Compte-rendu individuel

Elie Ruggiero
(Avec Eloi Leclair)

Léonard de Vinci - Montaigu-Vendée
21 novembre 2024

Table des matières

Notations et Définitions

Référence à une notation

mot (not.) signifit que le mot est définit ou expliqué dans le présent chapitre.

Acronymes et abréviations

POO	–	Programmation orientée objet
GUI	–	Graphical user interface (Interface utilisateur graphique)
CLI	–	Command-Line Interface (Interface de ligne de commande)
CTA	–	Call to action

Définitions

Spécifications fonctionnelles Les spécifications fonctionnelles (Functional requirement) ont pour objectif de décrire précisément l'ensemble des fonctions d'un logiciel ou d'une application afin de déterminer le périmètre fonctionnel du projet. Une fonction est alors décrite comme un résumé (ou une spécification ou une déclaration) du comportement entre les entrées et les sorties.

Spécifications fonctionnelles générales Les spécifications fonctionnelles générales (SFG) sont la représentation fonctionnelle du besoin métier. Lobjectif premier des SFG est de mettre daccord léquipe de développement sur les principales fonctionnalités à implémenter dans une future solution digitale.

Spécifications fonctionnelles détaillées Les spécifications fonctionnelles détaillées (SFD) ont pour objectif de venir préciser les fonctionnalités définies dans les spécifications fonctionnelles générales. Les SFD décrivent donc en détail les fonctionnalités mais aussi les sous-fonctions de votre solution. Une fonction est alors décrite comme un résumé (ou une spécification ou une déclaration) du comportement entre les entrées, les sorties et son rôle dans la solution.

Call to action Call to action (appel à l'action) dans la conception de sites web et dans l'expérience utilisateur (UX) en particulier - est un terme utilisé pour les éléments d'une page web qui sollicitent une action de la part de l'utilisateur. La manifestation la plus courante de l'appel à l'action dans les interfaces web se présente sous la forme de boutons cliquables.

Scène (jeu vidéo/animation) Ensemble cohérent de lieux, d'éléments visuels, sonores et narratifs réunis, où des actions spécifiques se déroulent dans un cadre particulier. Elle constitue une unité de jeu caractérisée par un espace matériel et sonore délimité.

Vue/Caméra (jeu vidéo/animation) La perspective et/ou l'angle de caméra à travers lequel le spectateur observe la scène. Les vues peuvent être à la première personne, à la troisième personne, en vue de dessus, en perspective isométrique, etc. (Fig. 1)

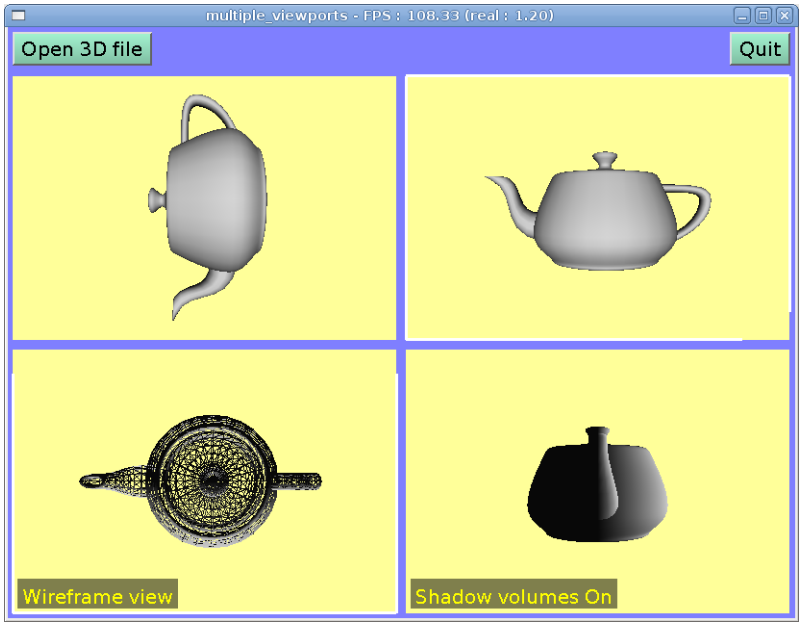

Figure 1 – Exemple de 4 vues sur une scène (source)

Chapitre 1

Rappel du Cahier des Charges

Rappelons d'abord les exigences minimales et consignes du projet.

1.1 Le projet

Développer une application du Jeu de la vie de Conway en utilisant la POO (not.).

