



Rapport de stage

BUT Informatique

Réalisation d'Applications : Conception, Développement, Validation (RACDV)

Informatisation du jeu sérieux *Course contre la mouche*

Réalisé par Julien RENAUD

Tuteur de stage :

Dr. Simon ROBILLARD (LIRMM, UM)

Encadrant :

Dr. Isabelle GRECHI (CIRAD UPR HortSys)

Co-encadrants :

Dr. François REBAUDO (IRD UMR EGCE)

Dr. Raphael BELMIN (CIRAD UPR HortSys)

Du 13 Janvier au 4 Avril 2025

Remerciements

Je tiens à remercier le CIRAD de m'avoir accueilli pour la réalisation de mon stage de deuxième année de BUT Informatique. Je remercie notamment Dr. Isabelle GRECHI (CIRAD UPR HortSys), mon maître de stage, Dr. François REBAUDO (IRD UMR EGCE) et Dr. Raphael BELMIN (CIRAD UPR HortSys), mes co-encadrants de stage, pour leur expertise, leur suivi régulier et leurs conseils tout au long de la réalisation du projet. Je remercie également Dr. Simon ROBILLARD (LIRMM, UM), mon tuteur universitaire, ainsi que l'équipe pédagogique de l'IUT de Montpellier. Finalement, j'aimerais remercier toute l'UPR Hortsys et tout le personnel du CIRAD qui m'ont accueilli et guidé tout au long de mon stage.

Abstract

The oriental fruit fly (*Bactrocera dorsalis*) is a major pest of crops, causing significant harvest losses in Africa. In response to this issue, a hybrid serious game based on a scientific model has been developed : *Race Against the Fly*. The goal of this game is to promote coordinated management of the mango supply chain in the fight against the fruit fly infestation. This project is part of the ongoing development of the game and focuses on several key areas for improvement such as : making manual data entry phases more fluid, ensuring clear traceability of output results for scientific research, and internationalization. Achieving these objectives requires partial computerisation of the game, including the development of an ergonomic graphical user interface hosting the model. For distribution purposes, the application will be made available as an executable compatible with the three major operating systems : *Windows*, *Linux*, and *macOS*.

Keywords : hybrid serious game, fruit fly, mango, infestation, graphical user interface, computerisation, scientific model

Résumé

La mouche des fruits orientale (*Bactrocera dorsalis*) est un ravageur majeur des cultures, provoquant des pertes considérables de récoltes en Afrique. Face à ce défi, un jeu sérieux hybride basé sur un modèle scientifique a été développé : *Course contre la mouche*. L'objectif de ce jeu est de promouvoir une gestion concertée de la filière mangue pour lutter contre l'infestation de la mouche des fruits. Ce projet s'inscrit dans la continuité du développement du jeu et vise plusieurs axes d'amélioration : la fluidification des phases de saisie des données, la traçabilité précise des résultats pour la recherche scientifique, et l'internationalisation. Ces objectifs nécessitent une informatisation partielle du jeu, incluant le développement d'une interface graphique ergonomique hébergeant le modèle. Dans le but de faciliter sa distribution, l'application est proposée sous forme d'un exécutable compatible avec les trois systèmes d'exploitation principaux : *Windows*, *Linux* et *macOS*.

Mots-clés : jeu sérieux hybride, mouche des fruits, mangue, infestation, interface graphique, informatisation, modèle scientifique

Table des matières

Remerciements	2
Abstract	3
Résumé	4
Table des matières	6
Table des figures	7
Table des équations	8
Glossaire	9
Acronymes	11
1 Introduction	12
2 Contexte du stage	13
2.1 Présentation du CIRAD	13
2.2 Présentation de l'UPR Hortsys	13
2.3 Enjeux du stage	14
3 Analyse	16
3.1 Spécificités fonctionnelles	16
3.2 Spécificités non fonctionnelles	16
3.3 Analyse du modèle existant	17
4 Rapport technique	21
4.1 Environnement de développement	21
4.2 Structure du projet	22
4.3 Conception de la mécanique du jeu	22
4.4 Retranscription du modèle en Python	23
4.5 Outils du jeu	26
4.5.1 Parser	26
4.5.2 Logger	27
4.5.3 Saver	28
4.5.4 Settings	30
4.6 Internationalisation	31
4.7 Identification des différents états du jeu	32
4.8 Conception GUI	34

4.8.1	Libraries graphiques	34
4.8.2	Prototype <i>Figma</i>	35
4.8.3	Conception et organisation des vues	36
4.8.4	Responsive	38
4.8.5	Widgets et styles	40
4.9	Packaging	42
4.10	Tests et validation	43
4.11	Difficultés rencontrées	45
5	Méthodologie du travail	47
5.1	Organisation du projet	47
5.2	Gitflow	47
5.3	Activités annexes	47
6	Conclusion	48
A	Organigramme du CIRAD	51
B	Diagrammes UML	52
C	Déroulement du jeu	55
D	Protype Figma	56
E	Tutoriel traduction	59
F	Tutoriel personnalisation	63
G	Tutoriel packaging	66

Table des figures

3.1	Diagramme états-transitions déduit du script R	19
3.2	Extrait du script R : saisie manuelle des pratiques jouées sous forme de chaînes de caractères	19
3.3	Extrait du script R : fonction de calcul de la production réelle	20
4.1	Arborescence du projet	22
4.2	Extrait de la fonction <code>calculRdt</code> du script R	23
4.3	Extrait de la fonction <code>calc_rdt</code> de la classe (<code>Modele</code>)	24
4.4	Extrait de la fonction <code>get_itkmouche_score</code> du script R	24
4.5	Extrait du fichier de configuration <code>modele_config.yaml</code>	25
4.6	Extrait de la fonction <code>get_score_pratique_mouche</code> de la classe <code>Modele</code>	25
4.7	Fichier de configuration <code>tool_config.yaml</code>	26
4.8	Diagramme de classes UML simplifié du <i>parser</i>	27
4.9	Diagramme de classes UML simplifié du <i>logger</i>	27
4.10	Exemple de fichier de logs généré	28
4.11	Diagramme de classes UML simplifié des <i>savers</i>	29
4.12	Exemple d'un graphique de sortie interactif affiché sur le GUI	29
4.13	Interface de modification des paramètres avancés du GUI	30
4.14	Implémentation de la classe <code>Language</code>	31
4.15	Extrait du dictionnaire de langue dans <code>en_dict.py</code>	31
4.16	Exemple d'utilisation de la méthode <code>get_text</code>	32
4.17	Diagramme de classes UML simplifié des états du jeu	33
4.18	Tableau récapitulatif des différentes librairies graphiques Python	34
4.19	Prototypage sur Figma, gestion des transitions et actions utilisateurs	35
4.20	Diagramme de classes UML simplifié de l'application	36
4.21	Exemple d'un widget réutilisable par le GUI : bouton par défaut	37
4.22	Redimensionnement de l'affichage des cartes (2880x1800, zoom 225%)	39
4.23	Exemple d'un widget utilisant l'appel de méthodes en cascade	40
4.24	Composants réutilisés : <code>DefaultFrameSlider(DefaultFrame)</code>	40
4.25	Composant complexe : <code>RecapFrameTableMatrice</code>	41
4.26	Exemple de test unitaire : fonction <code>calc_rdt</code> du modèle	44
4.27	Exemple de tests validés avec <i>GitHub Actions</i>	44
4.28	Tests fonctionnels rédigés pour la version 1.0 de l'application	45
4.29	Exemple de message d'erreur dû à une importation circulaire	46
4.30	Solution envisagée face à l'importation circulaire	46
B.1	Diagramme de classes UML de conception de la mécanique du jeu (en bleu les packages internes et en jaune les packages externes)	53
B.2	Diagramme etats-transitions de l'objet <code>Jeu</code>	54

Table des équations

3.1 Calcul du niveau d'infestation d'un agriculteur	17
3.2 Calcul de la production d'un agriculteur	18

Glossaire

Anaconda distribution de Python spécialement conçue pour la science des données, le machine learning et l'analyse de données. Elle inclut un grand nombre de bibliothèques scientifiques. [21](#), [46](#)

CPython distribution officielle du langage Python, écrite en C. [21](#), [46](#)

CSS (Cascading Style Sheets) langage de style utilisé pour décrire l'apparence et la mise en forme des pages web. [41](#)

csv (Comma-separated values) format de fichier texte utilisé pour représenter des données tabulaires sous forme de valeurs séparées par des virgules. [16–18](#), [22](#), [26](#)

dataframe structure bidimensionnelle de données qui organise les données sous forme de lignes et de colonnes similaires à une feuille de calcul, à la différence où les colonnes sont typées et nommées et qu'il est possible de partitionner un même dataframe sur plusieurs machines. Cette structure est couramment utilisée dans l'analyse et dans l'exploitation de données. [17](#), [28](#), [29](#), [32](#)

DPI (Dots Per Inch) unité de mesure utilisée pour évaluer la résolution des images imprimées ou affichées à l'écran. Elle indique combien de points (ou pixels) sont présents dans un pouce. [38](#)

Figma outil de conception d'interface utilisateur et de prototypage en ligne. [35](#)

GitHub plateforme en ligne qui permet d'héberger du code, de collaborer sur des projets et de gérer des versions de fichiers à l'aide de *Git*, un système de gestion de version. [21](#)

HTML (HyperText Markup Language) langage de balisage utilisé pour structurer le contenu des pages web. [38](#), [41](#), [42](#)

interface en programmation orientée objet, classe abstraite (non instanciable) et sans attributs et qui liste toutes les méthodes que doivent implémenter les classes qui l'implémentent. Il s'agit d'un contrat que doit respecter les objets qui implémentent l'interface. [36](#)

jeu sérieux activité qui combine une intention sérieuse, comme l'éducation, l'information ou la formation, avec des éléments ludiques. [12](#), [14](#)

