort)
00166                 # On ajoute la cellule au tableau
00167                 self.tableau[i].append(temp)
00168                 # On ajoute la cellule à la scène
00169                 self.addItem(temp)
00170
00171         # Copie le tableau dans tableau_precedent
00172         self.tableau_precedent = deepcopy(self.tableau)
00173
```

total_voisins()

```
int dependances.jdlv.scene.Scene.total_voisins (
              self,
             int ord,
             int absc)
```

```
Entrées:
    self: Scene
    ord: int
    absc: int
Sortie:
    int
Rôle:
    Retourne le total de voisins de la cellule (ord;absc)
```

Définition à la ligne 321 du fichier scene.py.

```
00321     def total_voisins(self, ord: int, absc: int) -> int:
00322         """
00323         Entrées:
00324             self: Scene
00325             ord: int
00326             absc: int
00327         Sortie:
00328             int
00329         Rôle:
00330             Retourne le total de voisins de la cellule (ord;absc)
```

```
00331         """
00332         # Récupère le voisin du bas (int(False) -> 0 et int(True) -> 1)
00333         b = int(self.valeur_case(ord + 1, absc))
00334         # Récupère le voisin du bas droit
00335         bd = int(self.valeur_case(ord + 1, absc + 1))
00336         # Récupère le voisin du bas gauche
00337         bg = int(self.valeur_case(ord + 1, absc - 1))
00338         # Récupère le voisin du haut
00339         h = int(self.valeur_case(ord - 1, absc))
00340         # Récupère le voisin du haut droit
00341         hd = int(self.valeur_case(ord - 1, absc + 1))
00342         # Récupère le voisin du haut gauche
00343         hg = int(self.valeur_case(ord - 1, absc - 1))
00344         # Récupère le voisin de droite
00345         d = int(self.valeur_case(ord, absc + 1))
00346         # Récupère le voisin de gauche
00347         g = int(self.valeur_case(ord, absc - 1))
00348 
00349         # retourne la somme des voisins
00350         return b + bd + bg + h + hd + hg + d + g
00351 
```

tour()

```
None dependances.jdlv.scene.Scene.tour (
              self)
```

```
Entrée:
    self: Scene
Sortie:
    None (modification en place)
Rôle:
    Execute un tour du jeu.
```

Définition à la ligne 604 du fichier scene.py.

```
00604     def tour(self) -> None:
00605         """
00606         Entrée:
00607             self: Scene
00608         Sortie:
00609             None (modification en place)
00610         Rôle:
00611             Execute un tour du jeu.
00612         """
00613         # économie de ressources:
00614         # Si l'auto stop est activé
00615         if self.auto_stop:
00616             # Copie le tableau dans tableau_precedent
00617             self.tableau_precedent = deepcopy(self.tableau)
00618 
00619         # On créer un tableau qui ne contient que les états.
00620         tableau_etat = construit(
00621             self.dimension.height(), self.dimension.width(), Etat.Mort)
00622 
00623         # On calcul l'état de chaque cellule
00624         # on itère dans les lignes
00625         for i in range(len(self.tableau)):
00626             # on itère dans la ligne i
00627             for j in range(len(self.tableau[i])):
00628                 # On stocke l'état dans le tableau d'état
00629                 tableau_etat[i][j] = self.resultat(i, j)
00630 
00631         # On met à jour le tableau
00632         # on itère dans les lignes
00633         for i in range(len(self.tableau)):
00634             # on itère dans la ligne i
00635             for j in range(len(self.tableau[i])):
00636                 # On met à jour l'état
00637                 self.tableau[i][j].set_etat(tableau_etat[i][j])
00638 
00639         # On incrémente le nombre de cycle effectué
00640         self.parent().nb_cycle += 1
00641         # Mise à jour du texte du label affichage_cycle
00642         self.parent().affichage_cycle.setText("Cycle n°" +
00643                                               str(self.parent().nb_cycle))
00644 
00645         # Si l'auto grandissement est activé:
```

```
00646         if self.auto_grandissement:
00647             # On cherche les directions vers lesquelles agrandir le tableau
00648             direction = self.doit_agrandir_tableau()
00649             # S'il faut agrandir
00650             if direction:  # vide -> False (conversion implicite par le if)
00651                 # On itère dans les directions vers lesquelles agrandir
00652                 for d in direction:
00653                     # Extension du tableau vers la direction d
00654                     self.extension(d)
00655 
00656         # Si l'auto stop est activé et que l'animation doit s'arrêter
00657         if self.auto_stop and self.arret_automatique():
00658             # On stop l'animation
00659             self.parent().stop_anim()
00660 
```

valeur_case()

```
bool dependances.jdlv.scene.Scene.valeur_case (
               self,
              int ord,
              int absc)
```

```
Entrées:
    self: Scene
    ord: int
    absc: int
Sortie:
    bool
Rôle:
    Retourne True si la cellule est vivante et donc est un voisin
    potentiel, False sinon.
```

Définition à la ligne 299 du fichier scene.py.

```
00299     def valeur_case(self, ord: int, absc: int) -> bool:
00300         """
00301         Entrées:
00302             self: Scene
00303             ord: int
00304             absc: int
00305         Sortie:
00306             bool
00307         Rôle:
00308             Retourne True si la cellule est vivante et donc est un voisin
00309             potentiel, False sinon.
00310         """
00311         # si les indices décrivent une valeur du tableau
00312         if 0 <= ord < len(self.tableau) and 0 <= absc < len(self.tableau[0]):
00313             # si True si vivant, False sinon
00314             return self.tableau[ord][absc].get_etat() is Etat.Vivant
00315         # si l'indice i ou j est trop grand ou négatif, ou si la cellule est
00316         # morte on retourne 0
00317         else:
00318             # Retourne False car l'élément inexistant est considéré mort
00319             return False
00320 
```

vide_scene()

```
None dependances.jdlv.scene.Scene.vide_scene (
               self)
```

```
Entrée:
    self: Scene
Sortie:
    None (modification en place)
Rôle:
    Retire les éléments de la scène
```

Définition à la ligne 73 du fichier scene.py.

```
00073     def vide_scene(self) -> None:
00074         """
00075         Entrée:
00076             self: Scene
00077         Sortie:
00078             None (modification en place)
00079         Rôle:
00080             Retire les éléments de la scène
00081         """
00082         # Pour chaque ligne du tableau
00083         for i in range(len(self.tableau)):
00084             # Pour chaque colonne du tableau
00085             for j in range(len(self.tableau[i])):
00086                 # On retire l'élément de la scène
00087                 self.removeItem(self.tableau[i][j])
00088
```

6.7.4 Documentation des données membres

auto_grandissement [1/2]

```
bool dependances.jdlv.scene.Scene.auto_grandissement = None
```

Définition à la ligne 50 du fichier scene.py.

auto_grandissement [2/2]

```
list[Direction] dependances.jdlv.scene.Scene.auto_grandissement = self.doit_agrandir_tableau()
```

Définition à la ligne 646 du fichier scene.py.

auto_stop

```
bool dependances.jdlv.scene.Scene.auto_stop = None
```

Définition à la ligne 52 du fichier scene.py.

chrono

```
dependances.jdlv.scene.Scene.chrono = QTimer(self)
```

Définition à la ligne 59 du fichier scene.py.

dimension

```
QSize dependances.jdlv.scene.Scene.dimension = taille
```

Définition à la ligne 48 du fichier scene.py.

est_souris_dans_scene

```
bool dependances.jdlv.scene.Scene.est_souris_dans_scene = False
```

Définition à la ligne 56 du fichier scene.py.

periode

```
dependances.jdlv.scene.Scene.periode = None
```

Définition à la ligne 54 du fichier scene.py.

run

```
dependances.jdlv.scene.Scene.run
```

Définition à la ligne 61 du fichier scene.py.

tableau

```
list dependances.jdlv.scene.Scene.tableau = []
```

Définition à la ligne 68 du fichier scene.py.

tableau_precedent

```
dependances.jdlv.scene.Scene.tableau_precedent = []
```

Définition à la ligne 66 du fichier scene.py.

La documentation de cette classe a été générée à partir du fichier suivant :

— D:/Jeu de la vie/source/dependances/jdlv/scene.py

7 Documentation des fichiers

7.1 Référence du fichier D:/Jeu de la vie/source/dependances/__init__.py

Espaces de nommage

— namespace dependances

7.2 __init__.py

Aller à la documentation de ce fichier.

```
00001
00008
00009 r"""
00010 Dépendences
00011 ===========
00012
00013 constantes
00014 ----------
00015 Constantes pour simplifier la lecture du code
00016
00017 jdlv
00018 ----
00019 Fonctions et classes relatives au Jeu de la vie
00020
00021 overload
00022 --------
00023 Permet de surcharger des méthodes (LSP)
00024 """
00025
00026 from dependances.constantes import *
00027 from dependances.jdlv import *
00028 from dependances.overload import *
```

7.3 Référence du fichier D:/Jeu de la vie/source/dependances/constantes/__init__.py

Espaces de nommage

- namespace dependances
- namespace dependances.constantes

7.4 __init__.py

Aller à la documentation de ce fichier.

```
00001
00008
00009 r"""
00010 Constantes
00011 ==========
00012 Etat
00013 ----
00014 Enumération pour les états d'une cellule (Vivant ou Mort)
00015 Direction
00016 ---------
00017 Enumération pour la direction d'extension du plateau
00018 """
00019
00020 from .constantes import Etat, Direction
```

7.5 Référence du fichier D:/Jeu de la vie/source/dependances/jdlv/__init__.py

Espaces de nommage

- namespace dependances
- namespace dependances.jdlv

7.6 __init__.py

Aller à la documentation de ce fichier.

```
00001
00008
00009 r"""
00010 JDLV UI
00011 =======
00012 Dépendences de jeu_de_la_vie_gui
00013
00014 Scene
00015 -----
00016 Scène du jeu de la vie
00017
00018 Cellule
00019 -------
00020 Représente une cellule du jeu de la vie
00021
00022 construit
00023 ---------
00024 Construit une matrice de taille désirée avec une valeur par défaut
00025
00026 est_template_valide
00027 --------------------
00028 Vérifie la validité d'un template pour devenir le nouveau tableau
00029 """
00030
00031 from .cellule import Cellule
00032 from .scene import Scene
00033 from .tableau import construit, est_template_valide
```

7.7 Référence du fichier D:/Jeu de la vie/source/dependances/overload/__init__.py

Espaces de nommage

- namespace dependances
- namespace dependances.overload

7.8 __init__.py

Aller à la documentation de ce fichier.

```
00001
00008
00009 r"""
00010 Surcharge selon le Principle de Substitution de Liskov
00011 ==================================================
00012 Cette librairie dérivée du module pythonlangutil.overload, disponible sur le
00013 dépots GitHub
      https://github.com/ehsan-keshavarzian/pythonlangutil/blob/master/pythonlangutil/overload.py
00014 """
00015
00016 from .overload import Overload, signature, get_signature_complete
```

7.9 Référence du fichier D:/Jeu de la vie/source/dependances/constantes/constantes.py

Classes

- class dependances.constantes.constantes.Etat
- class dependances.constantes.constantes.Direction

Espaces de nommage

— namespace dependances

— namespace dependances.constantes

— namespace dependances.constantes.constantes

7.10 constantes.py

Aller à la documentation de ce fichier.

```
00001 # Import Enum depuis enum
00002 from enum import Enum
00003
00004
00005 class Etat(Enum):
00006     """
00007     Hérite de:
00008         Enum
00009     Rôle:
00010         Abstaitiste les états de la cellule pour simplifier la lecture du code
00011     """
00012     # Déclaration de Vivant
00013     Vivant = True
00014     # Déclaration de Mort
00015     Mort = False
00016
00017
00018 class Direction(Enum):
00019     """
00020     Hérite de:
00021         Enum
00022     Rôle:
00023         Abstaitiste les directions pour simplifier la lecture du code
00024     """
00025     # Déclaration de Nord
00026     Nord = 0
00027     # Déclaration de Est
00028     Est = 1
00029     # Déclaration de Sud
00030     Sud = 2
00031     # Déclaration de Ouest
00032     Ouest = 3
```

7.11 Référence du fichier D:/Jeu de la vie/source/dependances/jdlv/cellule.py

Classes

— class dependances.jdlv.cellule.Cellule

Espaces de nommage

— namespace dependances

— namespace dependances.jdlv

— namespace dependances.jdlv.cellule

7.12 cellule.py

Aller à la documentation de ce fichier.