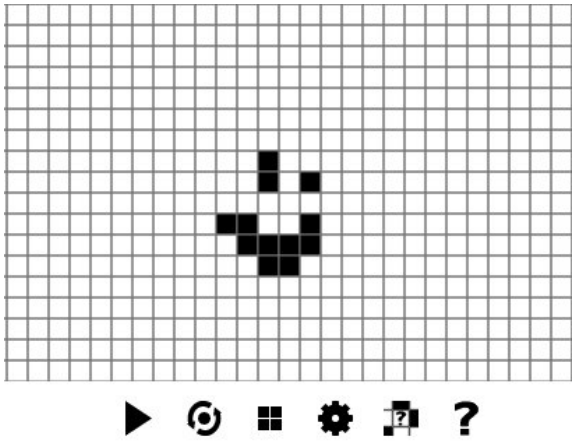

Figure 1.1 – Exemple d'un plateau du Jeu de la vie

1.2 Les objectifs

- Jouer au Jeu de la viedurant n cycles de t seconde ;
- Modifier le plateau de jeu.

1.3 Les responsabilités

Maîtrise d'ouvrage, maîtrise d'oeuvre : Elie Ruggiero et Eloi Leclair

1.4 Description de la solution

1.4.1 Les besoins fonctionnels

- Visualiser le plateau
- Lancer le jeu
- Modifier le plateau du jeu
- Modifier les paramètres du jeu [1]

1.4.2 Les besoins non fonctionnels

- optimisation
- documentation

1.5 Les contraintes

1.5.1 Qualité de la solution

- Maintenabilité
- Portabilité
- Fiabilité
- Facilité d'utilisation
- Performance
- Expérience utilisateur agréable

1.5.2 Temps de développement

Le projet doit être livré pour le 22 novembre 2024 au plus tard.

1.6 Les livrables attendus

- Le présent document
- Les fichiers sources
- La documentation
- Guide utilisateur (README.md)

1. Le nombre de cycles et leur durée, l'affichage etc.

Chapitre 2

Structuration du projet

Le projet s'est structuré comme suit

- Conception
  - Analyse du projet
  - Architecture de la solutions
- Développement
  - Répartiton
  - Organisation
  - Organisation temporel/planification
- Tests et corrections
- Rédaction des documents
- Retour d'expérience pour les projets futures

2.1 Organisation temporel/planification

Notre objectif était, en premier lieu, de terminer l'implémentation de la version CLI (not.) avant les vacances pour pouvoir s'attaquer à la version GUI pendant les vacances et la terminer avant la rentrée. Finir la version CLI avant les vacances pour pouvoir s'attaquer à la version GUI pendant les vacances.

Dans les faits, nous avons terminer la version CLI dans le courant de la première semaine des vacances, et la version GUI ne fut achevée que le 6 novembre 2024.

2.2 Moyens de communication

Nos échanges et retours sur nos codes respectifs s'est d'abord effectué en face à face. Le code était échangé par clé USB. Avec l'arrivé des vacances, nous échangions par Instagram et transmettions notre code via email. Après les vacances nous avons continuer d'échanger par instagram et nous repris les échanges en face à face. Par ailleurs, Eloi n'avais plus de tâches, j'ai donc publié le code sur un dépôt GitHub pour qu'il puisse suivre l'avancement sans m'obliger à lui transmettre régulièrement une copie du code source.

Chapitre 3

Conception du projet

3.1 Analyse des exigences : CLI (not.)

3.1.1 Collecte des exigences

Exigences[2] :

- Visualiser le plateau
- Lancer le jeu
- Modifier le plateau du jeu
- Modifier les paramètres du jeu[3]

3.1.2 Exigences fonctionnelles

Visualiser le plateau : `affiche()`
Lancer le jeu : `run()`
Modifier le plateau du jeu : `set_plateau()`, `importe_template()`
Modifier les paramètres du jeu : Attributs de la classe ou paramètre des méthodes

3.1.3 Spécifications fonctionnelles générales

Fonction		Entrée		Rôle
Type	Nom	Nom	Type	
None	affiche()	self	JeuDeLaVie	Affiche le plateau du jeu de la vie.
None	run()	self	JeuDeLaVie	Fait tourner le jeu de la vie nombre_tours cycles de delai seconde.
		nombre_tours	int	
		delai	float	
bool	set_plateau()	self	JeuDeLaVie	Redéfinit l'attribut tableau de même dimension que le matrice. Si l'élément vaut vivant, son état sera Vivant, sinon Mort.
		plateau	list[list[Any]]	
		vivant	Any	
bool	importe_template()	self	JeuDeLaVie	Importe le template si celui-ci est valide.
		fichier	str	

3.1.4 Division des responsablilités

On comptera ici le nombre de responsabilité en plus de celle de la fonction.

2. Besoins ou contraines.
3. Le nombre de cycles et leur durée par exemple.

run()

4 responsabilités :

- savoir s'il faut arrêter l'animation ;
- affichage ;
- execution d'un tour ;
- attendre le délai demandé.