Mocks objet qui sert à simuler le comportement d'une classe, utilisé pour les tests. [26](#), [43](#)

package en *Python*, un ensemble de modules dans un même répertoire. [27](#), [37](#)

pipeline chaîne de processus ou d'étapes successives. [44](#)

pop-ups fenêtre ou élément qui apparaît soudainement sur l'écran. [37](#)

- pull request** action de demander la fusion des modifications sur une branche dans un dépôt de code. 47
- Python** langage de programmation interprété et typé dynamiquement. Il est très utilisé dans le monde scientifique pour ses librairies de calculs numériques. 16, 17, 21–23, 27, 34, 41–46, 48
- R** langage de programmation et logiciel dédiés à la science des données et à la statistique. 17–19, 23, 27, 43
- refactoring** processus de restructuration du code existant sans altérer son fonctionnement. Il est très souvent réalisé dans l’optique d’améliorer la clarté et la modularité du code. 12, 17, 20, 48
- responsive** approche de conception qui permet à une application de s’adapter automatiquement à différentes tailles d’écrans et résolutions. 38
- template** modèle réutilisable qui permet de généraliser du code. 32
- vue** représentation des données ou des informations dans un système informatique. Elle désigne l’affichage, l’organisation ou la manière dont un utilisateur interagit avec les données. 35–38, 41, 48
- widget** élément graphique interactif et autonome qui permet aux utilisateurs d’interagir avec une application. 36–38, 40–42
- wrapper** code qui entoure une autre fonction, classe ou bibliothèque pour ajouter des fonctionnalités sans modifier le code original. 27
- xlsx** (*Excel Microsoft Office Open XML Format Spreadsheet file*) format de fichier par défaut pour les feuilles de calcul *Microsoft Excel* depuis la version 2007. 18, 33
- XML** (Extensible Markup Language), langage de balisage conçu pour stocker, organiser et transporter des données de manière lisible par les humains et les machines. 35

Acronymes

ANR Disland Projet financé par l’Agence Nationale de Recherche : Inferring pest DISpersal in agricultural LANDscapes to improve management strategies. 12, 14

Bios Systèmes biologiques. 13

CIRAD Centre de coopération Internationale en Recherche Agronomique pour le Développement. 12–14, 48

CLI Interface en Ligne de Commande (Command Line Interface). 21, 33, 47

ES Environnements et sociétés. 13

GUI Interface Graphique Utilisateur (Graphical User Interface). 21, 29–31, 33, 36–38, 41, 42, 44, 45, 47

Hortsys Fonctionnement agroécologique et performances des systèmes de culture horticoles. 13, 14, 47

IDE Environnement de Développement Intégré (Integrated Development Environment). 21

INRAE Institut National de Recherche pour l’agriculture, l’alimentation et l’environnement. 12

IRD Institut de recherche pour le développement. 12, 14

ISRA Institut Sénégalais de Recherche Agricole. 12

IUT Instituts Universitaires de Technologie. 17, 48

ODD Objectifs de Développement Durable. 13

ONU Organisation des Nations Unies. 13

PDF Portable Document Format. 41

Persyst Performances des systèmes de production et de transformation tropicaux. 13

POO Programmation Orientée Objet. 18, 22, 34, 46, 48

UML Langage de Modélisation Unifié (Unified Modeling Language). 22

UPR Unité propre de recherche. 13, 14, 47

Chapitre 1

Introduction

La mouche orientale des fruits (*Bactrocera dorsalis*) est un ravageur des cultures originaire d'Asie qui est responsable de nombreuses pertes de récoltes sur le continent africain. Identifiée comme ravageur de plus de 300 plantes hôtes, et notamment de la mangue (*Mangifera indica*), cette mouche a été classée organisme de quarantaine prioritaire aux États-Unis d'Amérique, en Australie et dans l'Union européenne.

Pour faire face à ce problème, le [Centre de coopération Internationale en Recherche Agronomique pour le Développement \(CIRAD\)](#) en partenariat avec l'[Institut Sénégalais de Recherche Agricole \(ISRA\)](#) développe depuis plusieurs années des techniques de prévention et de contrôle de la mouche. Ces techniques, bien qu'efficaces, ne permettent toutefois pas de réduire notablement les pertes de récoltes, car leur application reste encore insuffisante et hétérogène entre les producteurs [1]. C'est dans ce contexte que s'inscrit le projet [ANR Disland](#) [2]. Initié en 2022 et financé par l'Agence Nationale de la Recherche, il regroupe des chercheurs, ingénieurs et techniciens du [CIRAD](#), de l'[Institut National de Recherche pour l'agriculture, l'alimentation et l'environnement \(INRAE\)](#) et de l'[Institut de recherche pour le développement \(IRD\)](#). Il s'agit d'un projet ayant pour but d'inférer les modalités de dispersion de la mouche des fruits dans les paysages agricoles sénégalais, par des approches de génétique du paysage. L'objectif est de représenter les interactions entre la dynamique des populations du ravageur et l'infestation annuelle des vergers, la structure du paysage, et les actions des acteurs du système socio-technique, afin d'améliorer les stratégies de gestion du ravageur. [3]

C'est dans ce contexte qu'un [jeu sérieux*](#), "Course contre la mouche", a été développé pour simuler l'apprentissage, l'action collective et l'innovation institutionnelle pour une meilleure gestion de la mouche des fruits par les acteurs de la filière mangue [4] [5]. Ce jeu hybride repose sur un modèle de simulation, des supports physiques et un protocole d'animation. Le jeu a été testé plusieurs fois en interne et déployé sur le terrain sénégalais en juin 2023. La mission que l'on m'a confiée s'inscrit parmi les axes d'améliorations du jeu. Il s'agit d'un travail de [refactoring*](#) et réécriture du code existant du modèle, de la réalisation d'une interface graphique ergonomique dans l'optique d'universaliser et fluidifier l'un des processus du jeu, et l'ajout de nouvelles fonctionnalités.

Après avoir présenté le [CIRAD](#), en expliquant son statut, organisation et objectifs, je présenterai le jeu en détail en spécifiant son déroulement ainsi que les enjeux de mon stage. Ensuite, j'aborderai l'analyse du projet en explicitant les attentes et les contraintes. Je terminerai cette section par l'analyse du modèle existant pour mettre en valeur l'intérêt de ma mission. Suite à cette analyse, j'évoquerai les différentes étapes de conception, de réalisation et de validation dans le cadre du développement de l'application. Enfin, je détaillerai la méthodologie appliquée, avant de conclure en partageant les apprentissages réalisés et les axes d'amélioration du projet.

Chapitre 2

Contexte du stage

2.1 Présentation du CIRAD

Le **CIRAD** est un organisme français fondé en 1984 de recherche agronomique et de coopération internationale pour le développement durable des régions tropicales et méditerranéennes. Il s'inscrit en tant que société d'Etat classée comme Établissement Public à Caractère Industriel et Commercial (EPIC). Le **CIRAD** intervient dans plus d'une centaine de pays et collabore en partenariat avec plus de 200 institutions sur tous les continents. Son objectif est de contribuer à un monde plus durable et solidaire à travers le développement de systèmes agricoles et alimentaires résilients, qui nourrissent sainement les populations et qui rémunèrent décemment les productrices et les producteurs, tout en préservant la biodiversité et les ressources naturelles. Son activité répond aux **Objectifs de Développement Durable (ODD)** de l'**Organisation des Nations Unies (ONU)**, notamment l'élimination de la pauvreté (**ODD 1**) et l'éradication de la faim et de la malnutrition (**ODD 2**).

Le **CIRAD** est localisé sur plusieurs sites de recherche et d'activités à travers la France et le monde. Les sites principaux du **CIRAD** se trouvent à Montpellier : le campus Baillarguet et le campus Lavalette. Le siège social du **CIRAD** est situé à Paris, et l'organisation dispose également de stations de recherche dans les départements et territoires d'outre-mer français.

A ce jour, le **CIRAD** rassemble près de 1750 salariés et 29 unités de recherche multidisciplinaires. Ces unités sont réparties dans trois départements scientifiques : **Systèmes biologiques (Bios)**, **Environnements et sociétés (ES)** et **Performances des systèmes de production et de transformation tropicaux (Persyst)**.

2.2 Présentation de l'UPR Hortsys

Le stage se déroule au sein de l'**Unité propre de recherche (UPR) Hortsys** du département **Persyst**. Elle est spécialisée dans l'horticulture [6]. L'**UPR Hortsys**, en partenariat avec les acteurs locaux, étudie les principes agroécologiques des systèmes horticoles tropicaux dans l'optique d'assurer une production saine, élevée et durable de fruits et légumes. L'**UPR Hortsys** est constitué d'une cinquantaine d'agents permanents, incluant des cadres scientifiques, techniciens, assistants administratifs, et accueille des doctorants et étudiants. Les agents de l'**UPR Hortsys** sont basés en métropole (à Montpellier), dans trois départements d'outre-mer (en Martinique, à La Réunion et en Guadeloupe), en Afrique de l'Ouest (Sénégal, République de Côte d'Ivoire, Bénin et Burkina Faso) et à Madagascar. Le stage prend place dans cet environnement sur le site de Baillarguet à Montferrier-sur-Lez.