```
00001 # Importe les classes utilisées du module PySide6.QtWidgets
00002 from PySide6.QtWidgets import QGraphicsSceneHoverEvent, QGraphicsItem, \
00003     QGraphicsRectItem, QGraphicsSceneMouseEvent
00004 # Importe les classes utilisées du module PySide6.QtGui
00005 from PySide6.QtGui import QPen, QBrush, QStatusTipEvent
00006 # Importe les classes utilisées du module PySide6.QtCore
00007 from PySide6.QtCore import Qt
00008 # Importe Etat depuis dependances.constantes
00009 from dependances.constantes import Etat
00010 # Importe deepcopy depuis copy
00011 from copy import deepcopy
00012 # Importe Any depuis typing
00013 from typing import Any
00014
00015
00016 class Cellule(QGraphicsRectItem):
00017     """
00018     Hérite de:
00019         QGraphicsRectItem
00020     Rôle:
00021         Représente une cellule
00022     """
00023     def __init__(self, parent: QGraphicsItem | None = None,
00024                  etat: Etat = Etat.Vivant) -> None:
00025         """
00026         Entrée:
00027             self: Cellule
00028             parent QGraphicsItem | None
00029                 valeur par défaut: None
00030             etat: Etat
00031         Sortie:
00032             None (ctor)
00033         Rôle:
00034             Construit un nouvel objet Cellule
00035         """
00036         # Initialise la classe mère
00037         QGraphicsRectItem.__init__(self, parent)
00038         # L'objet accepte de recevoir les événements de survole de la souris
00039         self.setAcceptHoverEvents(True)
00040         # on définit l'état
00041         self.etat = etat
00042         # On peint la cellule en fonction de l'état
00043         self.peint()
00044         # Définit le rectangle d'affichage (x, y, w, h)
00045         self.setRect(0, 0, 50, 50)
00046         # Définit la couleur et taille du contour sur transparent et 0
00047         # car le contour agrandit le rectangle
00048         self.setPen(QPen(Qt.GlobalColor.transparent, 0))
00049
00050     def peint(self) -> None:
00051         """
00052         Entrée:
00053             self: Cellule
00054         Sortie:
00055             None (modification en place)
00056         Rôle:
00057             Définit la couleur de la cellule
00058         """
00059         # Si la cellule est vivante
00060         if self.etat is Etat.Vivant:
00061             # On remplit en noir
00062             self.setBrush(QBrush(Qt.GlobalColor.black))
00063         # si la cellule est morte
00064         else:
00065             # On remplit en blanc
00066             self.setBrush(QBrush(Qt.GlobalColor.white))
00067
00068     def mousePressEvent(self, even: QGraphicsSceneMouseEvent) -> None:
00069         """
00070         Réimplémentation de mousePressEvent hérité de QGraphicsRectItem
00071         Entrées:
00072             self: Cellule
00073             even: QGraphicsSceneMouseEvent (événement)
00074         Sortie:
00075             None (modification en place)
00076         Rôle:
00077             Détecte les événements de pression de souris sur la cellule.
00078         """
00079         # Si le click est un click gauche
00080         if even.button() is Qt.MouseButton.LeftButton:
00081             # Inverse l'état de la cellule
00082             self.etat = Etat.Vivant
00083         elif even.button() is Qt.MouseButton.RightButton:
```

```
00084             # Inverse l'état de la cellule
00085             self.etat = Etat.Mort
00086         # On peint la cellule en fonction de l'état
00087         self.peint()
00088 
00089         # On rend l'événement à la classe mère
00090         return super().mousePressEvent(even)
00091 
00092     def hoverEnterEvent(self, even: QGraphicsSceneHoverEvent) -> None:
00093         """
00094         Réimplémentation de hoverEnterEvent hérité de QGraphicsScene
00095         Entrées:
00096             self: Cellule
00097             even: QEvent (et les classes qui en hérite)
00098         Sortie:
00099             bool
00100         Rôle:
00101             Capture les événements de QGraphicsRectItem, les traites, puis les
00102             rends.
00103         """
00104         # Si la cellule est vivante
00105         if self.etat is Etat.Vivant:
00106             # Envoie un évènement QStatusTipEvent à la scène mère
00107             self.scene().event(QStatusTipEvent(
00108                 "Cliquez droit pour changer l'état. Pour modifier l'état de " +
00109                 "plusieurs cellules, maintenez le clique droit en déplaçant " +
00110                 "la souris."))
00111         # Si la cellule est morte
00112         else:
00113             # Envoie un évènement QStatusTipEvent à la scène mère
00114             self.scene().event(QStatusTipEvent(
00115                 "Cliquez gauche pour changer l'état. Pour modifier l'état " +
00116                 "de plusieurs cellules, maintenez le clique gauche en " +
00117                 "déplaçant la souris."))
00118 
00119         # Rend l'évènement
00120         return super().hoverEnterEvent(even)
00121 
00122     def set_etat(self, etat: Etat) -> None:
00123         """
00124         Entrées:
00125             self: Cellule
00126             etat: Etat
00127         Sortie:
00128             None (modification en place)
00129         Rôle:
00130             Redéfinit l'état de la cellule.
00131         """
00132         # On met à jour l'attribut est_vivant
00133         self.etat = etat
00134         # On peint la cellule en fonction de l'état
00135         self.peint()
00136 
00137     def get_etat(self) -> Etat:
00138         """
00139         Entrée:
00140             self: Cellule
00141         Sortie:
00142             Etat
00143         Rôle:
00144             Retourne l'état de la cellule.
00145         """
00146         # On retourne la valeur de l'attribut est_vivant
00147         return self.etat
00148 
00149     def __repr__(self) -> str:
00150         """
00151         Entrée:
00152             self: Cellule
00153         Sortie:
00154             str
00155         Rôle:
00156             Méthode magique __repr__, représentation de la cellule en str.
00157         """
00158         # Si la cellule est vivante
00159         if self.etat is Etat.Vivant:
00160             # Retourne ■
00161             return "■"
00162         # Si la cellule est morte
00163         else:
00164             # Retourne □
00165             return "□"
00166 
00167     def __deepcopy__(self, memo: dict) -> object:
00168         """
00169         Entrées:
00170             self: Cellule
```

```
00171             memo: dict
00172         Sortie:
00173             object (copie de self)
00174         Rôle:
00175             Méthode magique du module copy pour effectuer des copies de
00176             l'object.
00177         Addendum:
00178             Ne connaissant pas le module copy, j'ai utilisé la réponse
00179             suivante : https://stackoverflow.com/a/15774013/15793884
00180         """
00181         # classe de self (Cellule)
00182         cls = self.__class__
00183         # Créer une nouvelle instance de Cellule
00184         result = cls.__new__(cls)
00185         # ajoute au dictionnaire memo le couple identifant du présent object et
00186         # nouvel object
00187         memo[id(self)] = result
00188         # Pour chaque nom d'attribut et sa valeur de self (le présent object)
00189         for k, v in self.__dict__.items():
00190             # On affecte à l'attribut k de result une copie de la valeur de
00191             # l'attribut k
00192             setattr(result, k, deepcopy(v, memo))
00193 
00194         # Revoie l'object copié
00195         return result
00196 
00197     def __eq__(self, valeur: Any) -> bool:
00198         """
00199         Entrées:
00200             self: Cellule
00201             valeur: Any
00202         Sortie:
00203             bool
00204         Rôle:
00205             Méthode magique appelée par l'opérateur d'égalité ==.
00206         Explications:
00207             Une explication complète du fonctionnement de __eq__ est disponible
00208             via le lien suivant: https://stackoverflow.com/a/3588809/15793884
00209         """
00210         # Retoure True si etat est égal à valeur
00211         return self.etat == valeur
```

7.13 Référence du fichier D:/Jeu de la vie/source/dependances/jdlv/scene.py

Classes

— class dependances.jdlv.scene.Scene

Espaces de nommage

— namespace dependances

— namespace dependances.jdlv

— namespace dependances.jdlv.scene

7.14 scene.py

Aller à la documentation de ce fichier.

```
00001 # Importe les classes utilisées du module PySide6.QtWidgets
00002 from PySide6.QtWidgets import QGraphicsScene, QWidget, \
00003     QGraphicsSceneMouseEvent
00004 # Importe les classes utilisées du module PySide6.QtGui
00005 from PySide6.QtGui import QStatusTipEvent
00006 # Importe les classes utilisées du module PySide6.QtCore
00007 from PySide6.QtCore import Qt, QSize, QTimer, QEvent
00008 # Importe construit depuis tableau
00009 from .tableau import construit
00010 # Importe Any depuis typing
```

```
00011 from typing import Any
00012 # Importe Overload, signature depuis dependances.overload
00013 from dependances.overload import Overload, signature
00014 # Importe Etat, Direction depuis dependances.constantes
00015 from dependances.constantes import Etat, Direction
00016 # Importe Cellule depuis cellule
00017 from .cellule import Cellule
00018 # Importe deepcopy depuis copy
00019 from copy import deepcopy
00020
00021
00022 class Scene(QGraphicsScene):
00023     """
00024     Hérite de:
00025         QGraphicsScene
00026     Rôle:
00027         Représente la scène du jeu de la vie (Scene)
00028     """
00029     def __init__(self, parent: QWidget | None = None,
00030                  taille: QSize = QSize(0, 0)) -> None:
00031         """
00032         Entrées:
00033             self: Scene
00034             parent QWidget (ligne d'héritage pour désallouer la mémoire)
00035                 valeur par défaut: None
00036             taille: QSize
00037                 valeur par défaut: QSize(0, 0)
00038         Sortie:
00039             None (ctor)
00040         Rôle:
00041             Construit un nouvel objet JDLV
00042         """
00043         # On initialise la classe mère
00044         QGraphicsScene.__init__(self, parent)
00045
00046         # Déclaration de dimension qui représente la dimension du plateau
00047         # (w x h)
00048         self.dimension: QSize = taille
00049         # Déclaration d'un attribut auto grandissement
00050         self.auto_grandissementauto_grandissement: bool = None
00051         # Déclaration d'un attribut auto stop
00052         self.auto_stop: bool = None
00053         # Déclaration d'interval
00054         self.periode: int = None
00055         # Déclaration d'un attribut est_souris_dans_scene
00056         self.est_souris_dans_scene = False
00057
00058         # Déclaration d'un chronomètre
00059         self.chrono = QTimer(self)
00060         # Relie le signal à run
00061         self.chrono.timeout.connect(self.runrun)
00062         # Chrono moins précis mais moins gourmant en ressources
00063         self.chrono.setTimerType(Qt.TimerType.CoarseTimer)
00064
00065         # Déclaration d'un tableau à l'état n-1
00066         self.tableau_precedent: list[list[Cellule]] = []
00067         # Déclaration d'un tableau vide
00068         self.tableau: list[list[Cellule]] = []
00069         # On construit le plateau de jeu d'après les attributs de la fenêtre
00070         self.set_tableauset_tableau(
00071             self.dimension.height(), self.dimension.width(), Etat.Mort)
00072
00073     def vide_scene(self) -> None:
00074         """
00075         Entrée:
00076             self: Scene
00077         Sortie:
00078             None (modification en place)
00079         Rôle:
00080             Retire les éléments de la scène
00081         """
00082         # Pour chaque ligne du tableau
00083         for i in range(len(self.tableau)):
00084             # Pour chaque colonne du tableau
00085             for j in range(len(self.tableau[i])):
00086                 # On retire l'élément de la scène
00087                 self.removeItem(self.tableau[i][j])
00088
00089     @Overload  # On appelle overload pour créer une méthode surchargée
00090     @signature("int", "int", "Etat")  # On précise sa signature
00091     def set_tableau(self, height: int, width: int, etat: Etat) -> None:
00092         """
00093         Entrée:
00094             self: Scene
00095             height: int
00096             width: int
00097             etat: Etat
```

```
00098         Sortie:
00099             None (modification en place)
00100         Rôle:
00101             Créer un plateau de dimension height x width dans l'état etat.
00102         """
00103         # On remet le plateau en ordre
00104         self.vide_scene()
00105         # On itère dans le nombre de ligne
00106         for i in range(height):
00107             # On ajoute une ligne
00108             self.tableau.append([])
00109             # On itère dans le nombre d'élément pas ligne
00110             for j in range(width):
00111                 # On créer une cellule
00112                 temp = Cellule()
00113                 # On ajuste sa position (w x h)
00114                 temp.setPos(j * 50, i * 50)
00115                 # On définit l'état de temp
00116                 temp.set_etat(etat)
00117                 # On ajoute la cellule au tableau
00118                 self.tableau[i].append(temp)
00119                 # On ajoute la cellule à la scène
00120                 self.addItem(temp)
00121 
00122         # Copie le tableau dans tableau_precedent
00123         self.tableau_precedent = deepcopy(self.tableau)
00124 
00125     @set_tableau.overload  # On surcharge la méthode set_tableau
00126     @signature("list", "object")  # On précise sa signature
00127     def set_tableau(self, tableau: list[list[Any]], vivant: Any) -> None:
00128         """
00129         Surcharge de set_tableau
00130         Entrées:
00131             self: Scene
00132             tableau: list[list[Any]]
00133             vivant: Any
00134         Sortie:
00135             None (modification en place)
00136         Rôle:
00137             Créer un plateau de même dimension que le tableau. Si l'élément
00138             vaut vivant, son état sera Vivant, sinon Mort.
00139         """
00140         # Redéfinit la hateur du plateau
00141         self.dimension.setHeight(len(tableau))
00142         # Redéfinit la largeur du plateau
00143         self.dimension.setWidth(len(tableau[0]))
00144         # Enlève toutes les cellules de la scène
00145         self.vide_scene()
00146         # Déclaration d'un tableau vide
00147         self.tableau = []
00148         # On itère dans le nombre de ligne
00149         for i in range(len(tableau)):
00150             # On ajoute une ligne
00151             self.tableau.append([])
00152             # On itère dans le nombre d'élément pas ligne
00153             for j in range(len(tableau[i])):
00154                 # On créer une cellule
00155                 temp = Cellule()
00156                 # On ajuste sa position (w x h)
00157                 temp.setPos(j * 50, i * 50)
00158                 # Si la cellule (i;j) est vivante
00159                 if tableau[i][j] == vivant:
00160                     # On définit l'état de temp sur vivant
00161                     temp.set_etat(Etat.Vivant)
00162                 # Si la cellule (i;j) est mort
00163                 else:
00164                     # On définit l'état de temp sur mort
00165                     temp.set_etat(Etat.Mort)
00166                 # On ajoute la cellule au tableau
00167                 self.tableau[i].append(temp)
00168                 # On ajoute la cellule à la scène
00169                 self.addItem(temp)
00170 
00171         # Copie le tableau dans tableau_precedent
00172         self.tableau_precedent = deepcopy(self.tableau)
00173 
00174     def mouseMoveEvent(self, event: QGraphicsSceneMouseEvent) -> None:
00175         """
00176         Réimplémentation de mouseMoveEvent hérité de QGraphicsScene
00177         Entrées:
00178             self: Scene
00179             even: QGraphicsSceneMouseEvent
00180         Sortie:
00181             None (modification en place)
00182         Rôle:
00183             Capture l'événement QGraphicsSceneMouseEvent, le traite et le rend.
00184         """
```