Donc ajout de 4 fonctions :

`arret_automatique()` qui définit s'il faut arrêter l'execution du jeu en cas de boucle d'un cycle de période ;

`affiche()` qui affiche le plateau ;

`tour()` qui execute un tour ;

`sleep()` qui attend un temps donné.

tour()

2 responsabilités :

- copier le plateau ;
- avoir le nouvel état d'une cellule donnée.

Donc ajout de 2 fonctions :

`deepcopy()` qui copie en mémoire un objet (une matrice) ;

`resultat()` qui, pour une cellule donnée, définit l'état de cette cellule en fonction de ses voisines.

resultat()

3 responsabilités :

- compter le nombre de cellule voisine ;
- les règles de mort d'une cellule ;
- les règles de naissance d'une cellule ;

Donc ajout de 3 fonctions :

`total_voisins()` qui compte le nombre de cellule voisine ;

`meurt()` qui porte les règles de mort ;

`nait()` qui porte les règles de naissance.

total_voisins()

1 responsabilité :

- obtenir la valeur d'une cellule.

Donc ajout d'une fonction :

`valeur_case()` obtient la valeur d'une cellule.

set_tableau()

1 responsabilité :

- vérifier que le tableau proposé est valide.

Donc ajout d'une fonction :

`est_tableau_valide()`.

importe_template()

2 responsabilités :

- vérifier que le fichier existe ;
- vérifier que le template est valide.

Donc ajout de 2 fonctions :

`exists()` qui contrôle que le fichier existe ;

`est_template_valide()` qui contrôle que le template est valide.

affiche()

3 responsabilités :

- Exécuter des commandes terminales notamment pour effacer le terminal ;
- Définit le caractère représentant une cellule morte ;
- Définit le caractère représentant une cellule vivante.

Donc ajout de 3 fonctions :

`system()` qui exécute des commandes terminales (dont cls pour effacer le terminal) ;

`set_symbole_mort()` qui redéfinit le symbole d'une cellule morte ;

`set_symbole_vivant()` qui redéfinit le symbole d'une cellule vivante.

3.1.5 Application des consignes

Le tableau du barème spécifie que le programme doit contenir un « Affichage basique » et un « Affichage amélioré ». Ainsi, l'affichage basique se fait en utilisant la méthode magique `__repr__()` de Python [4] et l'affichage amélioré se fait via la méthode `affiche()`.

Par ailleurs, la section « Aides et conseils » nous propose d'utiliser pour copier le tableau la fonction `deepcopy()`, du module copy [5] de Python ainsi que la fonction `sleep()` du module time [6], ce qui explique le nom des fonctions dans les spécifications fonctionnelles.

3.1.6 Spécifications fonctionnelles détaillées

4. Documentation de la méthode : https ://docs.python.org/3/reference/datamodel.html
5. Documentation du module copy : https ://docs.python.org/3/library/copy.html
6. Documentation du module time : https ://docs.python.org/fr/3/library/time.html

Fonction		Entrée		Rôle
Type	**Nom**	**Nom**	**Type**	
None	set_symbole_mort()	self	JeuDeLaVie	Redéfinit le symbole des cases mortes (affichage complexe)
		symb_mort	Any	
None	set_symbole_vivant()	self	JeuDeLaVie	Redéfinit le symbole des cases vivantes (affichage complexe).
		symb_vivant	Any	
None	__repr__()	self	JeuDeLaVie	Retourne une représentation en chaîne de caractères de manière simple (matrice de 1 et de 0).
None	affiche()	self	JeuDeLaVie	Afficher de manière complexe un tableau de cellules remplacées par des caractères.
Literal[0] | Literal[1]	valeur_case()	self	JeuDeLaVie	Retroune la valeur d'une case (1 ou 0).
		i	int	
		j	int	
int	total_voisins()	self	JeuDeLaVie	Retourne le total de voisins de la cellule (i ;j).
		i	int	
		j	int	
bool	meurt()	nb_voisins	int	Retourne True si la cellule meurt, False sinon.
bool	nait()	nb_voisins	int	Retourne True si la cellule naît, False sinon.
Literal[0] | Literal[1]	resultat()	self	JeuDeLaVie	Retourne la valeur de la cellule (i ;j) en fonction du total de ses voisins.
		i	int	
		j	int	
None	tour()	self	JeuDeLaVie	Execute un tour du jeu.
bool	arret_automatique()	self	JeuDeLaVie	Renvoie True si le plateau est identique deux tours de suite.
None	run()	self	JeuDeLaVie	Effectue nombre_tours cycles de delai secondes.
		nombre_tours	int	
		delai	float	
bool	importe_template()	self	JeuDeLaVie	Importe le template si celui-ci est valide, la fonction renvoie alors True, False sinon.
		fichier	str	
Any	deepcopy()	matrice	list[list[Any]]	Retourne une copie mémoire de matrice
bool	est_template_valide()	tableau	Any	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.
None	sleep()	secs	float	Suspend l'exécution du thread appelant pendant le nombre de secondes indiqué.