2.3 Enjeux du stage

Le stage se déroule dans le cadre du projet [ANR Disland](#), dans le but d’informatiser le [jeu sérieux](#) intitulé *Course contre la mouche*. Ce jeu a été développé et formalisé en 2023 par une équipe de quatre chercheurs du [CIRAD](#), de l’[IRD](#), et une étudiante d’AgroParisTech [4] [5]. Il s’agit d’un jeu de rôle hybride basé sur un modèle scientifique ayant pour but de sensibiliser les populations locales sur la gestion des exploitations de mangues vis à vis du ravageur. Il vise à « accompagner les principaux acteurs de la filière mangue, les producteurs et les acheteurs, vers une gestion individuelle et collective du risque lié à l’infestation des fruits par les mouches » [4]. Je m’inscris dans ce contexte en tant que développeur informatique affilié à l’[UPR Hortsys](#). L’unité de recherche étant principalement constituée d’agronomes, les missions demandent une grande autonomie au niveau des solutions informatiques envisagées, ainsi que des compétences en pédagogie pour transmettre les développements réalisés.

Le [jeu sérieux](#) *Course contre la mouche* repose sur un modèle de simulation informatisé, des supports physiques pour matérialiser les objets (vergers, argent, récoltes, pratiques agricoles) et d’un protocole d’animation. Il est principalement destiné aux équipes de recherche et aux acteurs de la filière mangue en Afrique concernés par les infestations de la mouche des fruits (*Bactrocera dorsalis*).

Le jeu comprend un groupe d’animateurs (classiquement deux), des joueurs agriculteurs (classiquement six), et des joueurs acheteurs (classiquement deux). L’objectif du joueur agriculteur est de gérer son verger afin de maximiser sa production de mangues en jouant des cartes de pratiques agricoles pour entretenir son verger et/ou lutter contre la mouche des fruits. L’objectif des acheteurs est d’atteindre leur objectif d’achat défini en début de tour, en achetant les mangues les moins infestées possibles aux producteurs pour réduire les probabilités de perte post récolte ou d’inventus à l’international.

Un joueur agriculteur est caractérisé par la surface agricole de son exploitation, sa capacité de production potentielle, sa composition variétale et son capital de départ. Il existe deux grands types de variétés : celles destinées au marché domestiques (généralement des variétés locales) et celles destinées au marché d’exportation. Elles se caractérisent aussi par leur précocité (variétés précoces, de saison ou tardives). Un joueur acheteur est caractérisé par un capital de départ, une catégorie (exportateur ou bana-bana pour le marché domestique) qui déterminent leur objectif d’achat en type de variétés et tonnage de mangues. Les cartes représentent chacune une pratique agricole. Elles sont caractérisées par un coût par unité de surface, un score lié à son efficacité face à la lutte contre la mouche, et un score lié à son impact sur la production. Les scores sont inconnus des joueurs.

Un tour de jeu simule une campagne agricole d’un an où les agriculteurs vont gérer leur verger puis vendre leurs mangues aux acheteurs. Le déroulement d’un tour de jeu peut se résumer en trois grandes phases. Premièrement, l’annonce des conditions initiales par les animateurs, notamment le niveau de pression globale de la mouche (lié aux conditions environnementales). Il s’ensuit une phase de choix de pratiques agricoles durant laquelle les agriculteurs sélectionnent les cartes à jouer. Ensuite, l’animation s’occupe de saisir toutes les informations nécessaires, en particulier la saisie des pratiques jouées par chaque agriculteur, dans le modèle. L’animateur annonce aux joueurs les taux d’infestation et les rendements par agriculteur prédits par le modèle.

La dernière phase du jeu consiste en une période de négociation entre producteurs et acheteurs, au cours de laquelle les acheteurs achètent des lots de mangues aux producteurs. Les pertes post-récolte sont ensuite définies sur la base du taux d'infestation des lots achetés. A chaque tour de jeu, l'animateur organise un debriefing sur les pratiques jouées et les performances des joueurs.

L'enjeu du stage est l'amélioration de la phase de configuration et de saisie des données dans le modèle du jeu, de manière à le rendre plus universel, fluide, ergonomique et accessible à un utilisateur sans compétences informatiques. Les attentes sont : la clarté de la documentation, la modularité du modèle, la génération de données de sortie à des fins d'analyse statistique, et l'ergonomie de l'interface.

Chapitre 3

Analyse

3.1 Spécificités fonctionnelles

L'équipe de chercheurs a pour objectif de rendre le jeu plus universel et compréhensible, afin de fournir un support d'éducation et de sensibilisation simple et ludique. Ainsi, la première étape de ma mission consiste à la réécriture et à la documentation du code du modèle. Par ailleurs, les sessions de jeu réalisées sur le terrain sénégalais et la session de test réalisée en interne en décembre 2024 et à laquelle j'ai pu participer, ont permis de pointer les axes d'améliorations possibles du jeu. Parmi ceux-ci, le temps de saisie des cartes jouées dans le modèle prenait trop de temps et avait tendance à disperser les joueurs. C'est pourquoi, et toujours dans l'optique de rendre le jeu universel, ma seconde mission est de concevoir et de développer une interface graphique utilisateur qui permet de saisir et de configurer aisément les données du modèle.

Un document de travail résumant l'ensemble des objectifs du stage m'a été transmis. De plus, une réunion avec mes trois encadrants a été réalisée en début de stage pour mieux préciser les spécificités fonctionnelles et non-fonctionnelles, et établir les besoins primaires de l'application :

- Une version refactorisée et modulaire du code du modèle existant et sa documentation
- Une interface graphique pour configurer, saisir les données du jeu et visualiser les résultats d'une partie
- Un système de génération de données résultats de sortie
- Un système qui permet de retracer le déroulement d'une partie
- Un système qui permet de retrouver les configurations utilisées lors d'une partie
- Un système de changement de langues à intégrer

3.2 Spécificités non fonctionnelles

La contrainte est que le code soit fourni sous **Python***, un langage de programmation largement utilisée par la communauté scientifique pour sa facilité de compréhension. Le code doit prendre en entrée un fichier **.csv*** fourni lié aux données des cartes, et des fichiers de configuration du jeu (format au choix). Les fichiers de configuration doivent être modifiables avant le lancement d'une partie. Des modules complémentaires, par rapport au modèle existant, seront développés en parallèle au stage et devront être intégrées. Le code doit pouvoir accueillir ces nouvelles extensions supplémentaires. Le code ne doit pas contenir d'élément sous une licence commerciale. Le tout devra être livré sous la forme d'un unique exécutable ou bien d'un installateur.

L'application est à usage hors connexion et doit être compatible sur les trois principaux systèmes d'exploitation (*Windows*, *Linux* et *macOS*).

Un défi du stage est de fournir un code fonctionnel et modulable en **Python** à partir d'un modèle codé en **R***, un langage qui m'est inconnu jusqu'à présent. Des tests unitaires seront à réaliser pour valider le modèle, de sorte à ce que le modèle sous **Python** produise des résultats identiques au modèle sous **R**.

Un autre défi de cette mission est d'adopter les bonnes pratiques et principes fondamentaux de qualité de développement enseignés à l'**Instituts Universitaires de Technologie (IUT)** dans un contexte concret et à travers l'utilisation d'un langage de programmation différent de ceux enseignés. **Python** étant un langage interprété, et typé dynamiquement, et plus dirigé vers l'approche fonctionnelle, des adaptations seront à prévoir au moment de la conception et de la réalisation d'une application modulable.

3.3 Analyse du modèle existant

Le modèle de simulation a pour objectif de quantifier l'effet des pratiques agricoles sur la production et sur l'infestation des fruits par la mouche [4]. Le modèle actuel a été développé sur un unique script en **R** version 4.2.2 [7], par une personne de formation agronome (non informaticien). **R** est un langage de programmation et un logiciel libre dédié à la statistique et à la visualisation des données.

Le premier défi du stage consiste à l'analyse du script **R** mis à disposition, en vue d'une ré-implémentation sous **Python** en prenant en compte les différents éléments à améliorer (**refactoring**). L'installation du logiciel **R** et des bibliothèques utilisées par le script (*writexl*, *xlsx*, *ggplot2*, *dplyr*, *tidyr*) était nécessaire pour son exécution. Les bibliothèques sont accessibles via le dépôt principal des bibliothèques **R**, nommé CRAN (Comprehensive R Archive Network). La bibliothèque *xlsx* utilisée par le script nécessite une dépendance extérieure *rJava*, une interface bas niveau pour la Machine Virtuelle Java le (JVM) qui permet d'utiliser les fonctionnalités Java sous un environnement **R**. Par conséquent, il a été nécessaire de procéder à l'installation de l'environnement d'exécution Java (JRE) et la configuration de la variable d'environnement `JAVA_HOME` qui pointe vers le JRE.

Le script est structuré en trois blocs à exécuter manuellement. Le bloc d'initialisation permet d'extraire les données d'entrée et de configurer les paramètres initiaux du jeu. Les données concernant les différentes pratiques (ou cartes) jouables sont définies dans un fichier extérieur au format *.csv*. Le script **R** se charge d'extraire ces données et de les stocker sous la forme d'un *dataframe**. L'initialisation comprend aussi la définition des différentes fonctions utilisées par le modèle et des variables de type *dataframe* pour stocker les résultats. Le modèle est principalement décrit par deux équations [4] :

Calcul du niveau d'infestation des fruits d'un agriculteur :

$$I = \begin{cases} 1 & \text{si } x > 1 \\ x = I_{env} \left(1 - 0.5 \left(\frac{score_I}{seuil_I} - 1 \right) \right) & \text{si } x \in [0; 1] \\ 0 & \text{si } x < 0 \end{cases} \quad (3.1)$$

- I_{env} est le niveau d'infestation global annuel lié au climat et qui touche tous les agriculteurs. Ce niveau est défini au début d'un tour et peut augmenter ou baisser en fonction des pratiques réalisées par les agriculteurs.
- $score_I$ est le score d'impact lié à l'incidence de la mouche
- $seuil_I$ est le score d'impact seuil à atteindre pour réduire le niveau d'infestation global

Calcul de la production d'un agriculteur :

$$P = 0.5P_{pot} + 0.5P_{pot} \times (1 - I) \times \frac{score_p}{seuil_p} \quad (3.2)$$

- P_{pot} est la production potentielle d'un agriculteur qui dépend de sa surface et de ses variétés
- I est le taux d'infestation individuelle des mangues de l'agriculteur calculé par (3.1)
- $score_p$ est le score d'impact lié à la production de mangues
- $seuil_p$ est le score d'impact seuil à atteindre pour maintenir la production optimale en l'absence du ravageur (il est fixé au score d'impact maximal vis-à-vis de la production qui puisse être obtenu)

Ces deux formules sont décomposées en plusieurs fonctions intermédiaires qui calculent les scores et les valeurs seuils qui dépendent des cartes jouées et des données du fichier des pratiques agricoles *.csv* fourni en entrée.