```
00185         # Abscisse de la souris
00186         x = event.scenePos().x()
00187         # Ordonnée de la souris
00188         y = event.scenePos().y()
00189         # Ordonnée du haut de la scène
00190         haut = self.sceneRect().top()
00191         # Ordonnée du bas de la scène
00192         bas = self.sceneRect().bottom()
00193         # Abscisse de côté gauche de la scène
00194         gauche = self.sceneRect().left()
00195         # Abscisse de côté droit de la scène
00196         droit = self.sceneRect().right()
00197         # Si la souris est dans le rectangle de la scène
00198         if gauche <= x < droit and haut <= y < bas:
00199             # Si la souris n'est pas encore entrée dans la scène
00200             if not self.est_souris_dans_scene:
00201                 # Pour chaque vue
00202                 for vue in self.views():
00203                     # On dit à la vue que la souris est partie
00204                     vue.event(QEvent(QEvent.Type.Leave))
00205             # On passe l'attribut est_souris_dans_scene à True
00206             self.est_souris_dans_scene = True
00207             # On calcul l'indice i de la cellule dans le tableau
00208             i = int((y - haut) // 50)
00209             # On calcul l'indice j de la cellule dans le tableau
00210             j = int((x - gauche) // 50)
00211             # Si le bouton gauche de la souris est pressé
00212             if event.buttons() is Qt.MouseButton.LeftButton:
00213                 # On définit l'état de la cellule touchée par la souris sur
00214                 # Vivant
00215                 self.tableau[i][j].set_etat(Etat.Vivant)
00216             # Si le bouton droit de la souris est pressé
00217             elif event.buttons() is Qt.MouseButton.RightButton:
00218                 # On définit l'état de la cellule touchée par la souris sur
00219                 # Mort
00220                 self.tableau[i][j].set_etat(Etat.Mort)
00221         # Si la souris n'est plus dans la scène et que l'attribut
00222         # est_souris_dans_scene est encore sur True
00223         elif self.est_souris_dans_scene:
00224             # On passe l'attribut est_souris_dans_scene sur False, puisqu'elle est
00225             # sortie
00226             self.est_souris_dans_scene = False
00227             # Pour chaque vue de la scène
00228             for vue in self.views():
00229                 # On dit à la vue que la souris est sortie de la scène, donc
00230                 # entrée dans la vue
00231                 vue.event(QEvent(QEvent.Type.Enter))
00232 
00233         # Rend l'événement à la classe mère
00234         return super().mouseMoveEvent(event)
00235 
00236     def event(self, even: QEvent) -> bool:
00237         """
00238         Réimplémentation de event hérité de QGraphicsScene
00239         Entrées:
00240             self: Scene
00241             even: QEvent (et les classes qui en hérite)
00242         Sortie:
00243             bool
00244         Rôle:
00245             Capture les événements de QGraphicsScene, les traites, puis les
00246             rends.
00247         """
00248         # Si le type de l'évènement est StatusTipEvent
00249         if even.type() is QEvent.Type.StatusTip:
00250             # On précise le type de l'évènement pour être sûr de pouvoir
00251             # accéder aux méthodes (optionnel)
00252             even: QStatusTipEvent = even
00253             # Relais l'évènement à la classe mère
00254             self.parent().event(even)
00255             """
00256             Fonctionnement:
00257             Chaque objet possédant l'attribut statusTip émet un signal
00258             QStatusTipEvent lorsque la souris le survole. Ce signal comporte un
00259             tip, un conseil, concernant le dit objet. Il est par défaut
00260             réceptionné par statusBar de QMainWindow, mais il peut être
00261             intercepté et affiché autre part.
00262             """
00263 
00264         # On rend l'évènement à la classe mère
00265         return super().event(even)
00266 
00267     def get_tableau(self, vivant: Any, mort: Any) -> list[list[Any]]:
00268         """
00269         Entrées:
00270             self: Scene
00271             vivant: Any
```

```
00272           mort: Any
00273       Sortie:
00274           list[list[Any]]  (consitué uniquement de vivant et mort)
00275       Rôle:
00276           Retourne le plateau de jeu avec des valeurs personnalisées de
00277           vivant et mort.
00278       """
00279       # Déclaration d'un tableau vide plateau
00280       plateau: list[list] = []
00281       # Pour chaque ligne de tableau
00282       for i in range(len(self.tableau)):
00283           # Ajout d'une nouvelle ligne dans plateau
00284           plateau.append([])
00285           # Pour chaque élément de la ligne
00286           for j in range(len(self.tableau[i])):
00287               # Si l'élément (i;j) est d'état vivant
00288               if self.tableau[i][j].get_etat() is Etat.Vivant:
00289                   # On ajoute la valeur vivant au plateau
00290                   plateau[i].append(vivant)
00291               # Si l'élément (i;j) est d'état mort
00292               else:
00293                   # On ajoute la valeur mort au plateau
00294                   plateau[i].append(mort)
00295
00296       # On retourne le plateau
00297       return plateau
00298
00299   def valeur_case(self, ord: int, absc: int) -> bool:
00300       """
00301       Entrées:
00302           self: Scene
00303           ord: int
00304           absc: int
00305       Sortie:
00306           bool
00307       Rôle:
00308           Retourne True si la cellule est vivante et donc est un voisin
00309           potentiel, False sinon.
00310       """
00311       # si les indices décrivent une valeur du tableau
00312       if 0 <= ord < len(self.tableau) and 0 <= absc < len(self.tableau[0]):
00313           # si True si vivant, False sinon
00314           return self.tableau[ord][absc].get_etat() is Etat.Vivant
00315       # si l'indice i ou j est trop grand ou négatif, ou si la cellule est
00316       # morte on retourne 0
00317       else:
00318           # Retourne False car l'élément inexistant est considéré mort
00319           return False
00320
00321   def total_voisins(self, ord: int, absc: int) -> int:
00322       """
00323       Entrées:
00324           self: Scene
00325           ord: int
00326           absc: int
00327       Sortie:
00328           int
00329       Rôle:
00330           Retourne le total de voisins de la cellule (ord;absc)
00331       """
00332       # Récupère le voisin du bas (int(False) -> 0 et int(True) -> 1)
00333       b = int(self.valeur_case(ord + 1, absc))
00334       # Récupère le voisin du bas droit
00335       bd = int(self.valeur_case(ord + 1, absc + 1))
00336       # Récupère le voisin du bas gauche
00337       bg = int(self.valeur_case(ord + 1, absc - 1))
00338       # Récupère le voisin du haut
00339       h = int(self.valeur_case(ord - 1, absc))
00340       # Récupère le voisin du haut droit
00341       hd = int(self.valeur_case(ord - 1, absc + 1))
00342       # Récupère le voisin du haut gauche
00343       hg = int(self.valeur_case(ord - 1, absc - 1))
00344       # Récupère le voisin de droite
00345       d = int(self.valeur_case(ord, absc + 1))
00346       # Récupère le voisin de gauche
00347       g = int(self.valeur_case(ord, absc - 1))
00348
00349       # retourne la somme des voisins
00350       return b + bd + bg + h + hd + hg + d + g
00351
00352   @staticmethod
00353   def meurt(nb_voisins: int) -> bool:
00354       """
00355       Entrée:
00356           nb_voisins: int
00357       Sortie:
00358           bool
```

```
00359         Rôle:
00360             Retourne True si la cellule meurt
00361         """
00362         # Retourne True si nb_voisins est différent de 2 ou 3
00363         return nb_voisins < 2 or nb_voisins > 3
00364
00365     @staticmethod
00366     def nait(nb_voisins: int) -> bool:
00367         """
00368         Entrée:
00369             nb_voisins: int
00370         Sortie:
00371             bool
00372         Rôle:
00373             Retourne True si la cellule naît
00374         """
00375         # Retourne True si nb_voisins est égal à 3
00376         return nb_voisins == 3
00377
00378     def resultat(self, ord: int, absc: int) -> Etat:
00379         """
00380         Entrées:
00381             self: Scene
00382             ord: int (ordonnée)
00383             absc: int (abscisse)
00384         Sortie:
00385             Etat
00386         Rôle:
00387             Retourne l'état de la cellule en fonction du nombre de voisins
00388         """
00389         # On calcul le nombre de voisin
00390         nb_voisins = self.total_voisins(ord, absc)
00391         # si la cellule est vivante
00392         if self.tableau[ord][absc].get_etat() is Etat.Vivant:
00393             # si la cellule meurt
00394             if self.meurt(nb_voisins):
00395                 # Retourne le nouvel état de la cellule
00396                 return Etat.Mort
00397             # si la cellule ne change pas d'état
00398             else:
00399                 # Retourne l'état de la cellule
00400                 return Etat.Vivant
00401         # si la cellule est morte
00402         else:
00403             # si la cellule naît
00404             if self.nait(nb_voisins):
00405                 # Retourne le nouvel état de la cellule
00406                 return Etat.Vivant
00407             # si la cellule ne change pas d'état
00408             else:
00409                 # Retourne l'état de la cellule
00410                 return Etat.Mort
00411
00412     def doit_agrandir_tableau(self) -> list[Direction]:
00413         """
00414         Entrée:
00415             self: Scene
00416         Sortie:
00417             tuple[bool, Direction]
00418         Rôle:
00419             Retourne la liste des directions vers lesquelles il faut agrandir
00420             le tableau
00421         """
00422         # Déclaration d'une liste qui représente les directions vers lesquelles
00423         # il faut agrandir le tableau
00424         liste_directions = []
00425
00426         # on teste la frontière nord:
00427         i = 0
00428         # Déclaration de variable d'arrêt et initialisation à False
00429         stop = False
00430         # Pour chaque cellule de la première ligne
00431         while not (i >= self.dimension.width() or stop):
00432             # Si la cellule est vivante
00433             if self.tableau[0][i].get_etat() is Etat.Vivant:
00434                 # On arrête la boucle
00435                 stop = True
00436                 # on ajoute Nord au direction
00437                 liste_directions.append(Direction.Nord)
00438             # On icrémente l'itérateur
00439             i += 1
00440
00441         # on teste la frontière sud:
00442         i = 0
00443         # Initialisation de stop à False
00444         stop = False
00445         # On cherche l'indice de la dernière ligne
```

```
00446         derniere_ligne = self.dimension.height() - 1
00447         # Pour chaque cellule de la dernière ligne
00448         while not (i >= self.dimension.width() or stop):
00449             # Si la cellule est vivante
00450             if self.tableau[derniere_ligne][i].get_etat() is Etat.Vivant:
00451                 # On arrête la boucle
00452                 stop = True
00453                 # on ajoute Sud au direction
00454                 liste_directions.append(Direction.Sud)
00455             # On icrémente l'itérateur
00456             i += 1
00457 
00458         # on teste la frontière ouest:
00459         i = 0
00460         # Initialisation de stop à False
00461         stop = False
00462         # Pour chaque ligne
00463         while not (i >= self.dimension.height() or stop):
00464             # Si la dernière cellule est vivante
00465             if self.tableau[i][0].get_etat() is Etat.Vivant:
00466                 # On arrête la boucle
00467                 stop = True
00468                 # on ajoute Ouest au direction
00469                 liste_directions.append(Direction.Ouest)
00470             # On icrémente l'itérateur
00471             i += 1
00472 
00473         # on teste la frontière ouest:
00474         i = 0
00475         # Initialisation de stop à False
00476         stop = False
00477         # On cherche l'indice de la dernière colonne
00478         derniere_colonne = self.dimension.width() - 1
00479         # Pour chaque ligne
00480         while not (i >= self.dimension.height() or stop):
00481             # Si la dernière cellule est vivante
00482             if self.tableau[i][derniere_colonne].get_etat() is Etat.Vivant:
00483                 # On arrête la boucle
00484                 stop = True
00485                 # on ajoute Est au direction
00486                 liste_directions.append(Direction.Est)
00487             # On icrémente l'itérateur
00488             i += 1
00489 
00490         # Retourne la liste
00491         return liste_directions
00492 
00493     def extension(self, direction: Direction) -> None:
00494         """
00495         Entrées:
00496             self: Scene
00497             direction: Direction
00498         Sortie:
00499             None (modification en place)
00500         Rôle
00501             Étend le tableau vers une direction donnée
00502         """
00503         # Si la direction est Nord
00504         if direction is Direction.Nord:
00505             # Déclaration de abscisse
00506             abscisse = self.tableau[0][0].pos().x()
00507             # Déclaration de ordonnée
00508             ordonnée = self.tableau[0][0].pos().y() - 50
00509             # On ajoute une ligne à l'indice 0 (déplace les éléments)
00510             self.tableau.insert(0, [])
00511             # Pour chaque colonne
00512             for i in range(self.dimension.width()):
00513                 # On créer une cellule
00514                 temp = Cellule()
00515                 # On ajuste sa position (w x h)
00516                 temp.setPos(abscisse + i * 50, ordonnée)
00517                 # On définit l'état de temp
00518                 temp.set_etat(Etat.Mort)
00519                 # On ajoute la cellule au tableau
00520                 self.tableau[0].append(temp)
00521                 # On ajoute la cellule à la scène
00522                 self.addItem(temp)
00523 
00524             # On ajuste la dimension du tableau
00525             self.dimension.setHeight(self.dimension.height() + 1)
00526         # Si la direction est Sud
00527         elif direction is Direction.Sud:
00528             # Déclaration de abscisse
00529             abscisse = self.tableau[0][0].pos().x()
00530             # Déclaration de ordonnée
00531             ordonnée = self.tableau[-1][0].pos().y() + 50
00532             # On ajoute une ligne à la fin du tableau
```