Fonction		Entrée		Rôle
Type	**Nom**	**Nom**	**Type**	
bool	exists()	path	str	Retourne True si le chemin d'accès fait référence à un chemin d'accès existant, False sinon.
int	system()	command	str	Exécuter la commande dans un sous-shell.
bool	est_tableau_valide()	plateau	list[list[Any]]	Vérifie la capacité du plateau à être un tableau.

3.2 Analyse des exigences : GUI (not.)

3.2.1 Collecte des exigences

Exigences [7] :

- Visualiser le plateau
- Lancer le jeu
- Modifier le plateau du jeu
- Modifier les paramètres du jeu [8]

Exigences de l'expérience utilisateur (UX (not.)) :

- Se repérer dans l'interface
- Interagir avec le plateau
- Se déplacer dans le jeu
- Sauvegarder une plateau de jeu
- Obtenir de l'aide

3.2.2 Choix d'une librairie graphique

Tout au long de ce projet, nous utiliserons le framework Qt [9]. Il existe plusieurs versions de Qt pour Python, dont PySide [10] qui est maintenu par Qt Group et est donc « officiel ». Nous utiliserons donc PySide, dans sa version 6 (dernière version en date).

3.2.3 Exigences fonctionnelles

Visualiser le plateau : `QMainWindow` ;
Lancer le jeu : `QPushButton`, `lance_anim` et un séquenceur ;
Modifier le plateau du jeu : `importe_template()` et `set_plateau` ;
Modifier les paramètres du jeu : les CTA (not.) de `QtWidgets` .

Se repérer dans l'interface : `QLabel`, `QFrame`, `QLayout` ;
Interagir avec le plateau : `mousePressEvent()` ;
Se déplacer dans le jeu : `QGraphicsScene` et `QGraphicsView` (not.) avec `zoom_in()` et `zoom_out()` ;
Sauvegarder un plateau de jeu : `enregistrer()` et `enregistrer_sous()` ;
Obtenir de l'aide : `QLabel` et `QStatusTipEvent`.

7. Besoins ou contraines.
8. Le nombre de cycles et leur durée par exemple.
9. Site web : https ://www.qt.io/
10. Documentation : https ://doc.qt.io/qt.html

3.2.4 Spécifications fonctionnelles générales

Les fonctions

Fonction		Entrée		Rôle
Type	**Nom**	**Nom**	**Type**	
None	lance_anim()	self	JeuDeLaVie	Lance l'animation du jeu de la vie
bool	set_plateau()	self	JeuDeLaVie	Redéfinit l'attribut tableau de même dimension que le matrice. Si l'élément vaut vivant, son état sera Vivant, sinon Mort.
		plateau	list[list[Any]]	
		vivant	Any	
None	set_periode	self	JeuDeLaVie	Redéfinit la période/durée d'un cycle de l'animation.
		valeur	float	
bool	importe_template()	self	JeuDeLaVie	Importe le template si celui-ci est valide.
None	mousePressEvent()	self	JeuDeLaVie	Détecte les événements de pression de souris sur la cellule.
		even	QGraphicsSceneMouseEvent	
None	zoom_in()	self	JeuDeLaVie	Zoom dans la vue
None	zoom_out()	self	JeuDeLaVie	Dézoom dans la vue
None	enregistrer()	self	JeuDeLaVie	Enregistre le plateau au format csv.
None	enregistrer_sous()	self	JeuDeLaVie	Enregistre sous le plateau au format csv.
None	event()	self	JeuDeLaVie	Capture les événements de QMainWindow, les traites, puis les rends.
		even	QEvent	

Les widgets

Classe	Nom	Rôle
QMainWindow	JeuDeLaVieGUI	Représente l'application du jeu de la vie
QPushButton	jouer	Lance l'animation
	zoom_in_entree	Zoom dans la vue
	zoom_out_entree	Dézoom dans la vue
QLabel	controles_titre	Titre pour les contrôles
	periode_label	Titre pour la période
	zoom_label	Titre pour les boutons de zoom
	bar_info_titre	Titre pour la bar d'infos
	bar_info	Bar d'infos
QDoubleSpinBox	periode_entree	Moyen de saisie de la durée d'un cycle
QFrame	ligne_post_controles	Séparateur pour compartimenter les widgets
QHBoxLayout	affichage	Mise en page horizontale principale
	controles_layout	Mise en page horizontale pour les contrôles
QVBoxLayout	outils_layout	Mise en page verticale pour les outils (menu)
QGridLayout	periode_layout	Mise en page en grille pour la période
	zoom	Mise en page en grille pour le zoom
QGraphicsScene	scene	Représente la scène du jeu de la vie
QGraphicsView	vue	Regarde sur la scène du jeu de la vie

3.2.5 Division des responsablilités

On comptera ici le nombre de responsabilité en plus de celle de la fonction.

lance_anim()

1 responsabilités :

- Exéctuer les cycles du Jeu de la vie.