Le deuxième bloc du script est constitué de la boucle du jeu, dans lequel les conditions initiales du tour sont définies et les cartes jouées par un joueur sont renseignées en brut dans le code. Les cartes jouées par un joueur agriculteur sont représentées par un vecteur de chaînes de caractères correspondant à leurs abréviations. Chaque tour de jeu est une duplication de ce bloc de code exécuté manuellement lors de chaque tour avec des configurations différentes.

Le troisième bloc correspond au traitement, à la restitution des données en fin de partie, et à la génération des fichiers de résultats. Le script se charge de générer une feuille de calcul *.xlsx* contenant les résultats calculés par le modèle et l'information sur les cartes jouées par les joueurs. Il génère aussi des graphiques de résultats.

Le jeu sous forme de script **R** ne comprend pas d'instance d'objet (**Programmation Orientée Objet (POO)**), mais on peut modéliser les différents états du jeu à travers la "vie" d'utilisation du script, comme montré dans le diagramme états-transitions suivant :

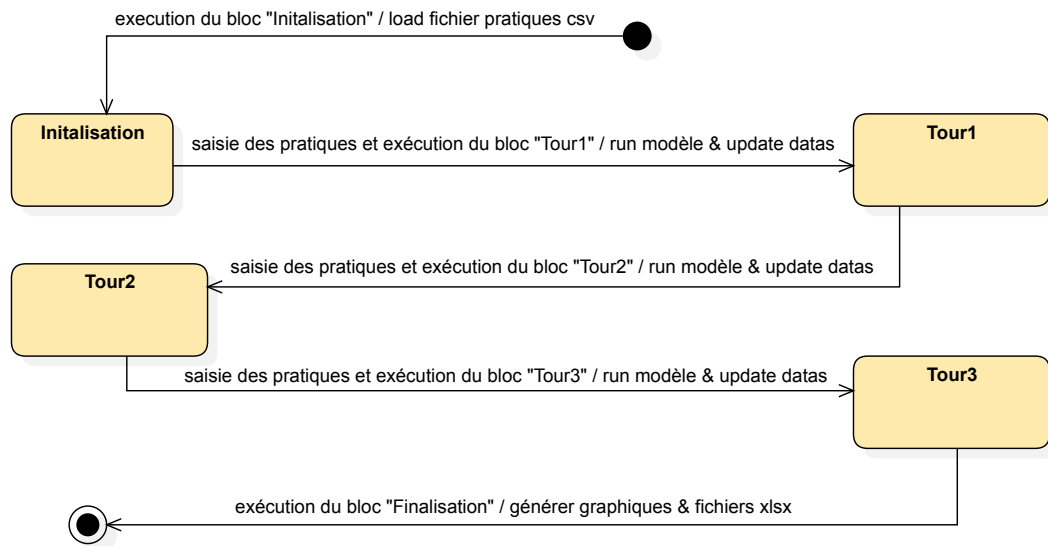


FIGURE 3.1 – Diagramme états-transitions déduit du script R

Chaque action de l'utilisateur correspond à une ré-exécution d'un bloc du script, avec, à chaque début de tour, la saisie manuelle des cartes en brut dans la section du code correspondante. C'est en partie ce processus que ma mission vise à améliorer et fluidifier.

```

1 bloc01 <- {
2   ...
3   ag01 <- list(
4     itk = c("dsb", "irr", "pba", "bio", "fer", "pre", "bio"),
5     ...
6   )
7   ...
8 }

```

FIGURE 3.2 – Extrait du script R : saisie manuelle des pratiques jouées sous forme de chaînes de caractères

L'analyse du code R a permis de mieux comprendre le déroulement de cette phase de jeu et d'identifier les variables d'entrée et de sortie du modèle. Cette analyse a également permis de comprendre la nécessité de passer sur un format plus modulable et fluide. En effet, le code est peu documenté et les différentes responsabilités telles que l'initialisation, la simulation, le traitement, la sauvegarde et l'affichage sont toutes centralisées dans un seul script. Malgré une séparation judicieuse en trois blocs, cette configuration rend le code difficilement modulable.

```

1  # Calcul nouveau taux de perte et rdt avec le taux de note ITK et
    incidence BD
2  calculRdt <- fonction() {
3  if (sum(calculTxNoteITKmouche() >= 1) >= (0.5 * (length(listAgriITKetX)))) {
4    newTauxDePertesBD <- tauxDePertesBD - tauxDePertesBD * 0.1
5  } else {
6    newTauxDePertesBD <- tauxDePertesBD + tauxDePertesBD * 0.1
7  }
8  msg <- paste0("Taux_de_pertes_BD_initial:_", ... , ".\n")
9  cat(msg)
10  tauxDePertesBD <-< newTauxDePertesBD
11  if (tauxDePertesBD > 1) { tauxDePertesBD <-< 1 }
12  rdtnoterdt <-< sapply(seq_along(listAgriITKetX), function(i) {
13    ...
14  })
15  rdtReel <-< sapply(seq_along(listAgriITKetX), function(i) {
16    rdtReel <- ...
17    msg <- paste0(
18      "Agri", i,
19      ...
20      "\n"
21    )
22    cat(msg)
23    return(rdtReel)
24  })
25  return(list(tauxDePertesBD, rdtReel))
26 }

```

FIGURE 3.3 – Extrait du script R : fonction de calcul de la production réelle
N.B. Certaines parties du code ont été remplacées par "..." pour alléger l’affichage.

À titre d’exemple, la fonction ci-dessus illustre le cumul des responsabilités. En effet, la fonction `calculRdt` effectue des calculs préliminaires (application d’un bonus ou malus de 10% sur le niveau de pression globale de la mouche et calcul de la production sans pertes) ; il effectue le calcul de la production réelle, met à jour des variables globales, concatène et affiche des messages (méthodes `cat` et `paste0`). Lors du développement, il sera important de séparer les différentes responsabilités, idéalement en créant une fonction ou une méthode par responsabilité, avec des noms simples et précis, ainsi que des paramètres et des valeurs de retour bien définis. Il conviendra également d’éviter les variables globales. En revanche, les variables seront plutôt regroupées au sein de structures de données spécifiques (classes) qui modéliseront des objets liés au jeu. Par exemple, un objet de type `Agriculteur` pourrait avoir la production réelle comme attribut.

L’étape suivante sera de concevoir une architecture et un **refactoring** du modèle qui permet la modularité, en respectant les principes de programmation. Les démarches, choix de conception, et implémentations seront détaillés dans la section suivante concernant le rapport technique.

Chapitre 4

Rapport technique

Le projet peut être divisé en deux grandes étapes. La première partie du développement était consacrée à la réalisation d'une première version prototype du jeu sans GUI en mode **Interface en Ligne de Commande (Command Line Interface) (CLI)**. Une fois ce prototype fonctionnelle, une **Interface Graphique Utilisateur (Graphical User Interface) (GUI)** a été construite par-dessus la mécanique via des interfaces. Finalement, le tout a été empaqueté sous la forme d'exécutables sur les trois principales plateformes (*Windows*, *Linux* et *macOs*). Dans cette section, j'évoque les différents éléments de conception, réalisation, tests et validation de l'application.

4.1 Environnement de développement

La première étape de ma mission était de mettre en place un environnement de travail et de développement adapté à ma mission. Je me suis occupé d'organiser le projet dans un dépôt *GitHub** privé partagé avec mes co-encadrants.

Le développement du projet s'est fait avec la version 3.12.9 de *Python* (version stable à la création du projet) et sur *PyCharm*, un **Environnement de Développement Intégré (Integrated Development Environment) (IDE)** spécifique au développement de grands projets *Python*. Le processus de création de l'environnement virtuel et d'installations des dépendances se fait automatiquement sur *PyCharm*. Néanmoins, un document `README.md` a été rédigé spécifiant les différentes étapes du lancement du projet, de l'installation de *Python* au lancement de l'application en passant par le paramétrage de l'environnement virtuel (*PyCharm*, *VS Code* et installation manuelle), en prenant en compte la distribution utilisée de *Python* (*Anaconda** ou *CPython**).

Comme c'est couramment fait pour les projets *Python*, toutes les dépendances directes et indirectes¹ du projet sont spécifiées dans un fichier `requirements.txt`. Les dépendances directes sont également listées dans un fichier `requirements.in`.

1. Une dépendance indirecte correspond aux dépendances des dépendances directes

4.2 Structure du projet

Le fichier `app.py` à la racine du projet est le point d'entrée de l'application.