```
00533             self.tableau.append([])
00534             # Pour chaque colonne
00535             for i in range(self.dimension.width()):
00536                 # On créer une cellule
00537                 temp = Cellule()
00538                 # On ajuste sa position (w x h)
00539                 temp.setPos(abscisse + i * 50, ordonnée)
00540                 # On définit l'état de temp
00541                 temp.set_etat(Etat.Mort)
00542                 # On ajoute la cellule au tableau
00543                 self.tableau[-1].append(temp)
00544                 # On ajoute la cellule à la scène
00545                 self.addItem(temp)
00546 
00547             # On ajuste la dimension du tableau
00548             self.dimension.setHeight(self.dimension.height() + 1)
00549         # Si la direction est Est
00550         elif direction is Direction.Est:
00551             # Déclaration de ordonnée
00552             ordonnée = self.tableau[0][0].pos().y()
00553             # Déclaration de abscisse
00554             abscisse = self.tableau[0][-1].pos().x() + 50
00555             # Pour chaque ligne
00556             for i in range(self.dimension.height()):
00557                 # On créer une cellule
00558                 temp = Cellule()
00559                 # On ajuste sa position (w x h)
00560                 temp.setPos(abscisse, ordonnée + i * 50)
00561                 # On définit l'état de temp
00562                 temp.set_etat(Etat.Mort)
00563                 # On ajoute la cellule au tableau
00564                 self.tableau[i].append(temp)
00565                 # On ajoute la cellule à la scène
00566                 self.addItem(temp)
00567 
00568             # On ajuste la dimension du tableau
00569             self.dimension.setWidth(self.dimension.width() + 1)
00570         # Si la direction est Ouest
00571         elif direction is Direction.Ouest:
00572             # Déclaration de ordonnée
00573             ordonnée = self.tableau[0][0].pos().y()
00574             # Déclaration de abscisse
00575             abscisse = self.tableau[0][0].pos().x() - 50
00576             # Pour chaque ligne
00577             for i in range(self.dimension.height()):
00578                 # On créer une cellule
00579                 temp = Cellule()
00580                 # On ajuste sa position (w x h)
00581                 temp.setPos(abscisse, ordonnée + i * 50)
00582                 # On définit l'état de temp
00583                 temp.set_etat(Etat.Mort)
00584                 # On ajoute la cellule au tableau
00585                 self.tableau[i].insert(0, temp)
00586                 # On ajoute la cellule à la scène
00587                 self.addItem(temp)
00588 
00589             # On ajuste la dimension du tableau
00590             self.dimension.setWidth(self.dimension.width() + 1)
00591 
00592     def arret_automatique(self) -> bool:
00593         """
00594         Entrée:
00595             self: Scene
00596         Sortie:
00597             bool
00598         Rôle:
00599             Vérifie si deux tour de suite sont identique
00600         """
00601         # retourne True si le tableau n'a pas changé entre deux cycles
00602         return self.tableau_precedent == self.tableau
00603 
00604     def tour(self) -> None:
00605         """
00606         Entrée:
00607             self: Scene
00608         Sortie:
00609             None (modification en place)
00610         Rôle:
00611             Execute un tour du jeu.
00612         """
00613         # économie de ressources:
00614         # Si l'auto stop est activé
00615         if self.auto_stop:
00616             # Copie le tableau dans tableau_precedent
00617             self.tableau_precedent = deepcopy(self.tableau)
00618 
00619         # On créer un tableau qui ne contient que les états.
```

```
00620         tableau_etat = construit(
00621             self.dimension.height(), self.dimension.width(), Etat.Mort)
00622 
00623         # On calcul l'état de chaque cellule
00624         # on itère dans les lignes
00625         for i in range(len(self.tableau)):
00626             # on itère dans la ligne i
00627             for j in range(len(self.tableau[i])):
00628                 # On stocke l'état dans le tableau d'état
00629                 tableau_etat[i][j] = self.resultat(i, j)
00630 
00631         # On met à jour le tableau
00632         # on itère dans les lignes
00633         for i in range(len(self.tableau)):
00634             # on itère dans la ligne i
00635             for j in range(len(self.tableau[i])):
00636                 # On met à jour l'état
00637                 self.tableau[i][j].set_etat(tableau_etat[i][j])
00638 
00639         # On incrémente le nombre de cycle effectué
00640         self.parent().nb_cycle += 1
00641         # Mise à jour du texte du label affichage_cycle
00642         self.parent().affichage_cycle.setText("Cycle n°" +
00643                                               str(self.parent().nb_cycle))
00644 
00645         # Si l'auto grandissement est activé:
00646         if self.auto_grandissementauto_grandissement:
00647             # On cherche les directions vers lesquelles agrandir le tableau
00648             direction = self.doit_agrandir_tableau()
00649             # S'il faut agrandir
00650             if direction:  # vide -> False (conversion implicite par le if)
00651                 # On itère dans les directions vers lesquelles agrandir
00652                 for d in direction:
00653                     # Extension du tableau vers la direction d
00654                     self.extension(d)
00655 
00656         # Si l'auto stop est activé et que l'animation doit s'arrêter
00657         if self.auto_stop and self.arret_automatique():
00658             # On stop l'animation
00659             self.parent().stop_anim()
00660 
00661     def run(self) -> None:
00662         """
00663         Entrées:
00664             self: Scene
00665         Sortie:
00666             None (modification en place)
00667         Rôle:
00668             Lance l'animation et le chrono (séquenceur).
00669         """
00670         # Si le chrono est activé
00671         if self.chrono.isActive():
00672             # Mise à jour de l'interval de temps
00673             self.chrono.setInterval(self.periode)
00674             # Execute un tour
00675             self.tour()
00676         # Si le chrono n'est pas encore activé
00677         else:
00678             # Démarre le chrono pour un temps indéterminé
00679             self.chrono.start()
00680             # Executre le premier tour
00681             self.tour()
```

7.15 Référence du fichier D:/Jeu de la vie/source/dependances/jdlv/tableau.py

Espaces de nommage

— namespace dependances

— namespace dependances.jdlv

— namespace dependances.jdlv.tableau

Fonctions

— list[list[Any]] dependances.jdlv.tableau.construit (int h, int l, Any valeur_defaut)

— tuple[bool, str] dependances.jdlv.tableau.est_template_valide (Any tableau)

7.16 tableau.py

Aller à la documentation de ce fichier.

```
00001 from typing import Any
00002 from re import search
00003
00004
00005 def construit(h: int, l: int, valeur_defaut: Any) -> list[list[Any]]:
00006     """
00007     Entrée:
00008         h: int (hauteur)
00009         l: int (largeur)
00010     Sortie:
00011         list[list[int]]
00012     Rôle:
00013         Construit un tableau de taille taille avec comme valeur valeur_defaut
00014     """
00015     # Déclaration d'un tableau vide
00016     plateau = []
00017
00018     # Pour chaque ligne de la matrice (de hauteur h)
00019     for i in range(h):
00020         # On ajoute une nouvelle ligne au tableau
00021         plateau.append([])
00022         # Pour chaque élément de la ligne (de longeur l)
00023         for _ in range(l):
00024             # On ajoute la valeur par défaut à la ligne du tableau
00025             plateau[i].append(valeur_defaut)
00026
00027     # On retourne le tableau
00028     return plateau
00029
00030
00031 def est_template_valide(tableau: Any) -> tuple[bool, str]:
00032     """
00033     Entrée:
00034         tableau: list[str]  (attendu)
00035     Sortie:
00036         tuple[
00037             bool,    (validité)
00038             str      (message d'erreur)
00039         ]
00040     Rôle:
00041         Retourne True si le tableau respecte le format d'un template. En cas
00042         d'erreur, le deuxième élément sera le message qui explique l'erreur.
00043     """
00044     message = ""
00045     # Si le tableau est de type list
00046     if type(tableau) is list:
00047         # Déclare taille et l'initialise à -1
00048         taille = -1
00049         # Déclaration d'une variable d'arrêt
00050         lignes_valide = True
00051         # Déclaration d'un itérateur
00052         i = 0
00053         # Pour chaque ligne du tableau tant que les lignes sont valides
00054         while i < len(tableau) and lignes_valide:
00055             # Si la ligne du tableau est une chaîne de caractères
00056             if type(tableau[i]) is str:
00057                 # On cherche l'expression régulière suivante dans la ligne
00058                 # ^ : on part du premier caractère
00059                 # $ : on s'arrête au dernier caractère
00060                 # 0-1 : l'élément est soit 0 soit 1
00061                 # , : il est suivit d'une virgule
00062                 # * : il est présent 0 ou plus fois
00063                 # \\n : la ligne finit par \n
00064                 correspondance = search("^[0-1,]*\\n$", tableau[i])
00065
00066                 # S'il y a une correspondance
00067                 if correspondance:
00068                     # Si la taille n'a pas encore été définit
00069                     if taille == -1:
00070                         # On définit la taille d'une ligne
00071                         taille = len(tableau[i])
00072                     # Si la ligne ne fait pas la même taille que la première
00073                     if len(tableau[i]) != taille:
00074                         # On définit un message d'erreur
00075                         message = "Les lignes doivent être de même taille"
00076                         # On arrête la boucle
00077                         lignes_valide = False
```

```
00078                 # S'il n'y a pas de correspondance
00079                 else:
00080                     # On définit un message d'erreur
00081                     message = "Les éléments doivent être 0 ou 1. Ils " +\
00082                         "doivent être séparés par des virgules."
00083                     # On arrête la boucle
00084                     lignes_valide = False
00085             else:
00086                 # On définit un message d'erreur
00087                 message = "Chaque ligne doit être une chaîne de caractère " +\
00088                     "encodant la valeurs des cellules."
00089                 # On arrête la boucle
00090                 lignes_valide = False
00091 
00092             # Incrémentation de l'itérateur
00093             i += 1
00094 
00095         # On retourne True si toutes les lignes sont valides, False sinon
00096         return lignes_valide, message
00097 
00098     # On retourne False sinon
00099     return False, message
```

7.17 Référence du fichier D:/Jeu de la vie/source/dependances/overload/overload.py

Classes

— class dependances.overload.overload.Overload

Espaces de nommage

— namespace dependances

— namespace dependances.overload

— namespace dependances.overload.overload

Fonctions

— callable dependances.overload.overload.signature (*types)

— tuple[str] dependances.overload.overload.get_signature_complete (obj, tuple sig=())

7.18 overload.py

Aller à la documentation de ce fichier.

```
00001 # Dérivé de pythonlangutil.overload
00002 # https://github.com/ehsan-keshavarzian/pythonlangutil/blob/master/pythonlangutil/overload.py
00003 
00004 # Importe Any depuis typing
00005 from typing import Any
00006 
00007 
00008 def signature(*types) -> callable:
00009     """
00010     Entrée(s):
00011         types: str
00012     Sortie:
00013         callable
00014     Rôle:
00015         Retourne la fonction dont signature est un décorateur avec une
00016         propriété signature qui est un tuple des types
00017     Addendum:
```

```
00018         Mes excuses pour les explications et commentaires, mais le code n'étant
00019         pas le mien, non commenté et parfois hors de ma porté, il a été
00020         complexe à comprendre. La notion de décoration en python n'est déjà pas
00021         facile à implémenter, alors à mettre en oeuvre, c'est une autre
00022         histoire.
00023     """
00024     def func(f: callable) -> callable:
00025         """
00026         Entrée:
00027             f: callable (fonction qui est décorée)
00028         Sortie:
00029             callable
00030         Rôle:
00031             Retourne la fonction dont signature est un décorateur avec une
00032             propriété signature qui est un tuple des types
00033         """
00034         def inner_func(callingObj, *args, **kwargs) -> callable:
00035             """
00036             Entrées:
00037                 callingObj: self (puisque f est une méthode Cf Overload)
00038                 *args: arguments passé à f lors de son appelle
00039                 **kwargs: couples (clé;argument) passé à f lors de son appelle
00040             Sortie:
00041                 Sortie de f(callingObj, *args, **kwargs)
00042             Rôle:
00043                 Retourne la valeur de f(callingObj, *args, **kwargs)
00044             """
00045             # retourne la valeur de retour de f, la fonction décorée, en lui
00046             # passant en paramètre les arguments qui lui sont passés en
00047             # paramètre
00048             return f(callingObj, *args, **kwargs)
00049 
00050         # Créer une propriété signature pour la fonction décorée et lui assigne
00051         # le tuple de types
00052         inner_func.signature = types  # types -> tuple
00053 
00054         # retourn l'objet inner_func
00055         return inner_func
00056 
00057     # retourne l'objet func
00058     return func
00059 
00060 
00061 def get_signature_complete(obj, sig: tuple = ()) -> tuple[str]:
00062     """
00063     Entrées:
00064         obj: Any
00065             obj est une variable quelconque
00066         sig: tuple
00067             signature de obj jusqu'alors
00068     Sortie:
00069         tuple[str]
00070     Rôle:
00071         Retourne l'arbre de signature de obj
00072     Exemple:
00073         >> a: int = 1
00074         >> get_inherited_signature(a)
00075         ['int', 'object']
00076     Explications:
00077         Le but de cette fonction est de savoir de quelleS classeS hérite obj.
00078         Plus obj est abstrait, plus sont arbre de signature sera grand.
00079         L'arbre ne possède qu'une dimension pour facilité son traitement.
00080     Addendum:
00081         Cette fonction est la raison pour laquelle j'ai recodé ce module.
00082         Ce dernier ne permet pas de faire de surcharge 'intelligente'.
00083         Il est évident les parmatères suivants de type:
00084             list, int, int
00085         correspond à la signature suivante:
00086             list, int, object
00087         car int hérite de object. Or l'héritage doit respecter le principe de
00088         substitution de Liskov.
00089     """
00090     # Si c'est le premier appelle
00091     if len(sig) == 0:
00092         # On redéfinit obj par sa classe
00093         obj = obj.__class__
00094         # on attribut le singleton du nom de la classe d'obj à sig
00095         sig = (obj.__name__,)
00096         # On retourne get_inherited_signature(obj, sig)
00097         return get_signature_complete(obj, sig)
00098     # Si obj est object
00099     elif obj.__name__ == object.__name__:
00100         # On retourne la signature complète plus object
00101         return sig + (obj.__name__,)
00102     # Sinon
00103     else:
00104         # parent_sig est initialisé avec les classes directement parentes d'obj
```