Donc ajout d'une fonction :

`run()` qui lance le chrono et exécute les cycles.

run()

5 responsabilités :

- execution d'un tour ;
- attendre le délai demandé.

Donc ajout de 5 fonctions :

`tour()` qui execute un tour ;

`chrono : QTimer` qui attend un temps donné.

tour()

7 responsabilités :

- garder l'état précédent ;
- avoir le nouvel état d'une cellule donnée ;
- construir un plateau fictif pour mettre à jour l'état des cellules ;
- redéfinir l'état d'une cellule ;
- savoir s'il faut agrandir le plateau ;
- extension du plateau.
- savoir s'il faut arrêter l'animation ;

Donc ajout de 7 fonctions :

`deepcopy()` qui copie en mémoire le plateau ;

`resultat()` qui, pour une cellule donnée, définit l'état de cette cellule en fonction de ses voisines ;

`construit()` qui construit une matrice de hauteur et largeur souhaitées avec l'élément par défaut souhaité ;

`set_etat()` qui redéfinit l'état d'une cellule ;

`doit_agrandir_tableau()` qui définit s'il faut agrandir le plateau, et dans quelle(s) direction(s), en cas de cellule(s) près de la frontière ;

`extension()` qui agrandit le plateau dans une direction donnée ;

`arret_automatique()` qui définit s'il faut arrêter l'exécution du jeu en cas de boucle d'un cycle de période ;

resultat()

3 responsabilités :

- compter le nombre de cellule voisine ;
- les règles de mort d'une cellule ;
- les règles de naissance d'une cellule ;

Donc ajout de 3 fonctions :

`total_voisins()` qui compte le nombre de cellule voisine ;

`meurt()` qui porte les règles de mort ;

`nait()` qui porte les règles de naissance.

total_voisins()

1 responsabilité :

- obtenir la valeur d'une cellule.

Donc ajout d'une fonction :

`valeur_case()` qui obtient la valeur d'une cellule.

set_etat()

1 responsabilité :

- redéfinir l'apparence d'une cellule.

Donc ajout d'une fonction :

`peint()` qui redéfinit l'apparence d'une cellule.

arret_automatique()

1 responsabilité :

- savoir si deux cellules ont le même état.

Donc ajout d'une fonction :

`__eq__()` qui permet de comparer deux cellules sur leur égalité commune.

doit_agrandir_tableau()

2 responsabilités :

- vider la scène ;
- obtenir l'état d'une cellule.

Donc ajout de 2 fonctions :

`vide_scene()` qui vide la scène et l'attribut tableau ;

`get_etat()` qui retourne l'état d'une cellule.

set_tableau()

1 responsabilité :

- redéfinir l'état d'une cellule.

Donc ajout d'une fonction :

`set_etat()` qui redéfinit l'état d'une cellule.

importe_template()

4 responsabilités :

- faire choisir le fichier ;
- vérifier que le template est valide ;
- lire un fichier csv ;
- redéfinir le plateau de jeu.

Donc ajout de 4 fonctions :

`QFileDialog` ouvre l'explorateur de fichier pour en sélectionner un ;

`est_template_valide()` qui contrôle que le template est valide.

`reader()` qui lit le fichier csv et retourne une liste ;

`set_tableau()` qui redéfinit le plateau de jeu.

mousePressEvent()

1 responsabilité :

- redéfinir l'apparence d'une cellule.

Donc ajout d'une fonction :

`peint()` qui redéfinit l'apparence d'une cellule.

enregistrer()

2 responsabilités :

- avoir le tableau sous forme de 0 et 1 ;
- enregistrer sous si aucun fichier n'est ouvert.

Donc ajout de 2 fonctions :

`get_tableau()` qui retourne le plateau de jeu avec des valeurs personnalisées de vivant et mort ;

`enregistrer_sous()`.

enregistrer_sous()

2 responsabilités :

- faire choisir le fichier ;
- enregistrer le fichier ouvert.

Donc ajout de 2 fonctions :

`QFileDialog` ouvre l'explorateur de fichier pour en sélectionner un ;

`enregistrer()`.

3.2.6 Application des consignes

Dans la version du Jeu de la vie avec interface graphique, il n'y a pas à proprement parler de méthode `affiche()`. C'est la combinaison « vue, qui regarde sur la scène, qui contient des cellules, qui sont peintes » qui créer l'affichage, sans même parler du framework.