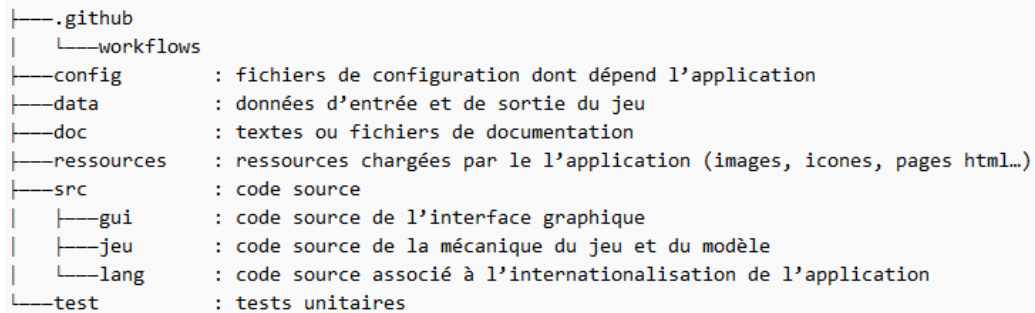


FIGURE 4.1 – Arborescence du projet

4.3 Conception de la mécanique du jeu

Avant de commencer à développer le programme, il a été important de lister les différents éléments qui interagiront avec le programme et de déterminer précisément les types de fichiers d'entrée et de sortie du programme. Par ailleurs, il a fallu identifier clairement les différentes étapes du jeu et les différentes entités qui interagissent avec celui-ci.

Le choix de la stratégie s'est porté vers la **POO**, qui s'est avérée être une approche appropriée pour ce type d'application. La **POO** permet de modéliser les différentes entités et états du jeu à travers des objets représentés par des classes, tout en organisant le code de manière modulaire. Un diagramme de classes **Langage de Modélisation Unifié (Unified Modeling Language) (UML)** a été réalisé à cet effet (cf. Annexe B).

L'entité centrale et principale de l'application est le **Jeu**. L'objet **Jeu** interagit avec les joueurs modélisés par les objets **Agriculteur** et **Acheteur**. La classe **Pratique** modélise les différentes pratiques jouées par les agriculteurs. Elle contient toutes les informations sur une pratique agricole renseignée dans le fichier `csv` d'entrée. Le jeu dispose également d'une boîte à outils qui permet de gérer les chargements des fichiers d'entrée et la génération des fichiers de sortie. Par ailleurs, les objets de type **Transaction** permettent de stocker l'information concernant les transactions entre agriculteurs et acheteurs lors des phases de négociations. L'implémentation du système de transactions ne figure pas parmi les objectifs du cahier des charges d'origine. Il été pensé, puis développé au cours du projet suite à des discussions sur le sujet. Un autre objet qui a été témoin de l'évolution constante du cahier des charges, est le modèle sur lequel est basé le jeu. Ce dernier est modélisé par la classe **Modele** dont les caractéristiques sont détaillées dans la section suivante sur la retranscription du modèle sous **Python**.

4.4 Retranscription du modèle en Python

Le Jeu a pour cœur le modèle. Le modèle est comme une boîte noire qui est appelée une fois par tour de jeu. Il prend en entrée les paramètres du jeu : le niveau de pression de la mouche, le niveau de difficulté et les options activées. Ensuite, il met à jour les variables des agriculteurs (infestation individuelle, production réelle...).

Cette boîte noire est représentée par la classe `Modele` et comprend toutes les méthodes de calculs du modèle scientifique, ainsi qu'une méthode `run_modele` qui sert de boîte noire utilisée par le jeu. Elle regroupe l'ensemble des étapes de calculs du modèle et prend en paramètre le jeu en question.

La première étape du projet consiste en la retranscription et la réfactorisation du modèle en `Python`. Pour cela, il a fallu identifier dans le script `R` les fonctions de calculs liées au modèle et omettre dans un premier temps, toutes les parties liées à la saisie et à la sauvegarde.

Une fois identifiées, les fonctions ont été retranscrites en python et commentées (docstrings²). Le modèle étant le cœur du jeu, et étant sensible à des modifications et des ajustements, il est important de garder une trace claire de son fonctionnement. Lors de cette étape de retranscription, il faut noter les changements prévus liés à la structuration des données.

Par exemple, auparavant, les fonctions utilisaient d'autres fonctions qui retournaient des vecteurs ou des listes de vecteurs. Chacun de ces vecteurs correspondait à une donnée d'un agriculteur. Dans l'extrait du code `R` suivant, le calcul des rendements réels dépend de variables et de fonctions qui sont ou retournent des vecteurs, où l'indice `i` correspond à une donnée de l'agriculteur associé.

```

1  calculRdt <- function() {
2    ...
3    rdtReel <-<- sapply(seq_along(listAgriITKetX), function(i) {
4      rdtReel <- 0.5 * rdtOptimal[i] +
5        ((0.5 * rdtOptimal[i]) * calculTxNoteITKrdt()[i] * (1 -
          calculperteindiv()[i]))
6      ...
7      return(rdtReel)
8    })
9    return(list(tauxDePertesBD, rdtReel))
10 }

```

FIGURE 4.2 – Extrait de la fonction `calculRdt` du script `R`

Dans le code réfactoré, la plupart des fonctions prennent des objets en paramètres, qui contiennent les données nécessaires sous forme d'attributs. En l'occurrence, pour le calcul du rendement, ce-dernier est réalisé pour l'agriculteur fourni en paramètre, et l'accès aux données nécessaires se fait à travers les attributs de l'objet.

2. Indications dans le code sous forme de commentaire formaté, sur le comportement d'une fonction, méthode, classe ou autre élément du code source)

```

1  def calc_rdt(self, agriculteur: Agriculteur) -> float:
2      '''
3      Anciennement sous R : calculRdt
4      Cacule le rendement reel de l'agriculteur entre en parametre
5      '''
6      rdt_reel = 0.5 * agriculteur.rdt_opti + ((0.5 * agriculteur.
7          rdt_opti) * agriculteur.note_rdt * (1 - agriculteur.
8          perte_indiv))
9      if rdt_reel > agriculteur.rdt_opti:
10         rdt_reel = agriculteur.rdt_opti
11     if rdt_reel < 0:
12         rdt_reel = 0
13     return rdt_reel

```

FIGURE 4.3 – Extrait de la fonction `calc_rdt` de la classe (`Modele`)

Par ailleurs, certaines valeurs du modèle du script R étaient codées en dur. À titre d'exemple, il y avait un système de pénalités pour le calcul des scores : lorsqu'une certaine pratique est jouée avec une autre, celle-ci impacte le score de l'autre. L'extrait du code suivant montre le système de pénalité implémenté sur le score mouche de l'agriculteur à travers les blocs conditionnels `if`.

```

1  get_itkmouche_score <- function(monTK){
2      score_adj <- 0
3      if ("ins" %in% monTK & "lbp" %in% monTK) { score_adj <- score_adj
4          - 2}
5      if ("ins" %in% monTK & "lbf" %in% monTK ) { score_adj <-
6          score_adj - 4}
7      if ("ins" %in% monTK & "lbf" %in% monTK & "ins" %in% monTK & "lbp"
8          %in% monTK )
9          { score_adj <- -6}
10     ...
11 }

```

FIGURE 4.4 – Extrait de la fonction `get_itkmouche_score` du script R

Afin de rendre le code plus modulable et de rendre le modèle configurable, toutes ces valeurs qui peuvent faire l'objet de modifications et d'ajustements ont été extraites dans un fichier de configuration. Ce fichier est chargé à l'initialisation du modèle puis stocké en mémoire dans l'attribut `dict_cfg` de la classe `Modele`. En parallèle, ces fichiers de configuration peuvent être partiellement modifiés par l'utilisateur via l'interface graphique.


```

1 mode_penalites: on
2
3 penalites_mouche_pratiques:
4   pen1:
5     pratique_affectee: "lbf"
6     pratique_declencheur: "ins"
7   pen2:
8     pratique_affectee: "lbp"
9     pratique_declencheur: "ins"

```

FIGURE 4.5 – Extrait du fichier de configuration modele_config.yaml

```

1 def get_score_pratique_mouche(self, pratique: Pratique,
2   abbrev_pratiques_realisees: list[str], penalites_mouche: dict):
3   '''
4   Retourne le score de gestion lutte associee a la pratique en
5   appliquant les penalites si besoin
6   :param pratique: la pratique a evaluer
7   :param abbrev_pratiques_realisees: l'ensemble des abreviations des
8   pratiques realisees
9   :param penalites_mouche: dictionnaire contenant l'information sur
10  les penalites liees a la gestion lutte
11  :return: le score de gestion lutte
12  '''
13  score_lutte = pratique.gestion_lutte
14  # application des penalites si besoin
15  if score_lutte > 0:
16    for penalite in penalites_mouche:
17      # verifier si figure dans le dictionnaire des penalites
18      # et si la penalite declencheuse est bien presente
19      if pratique.abbrev == penalites_mouche[penalite]['
20        pratique_affectee'] and penalites_mouche[penalite][
21        'pratique_declencheur'] in abbrev_pratiques_realisees:
22        # appliquer la penalite
23        score_lutte = 0
24  return score_lutte

```

FIGURE 4.6 – Extrait de la fonction get_score_pratique_mouche de la classe Modele

La fonction ci-dessus prend en entrée un dictionnaire de pénalités, correspondant à la configuration. De cette manière le programme vérifie les différentes pénalités possibles qui figurent dans la configuration. Il applique la pénalité si, parmi les configurations, la pratique affectée est jouée avec la pratique déclencheuse. Il a été décidé que la pénalité consiste en l'annulation du score de la pratique affectée.