```
00105         parent_sig = obj.__bases__
00106         # Si obj à plus d'un parent directe
00107         if type(parent_sig) is tuple:
00108             # Pour chaque parent
00109             for par in parent_sig:
00110                 # Si le parent est object
00111                 if par.__name__ == object.__name__:
00112                     # on ajoute ('object',) à sig
00113                     sig += (par.__name__,)
00114                 # Sinon
00115                 else:
00116                     # on ajoute l'arbre de signature du parent
00117                     sig += get_signature_complete(par, (par.__name__,))
00118             # On retourne l'arbre de signature de obj
00119             return sig
00120         # Si sig n'a qu'un parent directe
00121         else:
00122             # On retourne sig plus son type plus son arbre de signature
00123             return sig  + (obj.__name__,) + (
00124                 get_signature_complete(parent_sig, sig),)
00125
00126
00127 class Overload(object):
00128     """
00129     Hérite de:
00130         object
00131     Rôle:
00132         Permet de surcharger une (uniquement) méthode (uniquement) et met en
00133         oeuvre le principe de substitution de Liskov.
00134     Exemple:
00135         >> @Overload  # (il est préférable de l'utiliser en décorateur)
00136         >> @signature(int)  # (obligatoire !!!)
00137         >> def double(valeur: int) -> int:
00138         >>     return int(valeur * 2)
00139         >> @double.overload  # Pas de majuscule à overload
00140         >> @signature(float)  # (obligatoire !!!)
00141         >> def double(valeur: float) -> float:
00142         >>     return float(valeur * 2)
00143         >> double(.2)  # .2 est de type float
00144         0.4  # float
00145         >> double(5)  # 5 est de type int
00146         10  # int
00147     Addendum:
00148         Overload est dérivée, comme signature, de pythonlangutil.overload donc
00149         les variables sont en englais.
00150     """
00151     def __init__(self, func: callable) -> None:
00152         """
00153         Entrées:
00154             self: Overload
00155             func: callable (fonction à surcharger)
00156         Sortie:
00157             None (ctor)
00158         Rôle:
00159             Construit un nouvel objet Overload
00160         """
00161         # Déclaration de l'attribut owner pour la méhtode magique __get__
00162         self.owner = None
00163         # Déclaration d'une liste de tuple où les tuples sont les signatures
00164         # des méthodes qui se surchargent
00165         self.signatures: list[tuple] = []
00166         # Liste des méthodes qui se surchargent
00167         self.methods: list[callable] = []
00168         # On ajoute la méthode surchargée à methods
00169         self.methods.append(func)
00170         # On ajoute la signature de la méthode à signatures
00171         self.signatures.append(func.signature)
00172
00173     def __get__(self, owner: object | None, ownerType: Any = None) -> object:
00174         """
00175         Entrée:
00176             self: Overload
00177             owner: object | None
00178             owerType: None | Any
00179         Sortie:
00180             Overload
00181         Rôle:
00182             Retourne l'objet Overload et définit l'attribut owner.
00183         Description:
00184     https://python-reference.readthedocs.io/en/latest/docs/dunderdsc/get.html
00185         """
00186         # owner n'est pas définit, self.owner = self, sinon, self.owner = owner
00187         self.owner = owner or self
00188
00189         # retourne l'objet Overload
00190         return self
00191
```

```
00192     def __call__(self, *args, **kwargs):
00193         """
00194         Entrée:
00195             self: Overload
00196             args: Any  (arguments passés à la fonction surchargée)
00197             kwargs: Any (couples (clé;arg) passés à la fonction surchargée)
00198         Sortie:
00199             Résultat de la fonction surchargé qui correspond à la signature des
00200             arguments passés en paramètre.
00201         Rôle:
00202             Trouve la fonction correspondant à la signature des arguments
00203             passés en paramètre, l'execute et retourne sa valeur de retour.
00204         """
00205         # Signature des arguments
00206         signature = []
00207         # Pour chaque argument sans clé
00208         for arg in args:
00209             # On ajoute le type des arguments sans clé
00210             signature.append(arg.__class__.__name__)
00211         # Pour chaque clé (_), argument (v)
00212         for _, v in kwargs:
00213             # On ajoute le type des arguments qui ont une clé
00214             signature.append(v.__class__.__name__)
00215         # Signature doit être un tuple car les signatures sont stockée sous
00216         # forme de tuple
00217         signature = tuple(signature)
00218         # si la signature exacte est enregistrée
00219         if signature in self.signatures:
00220             # On trouve l'indice correspondant à cet signature
00221             index = self.signatures.index(signature)
00222         # si la signature exacte est n'est pas enregistrée
00223         else:
00224             # extrait l'arbre d'héritage pour chaque argument de la signature
00225             inherited_signature = []
00226             # Pour chaque argument sans clé
00227             for arg in args:
00228                 # On ajoute l'arbre d'héritage des arguments sans clé
00229                 inherited_signature.append(get_signature_complete(arg))
00230             # Pour chaque clé (_), argument (v)
00231             for _, v in kwargs:
00232                 # On ajoute l'arbre d'héritage des arguments qui ont une clé
00233                 signature.append(get_signature_complete(v))
00234 
00235             # signature éligibles
00236             sig_eligibles = []
00237             # Pour chaque signature des signatures enregistrées
00238             for sig in self.signatures:
00239                 # Si son nombre d'argument correspond à la signature des args
00240                 if len(inherited_signature) == len(sig):
00241                     # On l'ajoute aux signature éligibles
00242                     sig_eligibles.append(sig)
00243 
00244             # Index négatif -> erreur
00245             index = -1
00246             # Variable d'arrêt, False s'il n'y a pas de signature éligible
00247             existe = len(sig_eligibles) > 0  # bool
00248             # Itérateur i
00249             i = 0
00250             # Pour chaque signature
00251             while i < len(sig_eligibles) and existe:
00252                 # Itérateur j
00253                 j = 0
00254                 # Variable d'arrêt de la seconde boucle
00255                 correspond = True
00256                 # Pour chaque type de la signature
00257                 while j < len(sig_eligibles[i]) and correspond:
00258                     # Si le type n'est pas dans les type hérité du paramètre
00259                     # c'est que ce n'est pas le bon type
00260                     if sig_eligibles[i][j] not in inherited_signature[j]:
00261                         # On stop la boucle
00262                         correspond = False
00263                     # Incrémentation de l'itérateur
00264                     j += 1
00265                 # Si la signature correspond
00266                 if correspond:
00267                     # On recherche l'indice car la signature est unique
00268                     # (cf self.overload)
00269                     index = self.signatures.index(sig_eligibles[i])
00270                 # Incrémentation de l'itérateur
00271                 i += 1
00272             # Si la signature ne correspond à aucune des méthodes enregistrées
00273             if not existe or index < 0:
00274                 # On lève une exception
00275                 raise Exception("There is no overload for this method with " +
00276                                 "this signature.")
00277 
00278         # On execute la méthode avec ses arguments et on retourne sa valeur de
```

```
00279         # retour
00280         return self.methods[index](self.owner, *args, **kwargs)
00281
00282     def overload(self, func: callable):
00283         """
00284         Entrée:
00285             self: Overload
00286             func: callable
00287         Sortie:
00288             Overload
00289         Rôle:
00290             Ajoute une nouvelle surcharge
00291         Explications:
00292             Faire:
00293             >> @Overload
00294             >> @signature(int)
00295             >> def double(valeur: int) -> int:
00296             >>     return int(valeur * 2)
00297             C'est faire:
00298             >> def double(valeur: int) -> int:
00299             >>     return int(valeur * 2)
00300             >> double = signature('int', double)
00301             >> double = Overload(double)
00302
00303             Donc quand on fait
00304             >> @double.overload  # appelle la méthode overload
00305             >> @signature(float)
00306             >> def double2(valeur: float) -> float:
00307             >>     return float(valeur * 2)
00308             On fait:
00309             >> def double2(valeur: float) -> float:
00310             >>     return float(valeur * 2)
00311             >> double2 = signature(double2)
00312             >> double2 = double.overload(double2)# appelle la méthode overload
00313             En pratique, il ne faut pas changer le nom de la méthode quand on
00314             la surcharge.
00315
00316             https://www.geeksforgeeks.org/decorators-in-python/
00317         """
00318         # Si la signature est unique
00319         if func.signature not in self.signatures:
00320             # On ajoute la méthode à methods
00321             self.methods.append(func)
00322             # On ajoute la signature de func à signatures
00323             self.signatures.append(func.signature)
00324             # Retourne l'objet Overload
00325             return self
00326         # Si la signature n'est pas unique
00327         else:
00328             # On lève une exception
00329             raise Exception("There is no overload for this method with this" +
00330                             " signature: the signature is already taken.")
```

7.19 Référence du fichier D:/Jeu de la vie/source/jeu_de_la_vie_cli.py

Classes

— class jeu_de_la_vie_cli.JeuDeLaVieCLI

Espaces de nommage

— namespace jeu_de_la_vie_cli

Variables

— jeu_de_la_vie_cli.jeu = JeuDeLaVieCLI()

7.20 jeu_de_la_vie_cli.py

Aller à la documentation de ce fichier.

```
00001 # Importe sleep depuis time
00002 from time import sleep
00003 # Importe system depuis os
00004 from os import system
00005 # importe exists depuis os.path
00006 from os.path import exists
00007 # Importe Any depuis typing
00008 from typing import Any, Literal
00009 # Importe copie depuis dependances.plateau
00010 from dependances.jdlv import est_template_valide
00011 # Importe argv, version_info depuis le module sys
00012 from sys import version_info
00013 # Importe reader depuis csv
00014 from csv import reader
00015 # Importe deepcopy depuis copy
00016 from copy import deepcopy
00017
00018
00019 class JeuDeLaVieCLI(object):
00020     """
00021     Hérite de:
00022         object
00023     Rôle:
00024         Représente le jeu de la vie.
00025     """
00026
00027     def __init__(self, plateau: list[list[Literal[1] | Literal[0]]] = [])\
00028         -> None:
00029         """
00030         Entrées:
00031             self: JeuDeLaVieCLI
00032             tableau: list[list[Literal[1] | Literal[0]]]
00033                 valeur par défaut: []
00034         Sortie:
00035             None (modification en place)
00036         Rôle:
00037             Construit un nouvel objet JeuDeLaVieCLI.
00038         """
00039         # Déclaration d'un tableau au cycle n-1, initialisé vide
00040         self.tableau_precedenttableau_precedent: list[list[Literal[1] | Literal[0]]] = []
00041         # Déclaration de tableau et l'initialise à plateau
00042         self.tableautableautableautableau: list[list[Literal[1] | Literal[0]]] = plateau
00043         # Déclaration de symbol_mort et l'initialise à □
00044         self.symbole_mort = "□"
00045         # Déclaration de symbole_vivant et l'initialise à ■
00046         self.symbole_vivant = "■"
00047
00048     def set_symbole_mort(self, symb_mort: Any) -> None:
00049         """
00050         Entrées:
00051             self: JeuDeLaVieCLI
00052             symb_mort: Any
00053         Sortie:
00054             None (modification en place)
00055         Rôle:
00056             Redéfinit le symbole des cases mortes (affichage_complexe).
00057         """
00058         # définie le nouveau symbole d'une cellule morte
00059         self.symbole_mort = str(symb_mort)
00060
00061     def set_symbole_vivant(self, symb_vivant: Any) -> None:
00062         """
00063         Entrées:
00064             self: JeuDeLaVieCLI
00065             symb_vivant: Any
00066         Sortie:
00067             None (modification en place)
00068         Rôle:
00069             Redéfinit le symbole des cases vivantes (affichage_complexe).
00070         """
00071         # définie le nouveau symbole d'une cellule morte
00072         self.symbole_vivant = str(symb_vivant)
00073
00074     @staticmethod
00075     def est_tableau_valide(plateau: Any) -> bool:
00076         """
00077         Entrée:
00078             plateau: list[list[Any]]
00079         Sortie:
00080             bool (validité)
00081         Rôle:
00082             Vérifie la capacité du plateau à être un tableau.
00083         """
```

```
00084         # Déclaration de valide et initialisation sur False
00085         valide = True
00086 
00087         # Si plateau est une liste
00088         if type(plateau) is list:
00089             # Déclaration d'un itérateur
00090             i = 0
00091             # Pour chaque valeur de plateau tant que valide vaut True
00092             while i < len(plateau) and valide:
00093                 # Si plateau à l'indice i n'est pas une liste
00094                 if type(plateau[i]) is not list:
00095                     # On passe valide à False
00096                     valide = False
00097                 # Incrémentation de l'itérateur
00098                 i += 1
00099         # Si plateau n'est pas une liste
00100         else:
00101             # On passe valide à False
00102             valide = False
00103 
00104         # Retourne la valeur de valide
00105         return valide
00106 
00107     def set_tableau(self, plateau: Any, vivant: Any = 1) -> bool:
00108         """
00109         Entrées:
00110             self: JeuDeLaVieCLI
00111             plateau: list[list[Any]] (attendu)
00112             vivant: Any
00113                 valeur par défaut: 1
00114         Sortie:
00115             bool (validité)
00116         Rôle:
00117             Redéfinit l'attribut tableau de même dimension que le matrice. Si
00118             l'élément vaut vivant, son état sera Vivant, sinon Mort.
00119         """
00120         # Si le plateau est apte à devenir tableau
00121         if self.est_tableau_valide(plateau):
00122             # On vide tableau
00123             self.tableautableautableautableau = []
00124             # Pour chaque indice de plateau
00125             for i in range(len(plateau)):
00126                 # On ajoute une ligne à tableau
00127                 self.tableautableautableautableau.append([])
00128                 # Pour chaque indice de plateau à l'indice i
00129                 for j in range(len(plateau[i])):
00130                     # Si l'élément (i;j) de plateau est vivant
00131                     if plateau[i][j] == vivant:
00132                         # On ajoute 1 à tableau (vivant)
00133                         self.tableautableautableautableau[i].append(1)
00134                     # Sinon
00135                     else:
00136                         # On ajoute 0 à tableau (mort)
00137                         self.tableautableautableautableau[i].append(0)
00138             # On retourne True car la procédure c'est bien déroulée
00139             return True
00140 
00141         # On retourne False car le plateau ne peut être utilisé comme tableau
00142         return False
00143 
00144     def __repr__(self) -> str:
00145         """
00146         Entrées:
00147             self: JeuDeLaVieCLI
00148         Sortie:
00149             str
00150         Rôle:
00151             Retourne une représentation en chaîne de caractères de manière
00152             simple (matrice de 1 et de 0).
00153         """
00154         reponse = ""
00155         # Pour chaque ligne du tableau
00156         for ligne in self.tableautableautableautableau:
00157             # Pour chaque cellule de la ligne
00158             for cellule in ligne:
00159                 # On ajoute la valeur de cellule en chaîne de caractères plus
00160                 # un espace
00161                 reponse += str(cellule) + " "
00162             # On ajoute un retour à la ligne
00163             reponse += "\n"
00164 
00165         # On retourne la chaîne de caractère
00166         return reponse
00167 
00168     def affiche(self) -> None:
00169         """
00170         Entrées:
```