3.2.7 Spécifications fonctionnelles détaillées

Fonction		Entrée		Rôle
Type	**Nom**	**Nom**	**Type**	
None	lance_anim()	self	JeuDeLaVie	Lance l'animation du jeu de la vie
None	run()	self	JeuDeLaVie	Lance l'animation et le chrono (séquenceur)
None	tour()	self	JeuDeLaVie	Execute un tour du jeu
None	resultat()	self	JeuDeLaVie	Retourne l'état de la cellule en fonction du nombre de voisins
		ord	int	
		absc	int	
int	total_voisins()	self	JeuDeLaVie	Retourne le total de voisins de la cellule
		ord	int	
		absc	int	
bool	valeur_case()	self	JeuDeLaVie	Retourne True si la cellule est vivante et donc est un voisin potentiel, False sinon.
		ord	int	
		absc	int	
bool	meurt()	nb_voisins	int	Retourne True si la cellule meurt
bool	nait()	nb_voisins	int	Retourne True si la cellule naît
list[list[Any]]	construit()	h	int	Construit un tableau de taille taille avec comme valeur valeur_defaut
		l	int	
		valeur_defaut	Any	
None	set_etat	self	JeuDeLaVie	Redéfinit l'état de la cellule
		etat	Etat	
None	peint()	self	JeuDeLaVie	Définit la couleur de la cellule
list[Direction]	doit_agrandir_tableau()	self	JeuDeLaVie	Retourne la liste des directions vers lesquelles il faut agrandir le tableau
None	vide_scene()	self	JeuDeLaVie	Retire les éléments de la scène
Etat	get_etat	self	JeuDeLaVie	Retourne l'état de la cellule
None	extension()	self	JeuDeLaVie	Étend le tableau vers une direction donnée
		direction	Direction	
bool	arret_automatique	self	JeuDeLaVie	Vérifie si deux tour de suite sont identique
bool	__eq__()	self	JeuDeLaVie	Méthode magique appelée par l'opérateur d'égalité ==
		valeur	Any	
bool	set_plateau()	self	JeuDeLaVie	Redéfinit l'attribut tableau de même dimension que le matrice. Si l'élément vaut
		plateau	list[list[Any]]	

Fonction		Entrée		Rôle
Type	**Nom**	**Nom**	**Type**	
		vivant	Any	vivant, son état sera Vivant, sinon Mort.
None	set_periode	self	JeuDeLaVie	Redéfinit la période/durée d'un cycle de l'animation.
		valeur	float	
bool	importe_template()	self	JeuDeLaVie	Importe le template si celui-ci est valide.
bool	est_template_valide()	tableau	Any	Retourne True si le tableau respecte le format d'un template.
				En cas d'erreur, le deuxième élément est le message d'erreur
None	mousePressEvent()	self	JeuDeLaVie	Détecte les événements de pression de souris sur la cellule.
		even	QGraphicsSceneMouseEvent	
None	zoom_in()	self	JeuDeLaVie	Zoom dans la vue
None	zoom_out()	self	JeuDeLaVie	Dézoom dans la vue
None	enregistrer()	self	JeuDeLaVie	Enregistre le plateau au format csv.
None	enregistrer_sous()	self	JeuDeLaVie	Enregistre sous le plateau au format csv.
None	event()	self	JeuDeLaVie	Capture les événements de QMainWindow, les traites, puis les rends.
		even	QEvent	

3.3 Architecture du projet

3.3.1 Division en solutions

Le projet est divisé en deux solutions (CLI et GUI) et une librairie de modules partagés entre les deux solutions. Aucun lien directe n'est nécessaire entre la solution CLI et GUI car il s'agit de deux versions du même projet, dont les exigences varient.

3.3.2 Librairie

Tous les comportements, de CLI et GUI, assez similaires pour être généralisé au deux versions sont mis en commun pour créer une librairie. Celle-ci est décomposée en plusieurs modules dont chacun regroupe plusieurs fonctions manipluant le même style de données, ayant des comportements similaire ou bien parce qu'elles sont liées.

Ainsi, on retrouverait `est_template_valide()`, `construit()` ou encore une classe (abstraite ou concrète) `JeuDeLaVie` qui serait mère des versions CLI et GUI puisque leur comportement est commun, en ce sens qu'elles représentent le Jeu de la vie. Cependant, les fonctionnements interne des méthodes est tellement différent qu'utiliser une même classe, même mère, pour faire tourner le Jeu de la vien'ajouterait que de la complexité, ce qui n'est pas l'objectif.

3.3.3 CLI

Le choix de l'Orienté Objet

Au vue des spécifications fonctionnelles, il est clair que la version CLI est représenté par une classe, mais expliquons cette décision. Une classe permet de rassembler des variables et des fonctions au sein d'une même structure de données : un objet. Cela permet une simplicité de développement (partage des attributs notamment) et une facilité de compréhension pour l'utilisateur qui n'a plus besoin que d'utiliser l'interface de la classe.