4.5 Outils du jeu

Les interfaces des outils du jeu sont injectées dans la classe `Jeu` à travers ses attributs. Il s'agit de compositions fortes car seul le jeu s'occupe d'instancier ses propres outils. Le fait de faire dépendre le jeu des interfaces d'outils renforce la modularité. On peut facilement écrire d'autres objets qui implémentent la même interface de l'outil et les remplacer dans le jeu si nécessaire, via des setters ou à travers un fichier de configuration (cas actuel). Entre autres, cette configuration facilite la testabilité de chaque composant et l'isolation des tests unitaires grâce la création d'objets `Mocks*`.

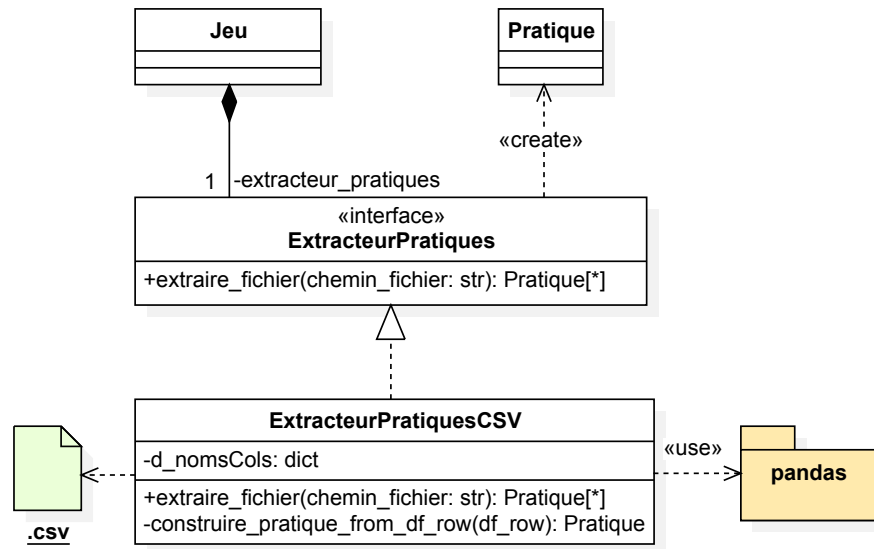
Le fichier de configuration suivant permet de déterminer le type d'objet ou la stratégie appliquée pour chaque outil (pattern stratégie). Il est chargé lors de l'initialisation du jeu pour construire l'objet correspondant.

```
1 extracteur_pratiques :
2   type: csv
3   path: ./data/in/pratiques04032025.csv
4 saver :
5   dir: ./data/out/
6   timestamp: true
7   saver_res_type: xlsx
8   saver_cfg_type: yaml
9   saver_log_type: pythonLogger
```

FIGURE 4.7 – Fichier de configuration `tool_config.yaml`

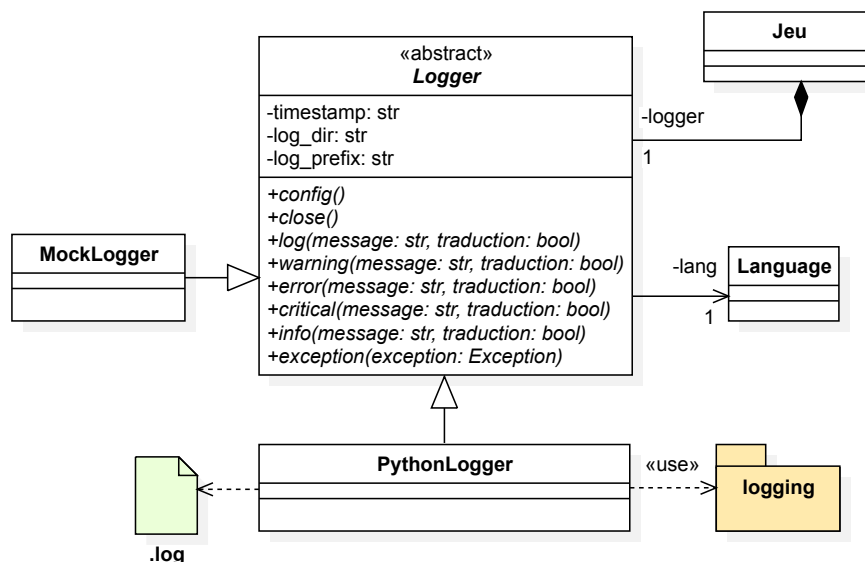
4.5.1 Parser

Parmi les différents outils, le jeu comprend un *parser* (ou extracteur), dont l'objectif est d'extraire les données du fichier d'entrée (jeu de données sur les pratiques agricoles). J'ai à ma disposition ce fichier au format `csv`. La principale fonction du *parser* est d'extraire les données du fichier pour les convertir en un format utilisable par l'application (modèle en mémoire ou classe). Il s'agit dans notre cas, d'une liste d'objets `Pratique`. Ainsi, la classe `ExtracteurPratiqueCSV` permet spécifiquement de charger le fichier de type `csv` et créer les objets `Pratique` associés.

FIGURE 4.8 – Diagramme de classes UML simplifié du *parser*

4.5.2 Logger

Une spécificité renseignée dans le cahier des charges d'origine est d'avoir en sortie une trace du déroulement de la partie. Parmi les fichiers de sortie, un fichier de log, renseignant les différentes phases et actions de la partie devait être généré (nouveau par rapport au script **R**). Pour cela, un objet de type **Logger** est utilisé. La principale fonction du **Logger** est d'écrire un message d'information selon le type d'événement (Information, warning, ou erreur) dans un fichier, dont le chemin et le nom sont définis lors de la construction de l'objet. Le **Logger** actuel du jeu est un **wrapper*** autour du logger du **package*** **logging** de **Python**.

FIGURE 4.9 – Diagramme de classes UML simplifié du *logger*

```

[2025-03-23 21:25:31,605] [ INFO] → Initialisation de PythonLogger
[2025-03-23 21:25:31,610] [ INFO] → DEBUT JEU
[2025-03-23 21:25:31,610] [ INFO] → Initialisation du jeu...
[2025-03-23 21:25:31,610] [ INFO] → SET nb_agriculteurs : 6
[2025-03-23 21:25:31,610] [ INFO] → SET nb_banabanas : 1
[2025-03-23 21:25:31,610] [ INFO] → SET nb_exportateurs : 1
[2025-03-23 21:25:31,610] [ INFO] → SET sauvegarde : False
[2025-03-23 21:25:31,611] [ INFO] → SET mode transactions : False
[2025-03-23 21:25:31,611] [ INFO] → SET option spatialisation : True
[2025-03-23 21:25:31,611] [ INFO] → SET option variétés : False
[2025-03-23 21:25:31,611] [ INFO] → SET niveau difficulté : 3
[2025-03-23 21:25:31,611] [ INFO] → SET niveau pression mouche : 0.4
[2025-03-23 21:25:31,611] [ INFO] → Fin initialisation
[2025-03-23 21:25:31,611] [ INFO] → DEBUT DU TOUR 1
[2025-03-23 21:25:31,611] [ INFO] → Début phase de choix des pratiques agricoles...
[2025-03-23 21:25:31,823] [ INFO] → La pratique lbp a été désactivée.
[2025-03-23 21:25:31,842] [ INFO] → La pratique lbe a été désactivée.
[2025-03-23 21:25:31,860] [ INFO] → La pratique lbf a été désactivée.
[2025-03-23 21:25:39,906] [ INFO] → Le joueur agriculteur 1 a joué les cartes ['irr', 'irr', 'ins', 'fer'] pour 21.0 jeton(s)
[2025-03-23 21:25:39,907] [ INFO] → Le joueur agriculteur 2 a joué les cartes ['irr', 'irr', 'fer'] pour 6.0 jeton(s)
[2025-03-23 21:25:39,908] [ INFO] → Le joueur agriculteur 3 a joué les cartes [] pour 0 jeton(s)
[2025-03-23 21:25:39,910] [ INFO] → Le joueur agriculteur 4 a joué les cartes [] pour 0 jeton(s)
[2025-03-23 21:25:39,912] [ INFO] → Le joueur agriculteur 5 a joué les cartes [] pour 0 jeton(s)
[2025-03-23 21:25:39,913] [ INFO] → Le joueur agriculteur 6 a joué les cartes [] pour 0 jeton(s)
[2025-03-23 21:25:39,913] [ INFO] → Fin phase de choix des pratiques agricoles
[2025-03-23 21:25:39,914] [ INFO] → Calculs des résultats du tour 1
[2025-03-23 21:25:41,933] [ INFO] → FIN DU TOUR 1
[2025-03-23 21:25:41,935] [ INFO] → Finalisation du jeu...
[2025-03-23 21:25:42,567] [ INFO] → FIN DU JEU

```

FIGURE 4.10 – Exemple de fichier de logs généré

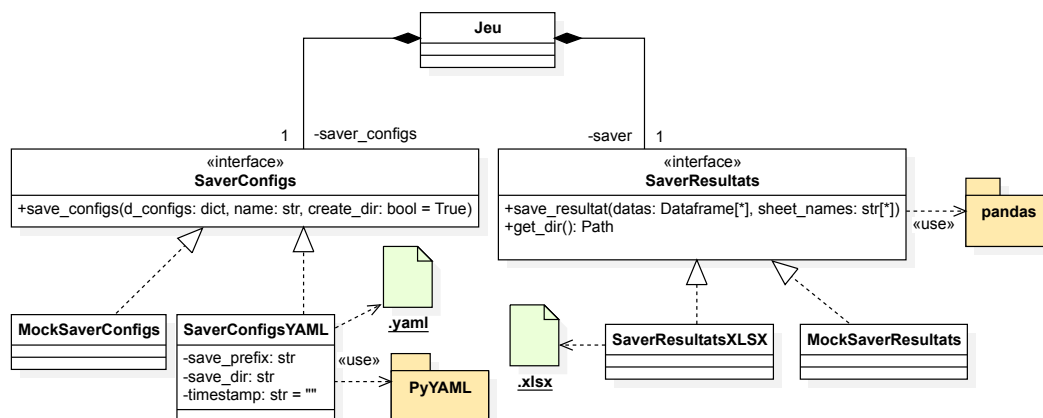
4.5.3 Saver

Afin de permettre la sauvegarde des données générées par le jeu dans des fichiers de sortie, le jeu utilise des objets de type **Saver**. Ces objets ont pour fonction de convertir les données stockées en mémoire dans la classe **Jeu**, sous la forme de dictionnaires ou de **dataframe** (avec la librairie **pandas**), en un fichier de sortie de format approprié contenant ces données.