```
00171             self: JeuDeLaVieCLI
00172         Sortie:
00173             None (affichage)
00174         Rôle:
00175             Afficher de manière complexe un tableau de cellules remplacées
00176             par des caractères.
00177         """
00178         # On efface le terminal
00179         system("cls")
00180         # crée la variable temporaire tab
00181         tab = self.tableautableautableautableau
00182         # pour chaque ligne du tableau
00183         for ligne in tab:
00184             # on crée une variable temporaire pour chaque ligne
00185             ligne_affiche = []
00186             for valeur in ligne:
00187                 # si la valeur dans le tableau est 0 on ajoute
00188                 # self.symbole_mort à la variable temporaire
00189                 if valeur == 0:
00190                     ligne_affiche.append(self.symbole_mort)
00191                 # si la valeur dans le tableau est 1 on ajoute
00192                 # self.symbole_vivant à la variable temporaire
00193                 else:
00194                     ligne_affiche.append(self.symbole_vivant)
00195             # On affiche la variable temporaire en enlevant les ""
00196             print(" ".join(ligne_affiche))
00197         # On affiche une ligne vide
00198         print()
00199 
00200     def valeur_case(self, i: int, j: int) -> Literal[0] | Literal[1]:
00201         """
00202         Entrées:
00203             self: JeuDeLaVieCLI
00204             i: int
00205             j: int
00206         Sortie:
00207             Literal[0] | Literal[1]
00208         Rôle:
00209             Donner l'état d'une case (1 ou 0).
00210         """
00211         # si les indices décrivent une valeur du tableau
00212         if 0 <= i < len(self.tableautableautableautableau) and 0 <= j <
      len(self.tableautableautableautableau[0]):
00213             # On retourne la valeur de la case (i;j)
00214             return self.tableautableautableautableau[i][j]
00215         # si l'indice i ou j est trop grand ou négatif
00216         else:
00217             # on retourne 0
00218             return 0
00219 
00220     def total_voisins(self, i: int, j: int) -> int:
00221         """
00222         Entrées:
00223             self: JeuDeLaVieCLI
00224             i: int
00225             j: int
00226         Sortie:
00227             int
00228         Rôle:
00229             Retourne le total de voisins de la cellule (i;j)
00230         """
00231         # Récupère le voisin du bas
00232         b = self.valeur_case(i + 1, j)
00233         # Récupère le voisin du bas droit
00234         bd = self.valeur_case(i + 1, j + 1)
00235         # Récupère le voisin du bas gauche
00236         bg = self.valeur_case(i + 1, j - 1)
00237         # Récupère le voisin du haut
00238         h = self.valeur_case(i - 1, j)
00239         # Récupère le voisin du haut droit
00240         hd = self.valeur_case(i - 1, j + 1)
00241         # Récupère le voisin du haut gauche
00242         hg = self.valeur_case(i - 1, j - 1)
00243         # Récupère le voisin de droite
00244         d = self.valeur_case(i, j + 1)
00245         # Récupère le voisin de gauche
00246         g = self.valeur_case(i, j - 1)
00247 
00248         # retourne la somme des voisins
00249         return b + bd + bg + h + hd + hg + d + g
00250 
00251     @staticmethod
00252     def meurt(nb_voisins: int) -> bool:
00253         """
00254         Entrée:
00255             nb_voisins: int
00256         Sortie:
```

```
00257             bool
00258         Rôle:
00259             Retourne True si la cellule meurt
00260         """
00261         # Retourne True si le nombre de voisins est différent de 2 ou 3
00262         return nb_voisins < 2 or nb_voisins > 3
00263
00264     @staticmethod
00265     def nait(nb_voisins: int) -> bool:
00266         """
00267         Entrée:
00268             nb_voisins: int
00269         Sortie:
00270             bool
00271         Rôle:
00272             Retourne True si la cellule naît
00273         """
00274         # Retourne True si le nombre de voisins est égal à 3
00275         return nb_voisins == 3
00276
00277     def resultat(self, i: int, j: int) -> Literal[0] | Literal[1]:
00278         """
00279         Entrées:
00280             self: JeuDeLaVieCLI
00281             i: int (ordonnée)
00282             j: int (abscisse)
00283         Sortie:
00284             Literal[0] | Literal[1] (Etat de la cellule)
00285         Rôle:
00286             Retourne l'état de la cellule en fonction du nombre de voisins
00287         """
00288         # On calcul le nombre de voisin
00289         nb_voisins = self.total_voisins(i, j)
00290         # si la cellule == 1 (vivante)
00291         if self.tableautableautableautableau[i][j]:
00292             # si la cellule meurt
00293             if self.meurt(nb_voisins):
00294                 # On met à jour la valeur dans le faux tableau
00295                 return 0
00296             # Sinon
00297             else:
00298                 # On retourne la valeur de (i;j)
00299                 return 1
00300         # si la cellule == 0 (morte)
00301         else:
00302             # si la cellule naît
00303             if self.nait(nb_voisins):
00304                 # On met à jour la valeur dans le faux tableau
00305                 return 1
00306             # Sinon
00307             else:
00308                 # On retourne la valeur de (i;j)
00309                 return 0
00310
00311     def tour(self) -> None:
00312         """
00313         Entrée:
00314             self: JeuDeLaVieCLI
00315         Sortie:
00316             None (modification en place)
00317         Rôle:
00318             Execute un tour du jeu.
00319         """
00320         # On copie le tableau pour pouvoir geler le vrai pour faire les modifs
00321         self.tableau_precedenttableau_precedent = deepcopy(self.tableautableautableautableau)
00322         # On déclare tableau comme une copie de l'attribut tableau (gèle)
00323         tableau = deepcopy(self.tableautableautableautableau)
00324
00325         # on itère dans les lignes
00326         for i in range(len(self.tableautableautableautableau)):
00327             # on itère dans la ligne i
00328             for j in range(len(self.tableautableautableautableau[i])):
00329                 tableau[i][j] = self.resultat(i, j)
00330
00331         # On attribut le tableau gelé au vrai tableau
00332         self.tableautableautableautableau = tableau
00333
00334     def arret_automatique(self) -> bool:
00335         """
00336         Entrée:
00337             self: JeuDeLaVieCLI
00338         Sortie:
00339             bool
00340         Rôle:
00341             Renvoie True si le plateau est identique deux tours de suite, False
00342             sinon.
00343         """
```

```
00344         # retourne True si le tableau n'a pas changé entre deux cycles
00345         return self.tableau_precedenttableau_precedent == self.tableautableautableautableau
00346
00347     def run(self, nombre_tours: int, delai: float) -> None:
00348         """
00349         Entrées:
00350             self: JeuDeLaVieCLI
00351             nombre_tours: int
00352             delai: float
00353         Sortie:
00354             None (modification en place)
00355         Rôle:
00356             Effectue nombre_tours cycles de delai seconde(s).
00357         """
00358         # Si le tableau n'est pas vide
00359         if self.tableautableautableautableau:
00360             # On déclare un itérateur i
00361             i = 0
00362             # se répète en fonction du nombre de tour
00363             while i < nombre_tours:
00364                 # si 2 tours à la suite sont identique
00365                 if self.arret_automatique():
00366                     # on stop la boucle
00367                     i = nombre_tours - 1
00368                 # sinon la boucle réactualise eu prochain tour
00369                 else:
00370                     # affiche la matrice de JeuDeLaVieCLI
00371                     self.affiche()
00372                     # actualise la matrice
00373                     self.tour()
00374                     # laisse un temps d'attente entre chaque tour
00375                     sleep(delai)
00376                 # on incrémente n pour que la boucle ait une terminaison
00377                 i += 1
00378                 # Affiche le dernier tour de boucle
00379             self.affiche()
00380
00381     def importe_template(self, fichier: str) -> bool:
00382         """
00383         Entrées:
00384             self: JeuDeLaVieCLI
00385             fichier: str
00386         Sortie:
00387             bool (validité)
00388         Rôle:
00389             Importe le template si celui-ci est valide.
00390         """
00391         # Si le fichier existe
00392         if exists(fichier):
00393             # On ouvre le fichier en mode lecture
00394             with open(file=fichier, mode='r', encoding='utf8') as f:
00395                 # On extrait les données
00396                 template = f.readlines()
00397                 # On ferme le fichier
00398                 f.close()
00399             # Si le template est valide
00400             valide, erreur = est_template_valide(template)
00401             if valide:
00402                 # Attribut à tableau une matrice vide
00403                 self.tableautableautableautableau = []
00404                 # Pour chaque ligne
00405                 for i, ligne in enumerate(reader(template)):
00406                     # On ajoute une nouvelle ligne à la matrice
00407                     self.tableautableautableautableau.append([])
00408                     # Pour chaque élément de la ligne
00409                     for element in ligne:
00410                         # On ajoute l'élément converti en int
00411                         self.tableautableautableautableau[i].append(int(element))
00412                 # On retoure True
00413                 return True
00414             else:
00415                 print(erreur)
00416         # On retourne False
00417         return False
00418
00419
00420 # Si le présent fichier est executé avec python 3.10 ou plus
00421 if __name__ == "__main__" and version_info >= (3, 10):
00422
00423     # Exemple:
00424
00425     # Déclare jeu en tant qu'instance de JeuDeLaVieCLI
00426     jeu = JeuDeLaVieCLI()
00427
00428     # importe le template choisit dans le jeu
00429     jeu.importe_template(r"..\\templates\\glider gun.csv")
00430
```

```
00431     # lance le jeu de la vie pour 75 cycles de 0.2 seconde
00432     jeu.run(75, 0.2)
```

7.21 Référence du fichier D:/Jeu de la vie/source/jeu_de_la_vie_gui.py

Classes

— class jeu_de_la_vie_gui.JeuDeLaVieGUI

Espaces de nommage

— namespace jeu_de_la_vie_gui

Variables

— jeu_de_la_vie_gui.reponse

— jeu_de_la_vie_gui.proc = run("pip install PySide6", stdout=PIPE)

— jeu_de_la_vie_gui.app = QApplication(argv)

— jeu_de_la_vie_gui.my_window = JeuDeLaVieGUI()

7.22 jeu_de_la_vie_gui.py

Aller à la documentation de ce fichier.

```
00001 # importe util depuis importlib
00002 from importlib import util
00003
00004 # Si le module PySide6 n'est pas installé
00005 if not util.find_spec("PySide6"):
00006     # Questionne l'utilisateur sur sa volonté d'installer PySide6
00007     reponse = input("Le module PySide6 n'est pas installé, souhaitez-vous " +
00008                     "l'installer automatiquement ? (o/n)")[0]
00009     # Si la réponse est o (pour oui)
00010     if reponse.lower() == "o":
00011         # import run et PIPE du module subprocess
00012         from subprocess import run, PIPE
00013         # Affichage d'un message pour que l'utilisateur patiente
00014         print("Intallation de PySide6, cela peut prendre quelques minutes. " +
00015             "Merci de ne pas intérompre le processus.")
00016         # Installation de PySide6
00017         proc = run("pip install PySide6", stdout=PIPE)
00018         # Affiche le flux de sortie du terminal
00019         print(proc.stdout.decode())
00020         # Si le code de sortie n'est pas 0
00021         if proc.returncode:
00022             # Affichage d'un message d'erreur
00023             print("Une erreur est intervenue, merci de la corriger avant de " +
00024                   "relancer le programme")
00025             # Termine le processus avec proc.returncode en code de sortie
00026             exit(proc.returncode)
00027     # Si le message d'erreur est n (pour n) ou autre
00028     else:
00029         # Affiche un message à l'utilisateur
00030         print("Le programme ne peut démarrer tant que PySide6 n'est pas " +
00031               "installé")
00032         # Termine le processus avec 1 en code de sortie
00033         exit(1)
00034
00035 # Importe les classes utilisées depuis PySide6.QtWidgets
00036 from PySide6.QtWidgets import QMainWindow, QApplication, QGraphicsView, \
00037     QHBoxLayout, QVBoxLayout, QPushButton, QWidget, QDoubleSpinBox, QLabel, \
```

```
00038     QGridLayout, QCheckBox, QFileDialog, QMessageBox, QFrame, QSizePolicy, \
00039     QMenuBar, QMenu
00040 # Importe les classes utilisées depuis PySide6.QtGui
00041 from PySide6.QtGui import QIcon, QAction, QKeySequence, QStatusTipEvent
00042 # Importe les classes utilisées depuis PySide6.QtCore
00043 from PySide6.QtCore import Qt, QSize,  QDir, QEvent
00044 # Importe reader depuis le module csv
00045 from csv import reader
00046 # Importe argv, version_info depuis le module sys
00047 from sys import argv, version_info
00048 # Importe est_template_valide, Scene du module plateau de dependances.jdlv
00049 from dependances.jdlv import est_template_valide, Scene
00050 # Importe basename depuis le module os.path
00051 from os.path import basename
00052
00053
00054 class JeuDeLaVieGUI(QMainWindow):
00055     """
00056     Hérite de:
00057         QMainWindow
00058     Rôle:
00059         Représente l'application du jeu de la vie
00060     """
00061     def __init__(self) -> None:
00062         """
00063         Entrée:
00064             self: JeuDeLaVieGUI
00065         Sortie:
00066             None (ctor)
00067         Rôle:
00068             Construit un nouvel objet JeuDeLaVieGUI.
00069         """
00070         # Initialisation de la classe mère
00071         QMainWindow.__init__(self)
00072
00073         # Attributs
00074
00075         # Déclaration de chemin_absolu pour accéder aux ressources (svg, ...)
00076         self.chemin_absolu: str = __file__[:-len(basename(__file__))]
00077         # Déclaration d'un booléan est_fichier_ouvert
00078         self.est_fichier_ouvert: bool = False
00079         self.fichier: str = ""
00080         # Déclaration d'un attribut nb_cycle
00081         self.nb_cycle: int = 0
00082
00083         # Fenêtre
00084
00085         # Définit une taille minimum pour la fenêtre
00086         self.setMinimumSize(710, 400)
00087         # Redimensionne la fenêtre
00088         self.resize(self.screen().geometry().width() - 400,
00089                     self.screen().geometry().height() - 300)
00090         # Replace la fenêtre
00091         self.move(
00092             (self.screen().geometry().width() - self.width()) // 2,
00093             (self.screen().geometry().height() - self.height()) // 2 - 50)
00094         # On créer widget_central
00095         self.widget_central = QWidget(self)
00096         # On définit widget_central comme le widget principal
00097         # (celui qui prend toute la place)
00098         self.setCentralWidget(self.widget_central)
00099         # On créer le layout principal
00100         self.affichage = QHBoxLayout()
00101         # On associe le layout au widget car setCentralLayout n'existe pas
00102         self.widget_central.setLayout(self.affichage)
00103
00104         # Menu bar
00105
00106         # Déclaration d'un attribut menuBar
00107         self.menu: QMenuBar = self.menuBar()
00108         # Déclaration d'un attribut menu fichier
00109         self.menu_fichier: QMenu = self.menu.addMenu("&Fichier")
00110         # Déclaration de importer_template
00111         self.importer_template = QAction(self)
00112         # Définition du texte de importer_template
00113         self.importer_template.setText("Importer un template (.csv)")
00114         # Définition d'un raccourci clavier
00115         self.importer_template.setShortcut(QKeySequence.StandardKey.Open)
00116         # Relie le signal à la méthode importe
00117         self.importer_template.triggered.connect(self.importe_templateimporte_template)
00118         # Ajoute l'action au menu
00119         self.menu_fichier.addAction(self.importer_template)
00120
00121         # Déclaration de enregistrer_template
00122         self.enregistrer_template: QAction = QAction(self)
00123         # Définition du texte de enregistrer_template
00124         self.enregistrer_template.setText("Enregistrer le template")
```