3.3.4 GUI

JeuDeLaVie

L'attribut scène a pour rôle de représenter le Jeu de la vie. Ainsi, ses responsabilités sont :

- `set_tableau()` : modifier le tableau ;
- `vide_scene()` vide la scène et le tableau ;
- `mouseMoveEvent()` capture les évenements de mouvement de souris ;
- `event()` fourit de l'aide à l'utilisateur ;
- `get_tableau()` retourne le tableau ;
- `valeur_case()` retourne True si la cellule est vivante, False sinon ;
- `total_voisins()` retourne le total de voisins ;
- `meurt()` définit les règles de mort ;
- `nait()` définit les règles de vivant ;
- `resultat()` retourne le nouvel état de la cellule ;

- `doit_agrandir_tableau()` retourne True si le tableau doit être agrandit ;
- `extension()` agrandit le tableau ;
- `arret_automatique()` retourne True si l'animation doit être arrêtée ;
- `tour()` exécute un tour de jeu ;
- `run()` lance l'animation.

Par application du principe *S*[11] des principes *SOLID*[12], l'attribut `scene` est séparé de la classe `JeuDeLaVie`. La scène a comme .devient une instance de `Scene` qui hérite de `QGraphicsScene`.

Scene

Les attributs cellules représentent les cases/cellules du Jeu de la vie. Ainsi leurs responsabilités sont :

- `peint()` redéfinit la couleur de la cellule en fonction de son état ;
- `mousePressEvent()` détecte les évenements de mouvements de souris ;
- `hoverEnterEvent()` détecte les évenements de souris, entrant sur la cellule ;
- `set_etat` redéfinit l'état de la cellule ;
- `get_etat` retourne l'état de la cellule ;
- `__repr__` méthode magique qui représente la cellule ;
- `__deepcopy__` méthode magique du module copy pour effectuer des copies de l'objet ;
- `__eq__` méthode magique appelée par l'opérateur d'égalité ==.

Par application du principe *S*, les instances de `QGraphicsRectItem` sont séprarée de la classe `Scene` et devient une instance de `Cellule` qui hérite de `QGraphicsRectItem`.

11. Principe S : https ://en.wikipedia.org/wiki/Single-responsibility_principle
12. Principes *SOLID* : https ://en.wikipedia.org/wiki/SOLID

Chapitre 4

Développement de la solution

4.1 Moyens de communication

Nos échanges et retours sur nos codes respectifs s'est d'abord effectué en face à face. Le code était échangé par clé USB. Avec l'arrivé des vacances, nous échangions par instagram et transmettions notre code via email. Après les vacances nous avons continuer d'échanger par instagram et nous repris les échanges en face à face. Par ailleurs, Eloi n'avais plus de tâches, j'ai donc publié le code sur un dépôt GitHub.

4.2 Organisation

Nous avons réparti les responsabilités comme suit :

Tâche	Elie	Eloi
Affichage basique		x
Affichage amélioré		x
Valeur case		x
Total voisins	x	
Tour	x	
Run		x
Arrêt automatique		x
Configuration pré-enregistrées		x
Importation des configurations	x	
Interface graphique	x	
Redéfinition des symboles		x
Agrandissement automatique	x	

4.3 Spécifications liées mes tâches

4.4 Attribution des crédit

Tout le code copié ou fortement inspiré du code de quelqu'un d'autre est déclaré comme tel dans le code source. Un lien vers la page web accompagne le dit code.

Voici les principales réutilisations :

- Le principe de fonctionnement de `Overload` et `signature()` du module overload (source) ;
- L'implémentation de la méthode magique `__deepcopy__()` (source).

Les dépendances sont les suivantes :

- time (Librairie Standard de Python)
- os (Librairie Standard de Python)
- typing (Librairie Standard de Python)
- sys (Librairie Standard de Python)
- csv (Librairie Standard de Python)
- importlib (Librairie Standard de Python)
- subprocess (Librairie Standard de Python)
- PySide6 (Version Python de Qt) [13]
- re (Librairie Standard de Python)
- copy (Librairie Standard de Python)
- enum (Librairie Standard de Python)

4.5 Tests

Pour chacune des fonctions développés, nous avons conçu un ensemble de données tests et leurs résultats par la fonction. En comparant la sortie de la fonction et le résultat attendu, nous avons pu recalibrer et corriger nos fonctions. Par ailleurs, nous avons fait usage de la fonction `ic()` du module `icecream` [14] et du debugger intégré de Python.

13. `pip install PySide6`
14. Code source du module icecream : https://github.com/gruns/icecream.