L'application se sert de deux types de **Saver** :

- **SaverResultats** : conversion des données de type **dataframe** en feuille de calcul, avec une page par **dataframe**.
- **SaverConfigs** : conversion des dictionnaires de configuration en un fichier de configuration de sortie

Pour enregistrer les données sous forme de **dataframe**, des classes utilitaires ont été utilisées afin de pouvoir faciliter la traduction des entêtes de la feuille de calcul générée (**DataResultsHeader** et **DataResultsTransaction**). Chaque attribut de cette classe correspond à la chaîne de caractère traduite d'une colonne du tableur, et l'attribut **datas** est un dictionnaire qui associe le titre à la valeur de la colonne. Ce dictionnaire sert à stocker temporairement les différentes valeurs d'une ligne d'un **dataframe**. Ce dernier est mis à jour lors de la création de chaque ligne d'un **dataframe**.

FIGURE 4.11 – Diagramme de classes UML simplifié des *savers*

Les graphiques de sorties sont créés en utilisant le module `matplotlib` et les données de sauvegarde du jeu (`dataframe`). Leurs créations sont renseignées dans un fichier `graphique.py`. Les fonctions de création d'un graphique prennent en paramètre un `dataframe`, et renvoient un objet de type `Figure` de `matplotlib`. Le jeu se charge d'appeler ces fonctions en fin de jeu et de stocker en mémoire ces objets. Cela permet l'affichage de graphiques interactifs avec le `GUI` et la sauvegarde de ces derniers via la méthode `savefig` proposée par la classe `Figure`.

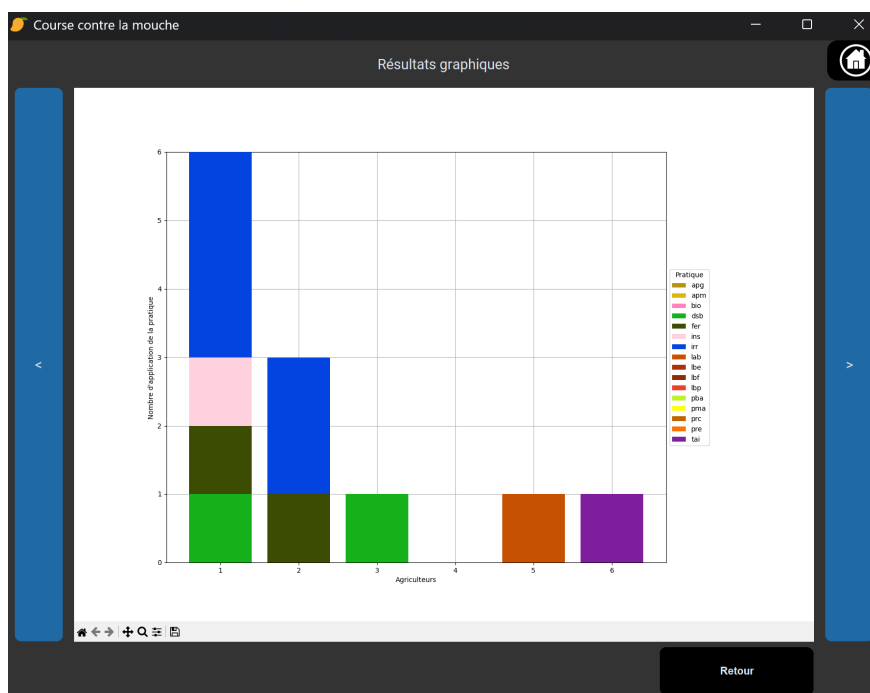


FIGURE 4.12 – Exemple d'un graphique de sortie interactif affiché sur le GUI

4.5.4 Settings

Une classe nommé **Settings** permet de spécifier les chemins de tous les fichiers de configuration utilisée. Pour chaque fichier de configuration, il existe un fichier par défaut, qui permet à l'utilisateur de revenir aux configurations de base. En effet, lors de l'initialisation du jeu, l'utilisateur configure les paramètres du jeu qui seront enregistrés dans le fichier de configuration associé (`jeu_config.yaml`). Cela permet, la sauvegarde des derniers paramètres utilisés lors du lancement de la prochaine instantiation du Jeu (nouvelle partie). Une option sur le **GUI** permet à l'utilisateur de revenir aux configurations de base. Le code s'occupe de copier le contenu du fichier par défaut (`jeu_config_default.yaml`) dans le fichier actuel et de charger ce-dernier. Le choix du format s'explique de par sa lisibilité et de par la possibilité d'ajouter des commentaires.

Les différents fichiers de configuration utilisés par l'application sont :

- `jeu_config.yaml` : configuration des paramètres du jeu, modifiable et chargé lors de l'initialisation du jeu.
- `modele_config.yaml` : configuration des variables du modèle, chargé lors de l'instanciation de l'objet `Modele` par le jeu durant son initialisation. Certaines valeurs de ce fichier sont modifiables par l'utilisateur du **GUI** à travers la rubrique "Paramètres avancés".
- `tool_config.yaml` : configuration des outils du jeu chargé lors de l'initialisation du jeu. Seul le chemin du répertoire de sauvegarde y est modifiable par l'utilisateur
- `setup_yaml` : configuration de toutes les constantes du jeu chargé lors de l'instanciation de la classe `Jeu`



FIGURE 4.13 – Interface de modification des paramètres avancés du GUI

4.6 Internationalisation

Une des fonctionnalités demandées par ma mission, est la capacité à traduire le contenu textuel des fichiers de sorties et des éléments affichés par l'interface graphique. Pour cela, l'objet `Language` permet la traduction des chaînes de caractères renseignées dans le code source dans la langue souhaitée, à travers un dictionnaire de langue. En effet, la classe `Language` est une énumération comprenant une valeur pour chaque instance, qui est un dictionnaire associé à la langue et stocké dans un fichier nommé `[abréviation de la langue]_dict.py`.

```
1 from enum import Enum
2 from . import fr_dict, en_dict, jp_dict
3
4 ...
5
6 class Language(DictEnum):
7     FR = fr_dict.fr
8     EN = en_dict.en
9     JP = jp_dict.jp
10
11     def get_text(self, original_text: str) -> str:
12         return self.value.get(original_text) or "Translation_not_found"
```

FIGURE 4.14 – Implémentation de la classe `Language`

```
1 en = {
2     # GUI
3     "Course_contre_la_mouche": "Race_against_the_fly",
4     ...
5     # GUI – Choix cartes
6     "Agriculteur" : "Farmer",
7     "Joueur" : "Player",
8     "Choix_des_cartes" : "Card_choices",
9     ...
10 }
```

FIGURE 4.15 – Extrait du dictionnaire de langue dans `en_dict.py`

La méthode `get_text` de la classe `Language` permet de récupérer la traduction de la chaîne de caractères entrée en paramètre si celle-ci est bien renseignée dans le dictionnaire associé. Les dictionnaires de langue doivent tous avoir les mêmes clés, correspondant aux chaînes de caractères dans le code source. Les valeurs associées aux clés correspondent à la traduction affichée dans les fichiers de sorties et dans le `GUI`. Cet objet est injecté dans le jeu et le `GUI`, ainsi que dans tous les objets qui nécessitent une traduction, comme le `Logger`. Le changement de langue s'opère avant l'instanciation du jeu en changeant l'attribut `lang` de la classe `Gui` (cf. section 4.8). Lorsque le jeu est instancié (e.g. click sur « Démarrer le jeu ») la langue du `GUI` va être injectée dans le constructeur de l'objet `Jeu` instancié par la classe `Gui`, ainsi que dans tous les objets qui en dépendent.

```
1 self.title(self.__lang.get_text("Course_contre_la_mouche"))
```

FIGURE 4.16 – Exemple d'utilisation de la méthode `get_text`

N.B. Un fichier commenté de *template** d'un dictionnaire de langue et un document de tutoriel ont été créés pour faciliter le processus d'ajout ou suppression d'une langue (cf. Annexe E).

4.7 Identification des différents états du jeu

Un des objectifs de ma mission est de fluidifier le processus décrit précédemment dans l'étape d'analyse (cf. Figure 3.1). Pour cela, il a fallu clairement identifier les différentes étapes du jeu, et surtout les différents états par lesquels va évoluer l'objet `Jeu` du programme.

L'informatisation du jeu comprend initialement le développement des parties 1, 2 et 3 du jeu, spécifiées dans le poster en annexe (cf. Annexe C). Cela peut se traduire par les états suivants, par lesquels l'objet `Jeu` va évoluer :

- A. **EtatInitialisation** : l'animateur saisit les conditions initiales du jeu
 - Initialiser les outils du jeu
 - Récupérer les conditions initiales du jeu saisies par l'utilisateur
 - Charger les configurations du jeu
- B. **EtatPhaseChoixPratiques** : l'animateur saisit les pratiques jouées par les agriculteurs qui servent d'entrée au modèle
 - Récupérer les pratiques saisies par l'utilisateur
- C. **EtatFinalisationTour** : l'application affiche les données générées par le modèle
 - Faire tourner la boîte noire du modèle
 - Enregistrer en mémoire (*dataframe*) les données relatives au tour, qui serviront pour la construction des données de sortie
- D. **EtatInitiliasationTour** : l'animateur initialise un nouveau tour
 - Récupérer les conditions initiales du tour saisies par l'utilisateur
- E. **EtatFinalisation** : l'application affiche et enregistre les données de sorties
 - Générer les graphiques de sorties à partir des données de sauvegarde (en mémoire) du jeu
 - Sauvegarder les données de sorties et les graphiques
 - Fermer les ressources utilisées par la classe `Jeu`

Le jeu est une boucle des étapes B, C et D jusqu'à ce que l'animateur décide de passer à l'état E depuis l'état D.