```
00125         # Définition d'un raccourci clavier
00126         self.enregistrer_template.setShortcut(QKeySequence.StandardKey.Save)
00127         # Relie le signal à la méthode enregistrer
00128         self.enregistrer_template.triggered.connect(self.enregistrerenregistrer)
00129         # Ajoute l'action au menu
00130         self.menu_fichier.addAction(self.enregistrer_template)
00131 
00132         # Déclaration de enregistrer_template_sous
00133         self.enregistrer_template_sous = QAction(self)
00134         # Définition du texte de enregistrer_template
00135         self.enregistrer_template_sous.setText("Enregistrer sous le template")
00136         # Définition d'un raccourci clavier
00137         self.enregistrer_template_sous.setShortcut(
00138             QKeySequence.StandardKey.SaveAs)
00139         # Relie le signal à la méthode enregistrer
00140         self.enregistrer_template_sous.triggered.connect(self.enregistrer_sousenregistrer_sous)
00141         # Ajoute l'action au menu
00142         self.menu_fichier.addAction(self.enregistrer_template_sous)
00143 
00144         # Scène - vue
00145 
00146         # On créer une scène du jeu de la vie
00147         self.scene = Scene(self, QSize(10, 10))
00148         # On créer une vue
00149         self.vue = QGraphicsView(self)
00150         # Définition d'un conseil pour l'utilisateur
00151         self.vue.setStatusTip(
00152             "Scène du jeu de la vie. Vous pouvez zommer, vous y déplacer et " +
00153             "modifier l'état des cellules en cliquant dessus")
00154         self.vue.enterEvent
00155         # On dit à la vue quelle scène afficher
00156         # Fonctionnement -> https://doc.qt.io/qt-6/graphicsview.html
00157         self.vue.setScene(self.scene)
00158         # On définit l'échelle (fonctionne par pourcentage d'agrandissement)
00159         self.vue.scale(0.5, 0.5)
00160         # On ajoute la vue au layout principal
00161         self.affichage.addWidget(self.vue)
00162 
00163         # Bar d'outils
00164 
00165         # Déclaration d'un layout vertical outils_layout, layout ≃ div en html
00166         self.outils_layout = QVBoxLayout()
00167         # On définit les options d'alignement du layout
00168         self.outils_layout.setAlignment(
00169             Qt.AlignmentFlag.AlignHCenter | Qt.AlignmentFlag.AlignTop)
00170         # On ajoute outils_layout au layout principal
00171         self.affichage.addLayout(self.outils_layout)
00172 
00173         # Déclaration d'un label
00174         self.affichage_cycle = QLabel(self)
00175         # Définition d'un conseil pour l'utilisateur
00176         self.affichage_cycle.setStatusTip("Compte le nombre de cycle effectué")
00177         # Modifie le texe du label
00178         self.affichage_cycle.setText("Cycle n°" + str(self.nb_cycle))
00179         # Définit l'alignement du texte
00180         self.affichage_cycle.setAlignment(Qt.AlignmentFlag.AlignRight)
00181         # Ajoute le label dans le layout
00182         self.outils_layout.addWidget(self.affichage_cycle)
00183         # On ajoute un peu d'espace entre le haut de la fenêtre et les boutons
00184         self.outils_layout.addSpacing(20)
00185 
00186         # Déclaration d'un label
00187         self.controles_titre = QLabel(self)
00188         # Définition d'un conseil pour l'utilisateur
00189         self.controles_titre.setStatusTip("Contrôle du jeu de l'animation.")
00190         # Définitons du texte du label
00191         self.controles_titre.setText("Contrôles :")
00192         # Définition du style du texte et de sa taille
00193         self.controles_titre.setStyleSheet(
00194             "font-style: bold; font-size: 11pt;")
00195         # Ajoute le titre à outils_layout
00196         self.outils_layout.addWidget(self.controles_titre)
00197         # Déclaration d'un layout horizontal jouer/pause
00198         self.controles_layout = QHBoxLayout()
00199         # Déclaration d'un bouton jouer
00200         self.jouer = QPushButton(self)
00201         # Définition d'un conseil pour l'utilisateur
00202         self.jouer.setStatusTip("Joue l'animation.")
00203         # Définition de l'icone de jouer
00204         icone_jouer = QIcon(self.chemin_absolu +
00205                             "assets\\icones\\play-button.svg")
00206         # Attribution de l'icone au bouton
00207         self.jouer.setIcon(icone_jouer)
00208         # Relie le signal à la méthode run
00209         self.jouer.clicked.connect(self.lance_animlance_anim)
00210         # Modification du style de jouer
00211         self.jouer.setStyleSheet("padding-top: 7px; padding-bottom: 7px")
```

```
00212         # Modification automatique de la taille
00213         self.jouer.adjustSize()
00214         # On ajoute jouer à controles_layout
00215         self.controles_layout.addWidget(self.jouer)
00216         # Déclaration d'un bouton pause
00217         self.pause = QPushButton(self)
00218         # Définition d'un conseil pour l'utilisateur
00219         self.pause.setStatusTip("Pause l'animation")
00220         # Définition de l'icone de pause
00221         icone_pause = QIcon(self.chemin_absolu +
00222                             "assets\\icones\\pause-button.svg")
00223         # Attribution de l'icone au bouton
00224         self.pause.setIcon(icone_pause)
00225         # Relie le signal à la méthode stop
00226         self.pause.clicked.connect(self.stop_animstop_anim)
00227         # Désactive le bouton pause
00228         self.pause.setEnabled(False)
00229         # Modification du style de pause
00230         self.pause.setStyleSheet("padding-top: 7px; padding-bottom: 7px")
00231         # Modification automatique de la taille
00232         self.pause.adjustSize()
00233         # On ajoute pause à controles_layout
00234         self.controles_layout.addWidget(self.pause)
00235         # On ajoute controles_layout à outils_layout
00236         self.outils_layout.addLayout(self.controles_layout)
00237
00238         # Déclaration du bouton tour par tour
00239         self.tour_par_tour = QPushButton(self)
00240         # Définition d'un conseil pour l'utilisateur
00241         self.tour_par_tour.setStatusTip("Exécute un tour de l'animation")
00242         # On définit le texte à afficher
00243         self.tour_par_tour.setText("Tour par tour")
00244         # Relie le signal à la méthode tour de la scène
00245         self.tour_par_tour.clicked.connect(self.scene.tour)
00246         # On ajoute tour_par_tour au layout du menu
00247         self.outils_layout.addWidget(self.tour_par_tour)
00248         # Ajoute de l'espace après
00249
00250         # Déclaration de periode_layout
00251         self.periode_layout = QGridLayout()
00252         # Déclaration de periode_label
00253         self.periode_label = QLabel(self)
00254         # Définition d'un conseil pour l'utilisateur
00255         self.periode_label.setStatusTip(
00256             "La période est la durée d'un cycle de l'animation.")
00257         # Définition du texte de periode_label
00258         self.periode_label.setText("Durée d'une période :")
00259         # Déclaration de periode_entree
00260         self.periode_entree = QDoubleSpinBox(self)
00261         # Définition d'un conseil pour l'utilisateur
00262         self.periode_entree.setStatusTip(
00263             "Une période trop courte peut causer des ralentissements.")
00264         # On définit la valeur minimal de periode_entree
00265         self.periode_entree.setMinimum(0.01)
00266         # Définition de la valeur de periode_entree
00267         self.periode_entree.setValue(0.5)
00268         # Définition du pas de periode_entree
00269         self.periode_entree.setSingleStep(0.1)
00270         # On définit le suffix de periode_entree
00271         self.periode_entree.setSuffix(" sec")
00272         # On définit la largeur minimal de periode_entree
00273         self.periode_entree.setMinimumWidth(120)
00274         # Relie le signal à la méthode set_periode
00275         self.periode_entree.valueChanged.connect(self.set_periodeset_periode)
00276         # Ajoute periode_label dans la cellule (0;0) du layout
00277         self.periode_layout.addWidget(self.periode_label, 0, 0)
00278         # Ajoute periode_entree dans la cellule (0;1) du layout
00279         self.periode_layout.addWidget(self.periode_entree, 0, 1)
00280         # Ajoute periode_layout à outils_layout
00281         self.outils_layout.addLayout(self.periode_layout)
00282         # Ajoute de l'espace après
00283         self.outils_layout.addSpacing(7)
00284         # Déclaration d'une ligne de séparation
00285         self.ligne_post_controles = QFrame(self)
00286         # Définition de la forme
00287         self.ligne_post_controles.setFrameShape(QFrame.Shape.HLine)
00288         # Définiton des ombres
00289         self.ligne_post_controles.setFrameShadow(QFrame.Shadow.Sunken)
00290         # Ajoute la ligne à outils_layout
00291         self.outils_layout.addWidget(self.ligne_post_controles)
00292         # Ajoute de l'espace après
00293         self.outils_layout.addSpacing(7)
00294
00295         # Déclaration d'une case à cocher
00296         self.auto_grandissement_entree = QCheckBox(self)
00297         # Définition d'un conseil pour l'utilisateur
00298         self.auto_grandissement_entree.setStatusTip("Quand est cochée, " +
```

```
00299             "agrandit le plateau pour s'adapter à l'animation")
00300         # Définition du texte de la case à cocher
00301         self.auto_grandissement_entree.setText("Auto grandissement")
00302         # On définit son état sur coché
00303         self.auto_grandissement_entree.setCheckState(Qt.CheckState.Checked)
00304         # Relie le signal à la fonction set_auto_grandissement
00305         self.auto_grandissement_entree.checkStateChanged.connect(
00306             self.set_auto_grandissementset_auto_grandissement)
00307         # On ajoute la case à cocher au layout
00308         self.outils_layout.addWidget(self.auto_grandissement_entree)
00309
00310         # Déclaration d'une case à cocher
00311         self.auto_stop_entree = QCheckBox(self)
00312         # Définition d'un conseil pour l'utilisateur
00313         self.auto_stop_entree.setStatusTip(
00314             "Quand est cochée, arrête automatiquement l'animation quand il " +
00315             "n'y a plus de changement.")
00316         # Définition du texte de la case à cocher
00317         self.auto_stop_entree.setText("Auto stop")
00318         # On définit son état sur coché
00319         self.auto_stop_entree.setCheckState(Qt.CheckState.Checked)
00320         # Relie le signal à la fonction set_auto_stop
00321         self.auto_stop_entree.checkStateChanged.connect(
00322             self.set_auto_stopset_auto_stop)
00323         # On ajoute la case à cocher au layout
00324         self.outils_layout.addWidget(self.auto_stop_entree)
00325         # Ajoute de l'espace après
00326         self.outils_layout.addSpacing(7)
00327         # Déclaration d'une ligne de séparation
00328         self.ligne_post_autos = QFrame(self)
00329         # Définition de la forme
00330         self.ligne_post_autos.setFrameShape(QFrame.Shape.HLine)
00331         # Définiton des ombres
00332         self.ligne_post_autos.setFrameShadow(QFrame.Shadow.Sunken)
00333         # Ajoute la ligne à outils_layout
00334         self.outils_layout.addWidget(self.ligne_post_autos)
00335         # Ajoute de l'espace après
00336         self.outils_layout.addSpacing(7)
00337
00338         # Déclaration d'un layout horizontal
00339         self.zoom = QGridLayout()
00340         # Déclaration d'un label zoom
00341         self.zoom_label = QLabel(self)
00342         # Définitions du text du label
00343         self.zoom_label.setText("Zoom :")
00344         # Définition du style du texte et de sa taille
00345         self.zoom_label.setStyleSheet("font-style: bold; font-size: 11pt;")
00346         # Définition d'un conseil pour l'utilisateur
00347         self.zoom_label.setStatusTip("Interface de zoom")
00348         # On ajoute zoom_label dans la cellule (0;0) sur un espace de 1 ligne
00349         # et 2 colonnes
00350         self.zoom.addWidget(self.zoom_label, 0, 0, 1, 2)
00351         # Déclaration d'un bouton zoom_in_entree (zoom in)
00352         self.zoom_in_entree = QPushButton(self)
00353         # Définition d'un conseil pour l'utilisateur
00354         self.zoom_in_entree.setStatusTip("Zoom dans la scène")
00355         # Définition du texte du bouton
00356         self.zoom_in_entree.setText("+")
00357         # Relie le signal à la méthode zoom_in
00358         self.zoom_in_entree.clicked.connect(self.zoom_inzoom_in)
00359         # On ajoute zoom_in_entree au layout zoom
00360         self.zoom.addWidget(self.zoom_in_entree, 1, 0)
00361         # Déclaration d'un bouton zoom_out_entree (zoom out)
00362         self.zoom_out_entree = QPushButton(self)
00363         # Définition d'un conseil pour l'utilisateur
00364         self.zoom_out_entree.setStatusTip("Dézoom dans la scène")
00365         # Définition du texte du bouton
00366         self.zoom_out_entree.setText("-")
00367         # Relie le signal à la méthode zoom_out
00368         self.zoom_out_entree.clicked.connect(self.zoom_outzoom_out)
00369         # On ajoute zoom_out_entree au layout zoom
00370         self.zoom.addWidget(self.zoom_out_entree, 1, 1)
00371         # On ajoute zoom au layout du menu
00372         self.outils_layout.addLayout(self.zoom)
00373         # On ajoute de l'étirement qui à pour effet de prendre le plus de place
00374         # possible, ainsi, les prochains éléments seront le plus bas possible
00375         self.outils_layout.addStretch()
00376
00377         # Déclaration d'un titre pour bar_info
00378         self.bar_info_titre = QLabel(self)
00379         # Définition du texte
00380         self.bar_info_titre.setText("Vue Info :")
00381         # Définition du style du texte et de sa taille
00382         self.bar_info_titre.setStyleSheet("font-style: bold; font-size: 11pt;")
00383         # Ajoute le titre à outils_layout
00384         self.outils_layout.addWidget(self.bar_info_titre)
00385         # Déclaration de bar_info
```