Chapitre 5

Retour d'expérience

5.1 Liste des améliorations possibles

- Faire des fonctions plus générique pour une utilisation plus générique :
  - `total_voisins`
  - `meurt`
  - `nait`
  - `resultat`
  - `tour`
  - `arret_automatique`
- La version CLI peut être redévelopper avec QCoreApplication de PySide6. Cela permettrait d'avoir les mêmes fonctionnalités que la version GUI, et de ne plus utiliser sleep, qui est peut optimisé, surtour pour l'affichage CLI/GUI (boucle while sur le même thread).
- Offrir une plus grande diversité de paramètres et d'éléments personnalisables. Par exemple, changer la couleur des cellules, personnaliser les règles du jeu, etc.
- Optimiser plus encore l'application graphique, en cherchant des techniques d'économie de ressources (RAM & CPU principalement). Une solution pourrait être l'utilisation de calculer les changements durant le temps d'attente (si possible) et de diminuer la quantité d'espace de mémoire vive utilisé par chaque cellule (méthode ou attribut inutiles)
- Clarifier l'interface graphique en regroupant mieux les éléments liés.
- Ajouter une traduction de la version GUI, car PySide6 le permet assez aisément.
- Écrire une documentation de meilleur qualité que celle générée avec Doxygen, un outil non adapté au langage Python [15].
- Ajouter plus de configuration pré-enregistée.
- Développer un algorithme capable de traduire une image représentant le Jeu de la vieen une configuration pré-enregistée. (cadrillage automatique de l'image, reconnaissance des couleurs, etc.)

15. Site web : https://www.doxygen.nl/

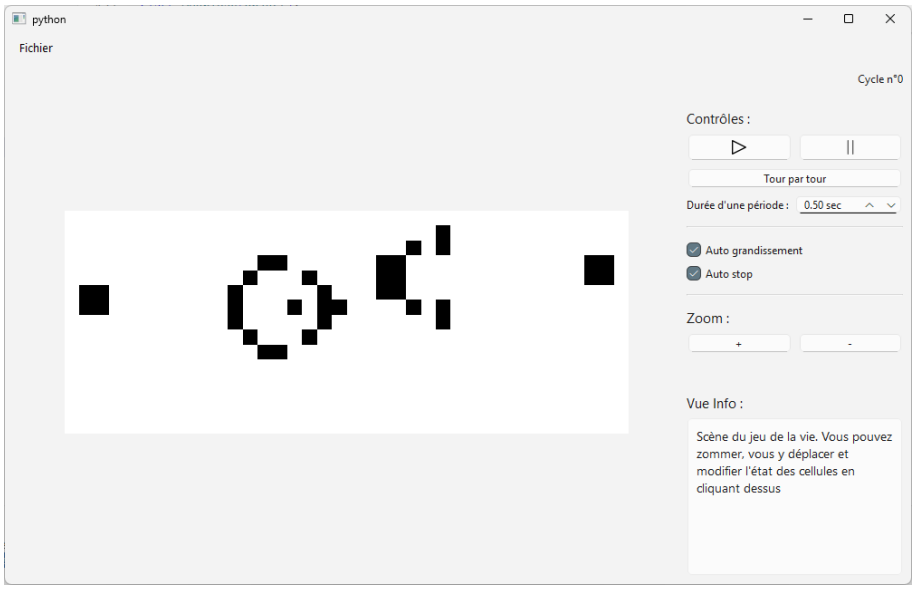

Figure 5.1 – Capture d'écran de l'interface graphique

5.2 Regard critique sur le projet

Malgé une interface professionnelle, l'application reste imparfaite. En effet, son optimisation reste à revoir et à améliorer, notamment lorsque la scène est conséquente en taille. Par ailleurs, l'architecture laisse à désirer tant elle est à refaire. J'aurais voulu créer une classe mère `JeuDeLaVie`dont la version CLI et `Scene` aurait hérité pour éviter de réécrire le même code plusieurs fois, ce qui multiplie les tests et les probabilités d'erreur de conception et de bugs.

Chapitre 6

Remerciements

6.1 À Eloi

Merci Eloi de m'avoir fait confiance pour architecturer la version CLI, pour avoir supporté mon perfectionnisme, mes modifications à tout-va et mes messages euphoriques que je t'ai envoyés ! Merci de m'avoir épaulé comme tu l'as fait, j'espère que tu es fier de toi !

6.2 À C. NAULEAU

Merci Monsieur d'avoir supporté ces 28 pages mal organisées !

6.3 Aux autres

Merci à Jason Champagne, alias *FormationVideo*[16], pour m'avoir montrer Python sous un jour nouveau. Merci à José Paumard[17] pour m'avoir tout appris sur l'architecture d'application. Merci à internet, et particulièrement à sa communauté, pour avoir répondu à chacune de mes intérogations, pour m'avoir accompagné et permis de créer cette version GUI. Et pas merci à LaTeX d'être si complexe à comprendre (surtout à compiler).

16. Chaîne YouTube : https://www.youtube.com/@formation-video
17. Chaîne YouTube : https://www.youtube.com/@coursenlignejava