```
00386         self.bar_info = QLabel(self)
00387         # Définit la taille minimum de bar_info en hauteur
00388         self.bar_info.setMinimumHeight(175)
00389         # On précise les marges du texte
00390         self.bar_info.setMargin(10)
00391         # Déclaration d'une Politique de taille et d'agrandissement
00392         bar_info_policy = QSizePolicy()
00393         # Spécifit que les demandes agrandissement sont ignorées
00394         # demandes effectuées à cause des marges
00395         bar_info_policy.setHorizontalPolicy(QSizePolicy.Policy.Ignored)
00396         # Attribution de la politique de taille à bar_info
00397         self.bar_info.setSizePolicy(bar_info_policy)
00398         # Définiton de l'attribut wordwrap sur True, pour les retous à la ligne
00399         self.bar_info.setWordWrap(True)
00400         # Précision de l'alignement du texte en haut à gauche
00401         self.bar_info.setAlignment(Qt.AlignmentFlag.AlignLeft |
00402                                    Qt.AlignmentFlag.AlignTop)
00403         # Définition de la taille du texte
00404         self.bar_info.setStyleSheet("font-size: 10.5pt;")
00405         # Définition du format du texte comme étant Markdown
00406         self.bar_info.setTextFormat(Qt.TextFormat.MarkdownText)
00407         # Sélection du style du cadre (classe mère que QLabel)
00408         self.bar_info.setFrameStyle(QFrame.Shape.StyledPanel)
00409         # Définition du statusTip (Cf méthode event)
00410         self.bar_info.setStatusTip(
00411             "La *Vue Info* fournit une brève description de l'élément " +
00412             "de l'interface utilisateur sur lequel la souris se trouve " +
00413             "actuellement.")
00414         # Ajoute bar_info à outils_layout
00415         self.outils_layout.addWidget(self.bar_info)
00416 
00417         # Initialisation de toutes les variables reliées aux champs d'entrées
00418 
00419         # Initialise auto_grandissement sur l'état de la case à cocher
00420         self.scene.auto_grandissement = bool(
00421             self.auto_grandissement_entree.checkState().value)
00422         # Initialise auto_stop sur l'état de la case à cocher
00423         self.scene.auto_stop = bool(self.auto_stop_entree.checkState().value)
00424         # Initialise periode sur la valeur de periode_entree
00425         self.scene.periode = int(self.periode_entree.value() * 1000)
00426 
00427         # Affiche la fenêtre
00428         self.show()
00429 
00430     def event(self, even: QEvent) -> bool:
00431         """
00432         Réimplémentation de event hérité de QMainWindow
00433         Entrées:
00434             self: JeuDeLaVieGUI
00435             even: QEvent (et les classes qui en hérite)
00436         Sortie:
00437             bool
00438         Rôle:
00439             Capture les événements de QMainWindow, les traites, puis les rends.
00440         """
00441         # Si le type de l'évènement est StatusTipEvent
00442         if even.type() is QEvent.Type.StatusTip:
00443             # On précise le type de l'évènement pour être sûr de pouvoir
00444             # accéder aux méthodes (optionnel)
00445             even: QStatusTipEvent = even
00446             # Définit le texte de bar_info sur le conseil de l'évènement
00447             self.bar_info.setText(even.tip())
00448             """
00449             Fonctionnement:
00450             Chaque objet possédant l'attribut statusTip émet un signal
00451             QStatusTipEvent lorsque la souris le survole. Ce signal comporte un
00452             tip, un conseil, concernant le dit objet. Il est par défaut
00453             réceptionné par statusBar de QMainWindow, mais il peut être
00454             intercepté et affiché autre part.
00455             """
00456 
00457         # On rend l'évènement
00458         return super().event(even)
00459 
00460     def enregistrer(self) -> None:
00461         """
00462         Entrée:
00463             self: JeuDeLaVieGUI
00464         Sortie:
00465             None (modification en place)
00466         Rôle:
00467             Enregistre le plateau au format csv.
00468         """
00469         # Si un fichier est "ouvert"
00470         if self.est_fichier_ouvert:
00471 
00472             # On ouvre le fichier (s'il n'existe plus, il est recréé)
```

```
00473             with open(self.fichier, 'w', encoding='utf-8') as f:
00474                 # Récupère le plateau avec 1 pour vivant et 0 pour mort
00475                 plateau: list[list[str]] = self.scene.get_tableau("1", "0")
00476                 # Pour chaque ligne
00477                 for i in range(len(plateau)):
00478                     # Pour chaque élément
00479                     for j in range(len(plateau[i])):
00480                         # On écrit l'élément
00481                         f.write(plateau[i][j])
00482                         # Si l'élément n'est pas le dernier
00483                         if j < len(plateau[i]) - 1:
00484                             # on écrit une virgule
00485                             f.write(",")
00486                     # Ecriture d'un retour chariot
00487                     f.write("\n")
00488                 # Fermeture du fichier
00489                 f.close()
00490             # On prévient l'utilisateur que le fichier a été enregistré
00491             self.event(QStatusTipEvent("Le fichier a été enregistré !"))
00492         # S'il n'y a pas de fichier ouvert
00493         else:
00494             # Appelle la méthode enregistrer_sous
00495             self.enregistrer_sousenregistrer_sous()
00496 
00497     def enregistrer_sous(self) -> None:
00498         """
00499         Entrée:
00500             self: JeuDeLaVieGUI
00501         Sortie:
00502             None (modification en place)
00503         Rôle:
00504             Enregistre sous le plateau au format csv.
00505         """
00506         # Déclaration d'un nouveau QFileDialog
00507         enregistrer_fichier = QFileDialog(self, caption="Enregistrer sous")
00508         # Définition du type de fichier attendu
00509         enregistrer_fichier.setNameFilter("Templates (*.csv)")
00510         # Définition du filtre
00511         enregistrer_fichier.setFilter(
00512             QDir.Filter.Readable | QDir.Filter.Files | QDir.Filter.NoSymLinks)
00513         # Définition du mode du fichier sur fichier existant
00514         enregistrer_fichier.setFileMode(QFileDialog.FileMode.ExistingFile)
00515         # Définition du suffix par défaut
00516         enregistrer_fichier.setDefaultSuffix(".csv")
00517         # Définition du mode
00518         enregistrer_fichier.setAcceptMode(QFileDialog.AcceptMode.AcceptSave)
00519 
00520         # Si l'utilisateur sélectionne un fichier
00521         if enregistrer_fichier.exec():
00522             # On met à jour l'attribut est_fichier_ouvert
00523             self.est_fichier_ouvert = True
00524             # Mise à jour du nom du fichier
00525             self.fichier = enregistrer_fichier.selectedFiles()[0]
00526             # On enregistre le template
00527             self.enregistrerenregistrer()
00528 
00529     def importe_template(self) -> None:
00530         """
00531         Entrée:
00532             self: JeuDeLaVieGUI
00533         Sortie:
00534             None (modification en place)
00535         Rôle:
00536             Importe un template d'un fichier csv.
00537         """
00538         # Déclaration d'un nouveau QFileDialog
00539         obtenir_fichier = QFileDialog(self, caption="Importer un template")
00540         # Définition du type de fichier attendu
00541         obtenir_fichier.setNameFilter("Templates (*.csv)")
00542         # Définition du filtre
00543         obtenir_fichier.setFilter(
00544             QDir.Filter.Readable | QDir.Filter.Files | QDir.Filter.NoSymLinks)
00545         # Définition du mode du fichier sur fichier existant
00546         obtenir_fichier.setFileMode(QFileDialog.FileMode.ExistingFile)
00547         # Définition du suffix par défaut
00548         obtenir_fichier.setDefaultSuffix(".csv")
00549         # Définition du mode
00550         obtenir_fichier.setAcceptMode(QFileDialog.AcceptMode.AcceptOpen)
00551 
00552         # Si l'utilisateur sélectionne un fichier
00553         if obtenir_fichier.exec():
00554             # Récupère le chemin du fichier
00555             fichier = obtenir_fichier.selectedFiles()[0]
00556 
00557             # On ouvre le fichier en mode lecture
00558             with open(file=fichier, mode='r', encoding='utf8') as f:
00559                 # On extrait les données
```

```
00560                 donnees = f.readlines()
00561                 # On ferme le fichier
00562                 f.close()
00563 
00564             # On teste la validité des données
00565             tableau_valide, message_erreur = est_template_valide(donnees)
00566 
00567             # si le tableau est valide
00568             if tableau_valide:
00569                 # On arrêtre l'animation (economie de ressources)
00570                 self.stop_animstop_anim()
00571                 # Déclaration d'une matrice vide
00572                 tableau = []
00573                 # Pour chaque ligne
00574                 for i, ligne in enumerate(reader(donnees)):
00575                     # On ajoute une nouvelle ligne à la matrice
00576                     tableau.append([])
00577                     # Pour chaque élément de la ligne
00578                     for element in ligne:
00579                         # On ajoute l'élément converti en int
00580                         tableau[i].append(int(element))
00581                 # On définit le plateau selon la matrice, sachant que l'élément
00582                 # vivant est 1
00583                 self.scene.set_tableau(tableau, 1)
00584                 # Remet le nombre de cycle à 0
00585                 self.nb_cycle = 0
00586                 # Modifie le texe du label
00587                 self.affichage_cycle.setText("Cycle n°" + str(self.nb_cycle))
00588                 # On met à jour l'attribut est_fichier_ouvert
00589                 self.est_fichier_ouvert = True
00590                 # Mise à jour du nom du fichier
00591                 self.fichier = fichier
00592             # Si le tableau n'est pas valide
00593             else:
00594                 # Affichage du message d'erreur
00595                 QMessageBox.warning(
00596                     self, "Erreur", message_erreur,
00597                     QMessageBox.StandardButton.Ok,
00598                     QMessageBox.StandardButton.Ok)
00599 
00600     def set_auto_grandissement(self, etat: Qt.CheckState) -> None:
00601         """
00602         Entrées:
00603             self: JeuDeLaVieGUI
00604             etat: Qt.CheckState
00605         Sortie:
00606             None (modification en place)
00607         Rôle:
00608             Redéfinit la valeur de auto_grandissement selon etat
00609         """
00610         # | etat              | etat.value
00611         # | Unchecked         | 0
00612         # | Checked           | 2
00613         # bool(0) -> False, bool(2) -> True
00614         # Attribut une nouvelle valeur à auto_grandissement
00615         self.scene.auto_grandissement = bool(etat.value)
00616 
00617     def set_auto_stop(self, etat: Qt.CheckState) -> None:
00618         """
00619         Entrées:
00620             self: JeuDeLaVieGUI
00621             etat: Qt.CheckState
00622         Sortie:
00623             None (modification en place)
00624         Rôle:
00625             Redéfinit la valeur de auto_stop selon etat
00626         """
00627         # | etat              | etat.value
00628         # | Unchecked         | 0
00629         # | Checked           | 2
00630         # bool(0) -> False, bool(1) -> True
00631         # Attribut une nouvelle valeur à auto_stop
00632         self.scene.auto_stop = bool(etat.value)
00633 
00634     def set_periode(self, valeur: float) -> None:
00635         """
00636         Entrées:
00637             self: JeuDeLaVieGUI
00638             valeur: double
00639         Sortie:
00640             None (modification en place)
00641         Rôle:
00642             Redéfinit la période/durée d'un cycle de l'animation.
00643         """
00644         self.scene.periode = int(valeur * 1000)
00645 
00646     def lance_anim(self) -> None:
```

```
00647         """
00648         Entrées:
00649             self: JeuDeLaVieGUI
00650         Sortie:
00651             None (modification en place)
00652         Rôle:
00653             Lance l'animation
00654         """
00655         # désactive le bouton jouer
00656         self.jouer.setEnabled(False)
00657         # active le bouton pause
00658         self.pause.setEnabled(True)
00659         # On enlève focus de periode_entree, puisqu'en se désactivant, le jouer
00660         # lui donne le focus
00661         self.periode_entree.clearFocus()
00662 
00663         # Démarre le jeu
00664         self.scene.run()
00665 
00666     def stop_anim(self) -> None:
00667         """
00668         Entrées:
00669             self: JeuDeLaVieGUI
00670         Sortie:
00671             None (modification en place)
00672         Rôle:
00673             Pause l'animation
00674         """
00675         # Arrête le chrono
00676         self.scene.chrono.stop()
00677         # active le bouton jouer
00678         self.jouer.setEnabled(True)
00679         # désactive le bouton pause
00680         self.pause.setEnabled(False)
00681         # On enlève focus de periode_entree, puisqu'en se désactivant, le pause
00682         # lui donne le focus
00683         self.periode_entree.clearFocus()
00684 
00685     def zoom_in(self) -> None:
00686         """
00687         Entrées:
00688             self: JeuDeLaVieGUI
00689         Sortie:
00690             None (modification en place)
00691         Rôle:
00692             Zoom dans la vue
00693         Addendum:
00694             Le nom est en anglais car le terme n'existe pas en français.
00695         """
00696         # Augmente les grandeurs de 10%
00697         self.vue.scale(1.1, 1.1)
00698 
00699     def zoom_out(self) -> None:
00700         """
00701         Entrées:
00702             self: JeuDeLaVieGUI
00703         Sortie:
00704             None (modification en place)
00705         Rôle:
00706             Dézoom dans la vue
00707         Addendum:
00708             Le nom est en anglais car le terme n'existe pas en français.
00709         """
00710         # Diminue les grandeurs de 10%
00711         self.vue.scale(0.9, 0.9)
00712 
00713 
00714 # Si le présent fichier est executé avec python 3.10 ou plus
00715 if __name__ == "__main__" and version_info >= (3, 10):
00716     # On instantie QApplication en lui passe argv (héritage du c++)
00717     app = QApplication(argv)
00718     # On instantie JeuDeLaVieGUI
00719     my_window = JeuDeLaVieGUI()
00720 
00721     # On execute l'application
00722     app.exec()
```

Index