L'évolution constante du cahier des charges au cours de ma mission, a impliqué par la suite l'informatisation des étapes 4 et 5 spécifiées dans le poster en annexe (cf. Annexe C). L'informatisation de ces étapes comprend l'enregistrement et la sauvegarde par le jeu des transactions entre acheteurs et producteurs via une saisie utilisateur (e.g. l'animateur saisie que l'acheteur 1 à acheté x tonnes à l'agriculteur 1). Une vérification est réalisée sur le tonnage acheté dépendant des résultats du tour : un acheteur ne peut

pas acheter une quantité de mangues supérieure à ce que l'agriculteur a produit durant le tour. Les données des transactions stockées en mémoire dans les objets **Transaction** sont ensuite sauvegardées dans le fichier de sortie **xlsx** sous forme de tableau. Cette étape a impliqué l'ajout d'une nouvelle classe **Transaction** et les états du jeu suivants :

- F. **EtatPhaseNegociation** : l'animateur renseigne les données des transactions
 - Créer les objets transactions suite à la saisie de l'utilisateur
 - Mettre à jour sa liste de transaction
- G. **EtatLancerDes** : l'application affiche l'ensemble des transactions du tour et l'animateur coche une case sur la transaction pour indiquer si le lot a été perdu (déterminé en physique par un lancer de dés)
 - Mettre à jour l'attribut **est_perdu** des transactions suite à la saisie de l'utilisateur

Un diagramme états-transitions a été réalisé retraçant l'évolution de l'objet **Jeu** au cours d'une partie (cf. Annexe B).

Pour l'implémentation de ces différents états, nous utilisons le pattern de conception État. Ce pattern permet d'attribuer un comportement différent à un même objet en fonction de son état. Le jeu est une succession de plusieurs phases et adopte un comportement différent à chaque phase.

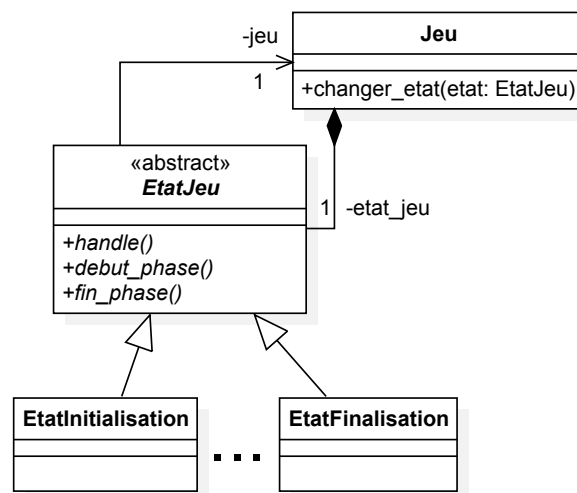


FIGURE 4.17 – Diagramme de classes UML simplifié des états du jeu

Un jeu ne possède qu'un seul état à la fois. Un objet **Etat** implémente deux méthodes : **debut_phase** et **fin_phase**, qui permettent d'exécuter du code relatif à l'état du jeu. Le jeu peut changer d'état grâce à la fonction **changer_etat(etat Etat)** de la classe **Jeu**. La finalisation d'un état (sauf pour l'**EtatFinalisation**) fait appel à cette méthode pour passer à l'état suivant.

À première vue, la classe **Etat** comprenait seulement une fonction **handle**, qui permettait d'exécuter le code relatif à cet état. Cette solution semblait fonctionner parfaitement avec la version de l'application en mode **CLI**, où chaque action se suit l'une après l'autre, et surtout avec le fait que la demande de saisie utilisateur mettait en pause le programme. Cependant, il a fallu séparer ce qui est exécuté au début et ce qui est exécuté à la fin de la phase pour la version **GUI**, qui a besoin de cette distinction pour mettre en pause le programme.

4.8 Conception GUI

4.8.1 Libraries graphiques

L'application ne devant pas utiliser des modules ou des librairies soumis à des licences commerciales ou à des licences imposant de publier le code source lors d'une distribution, une analyse succincte des différentes librairies graphiques **Python** a été réalisée.

Librairie	Licence	Avantages	Inconvénients
PyQt	GPL / Commercial	Grande customisation Programmation POO Grande variété de widgets Application pour placer les widgets (<i>Qt Designer</i>) Performant	Taille lourde Complexe
Tkinter	Standard Python Licence (libre)	Léger Facile d'utilisation Intégré dans la distribution Python	Uniquement desktop Peu de widgets Peu de customisation
CustomTkinter	MIT (libre)	Style moderne Programmation POO Customisation décente	Uniquement desktop
Kivy	MIT (libre)	Grande customisation Beaucoup de widgets et fonctionnalités Support mobile et touchscreen Performant	Complexité
PySimpleGUI	LGPL	Tres simple d'utilisation	Limité en complexité Peu de customisation Peu maintenu Desktop only
PySide	LGPL	Grande customisation Programmation POO Grande variétés de widgets Application pour placer les widgets (<i>Qt Designer</i>)	Taille lourde Complexité

FIGURE 4.18 – Tableau récapitulatif des différentes librairies graphiques Python

Le choix s'est porté sur *CustomTkinter*, une librairie construite par-dessus *Tkinter*, pour avoir un compromis entre ergonomie, modernité et simplicité. L'utilisation

de *Tkinter* implique que tout soit codé en mode procédural. Le builder drag-and-drop le plus connu est *TkinterDesigner*, mais ce dernier n'utilise pas de fichier XML* ou de langage markup similaire pour définir la disposition des composants³. Il se contente de générer le code Python nécessaire à partir d'un template *Figma**. Par ailleurs, *TkinterDesigner* n'utilise pas les composants de *CustomTkinter* et devient plus complexe à utiliser pour des interfaces nécessitant plusieurs fenêtres.

4.8.2 Prototype *Figma*

Afin d'avoir une idée générale de la disposition des différents types de *vue** et des composants, un premier prototype a été réalisé sur *Figma*. Une discussion a eu lieu autour de ce prototype avec l'équipe de chercheurs du projet. Le développement de l'interface a commencé après la prise en compte des points abordés, les changements demandés et la validation des différentes dispositions du prototype (cf. Annexe D).

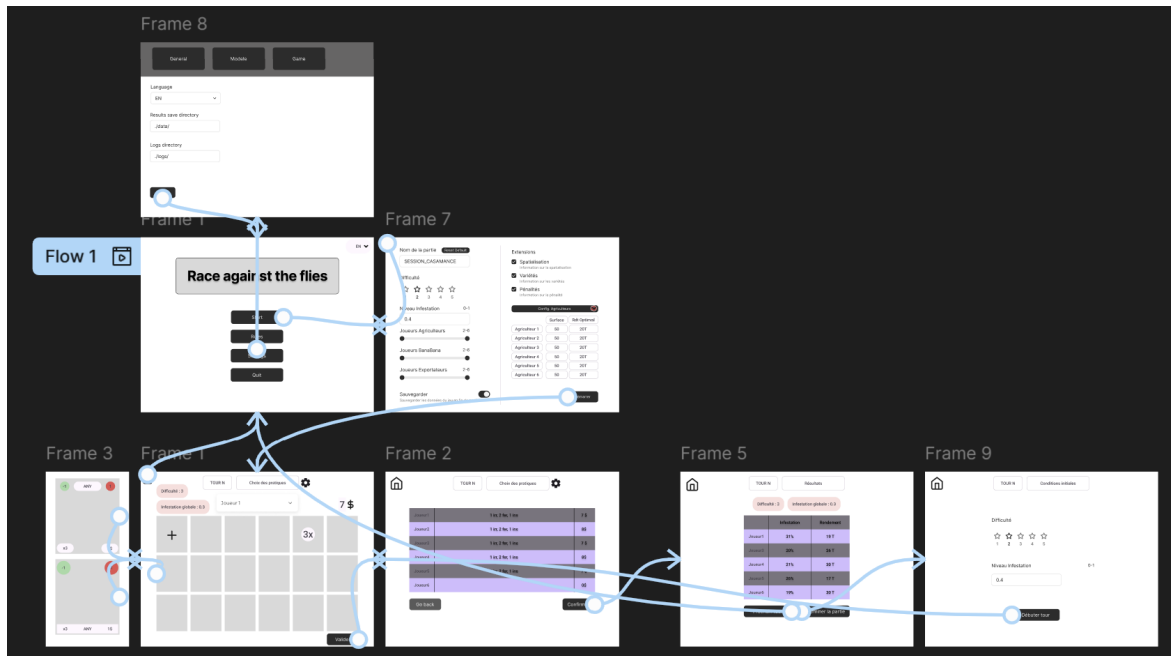


FIGURE 4.19 – Prototypage sur Figma, gestion des transitions et actions utilisateurs

3. Les frameworks *Qt* utilisent des fichiers XML avec *Qt Designer*, et *Kivy* sépare la logique de la disposition graphique à travers les fichiers *.kv*.

4.8.3 Conception et organisation des vues

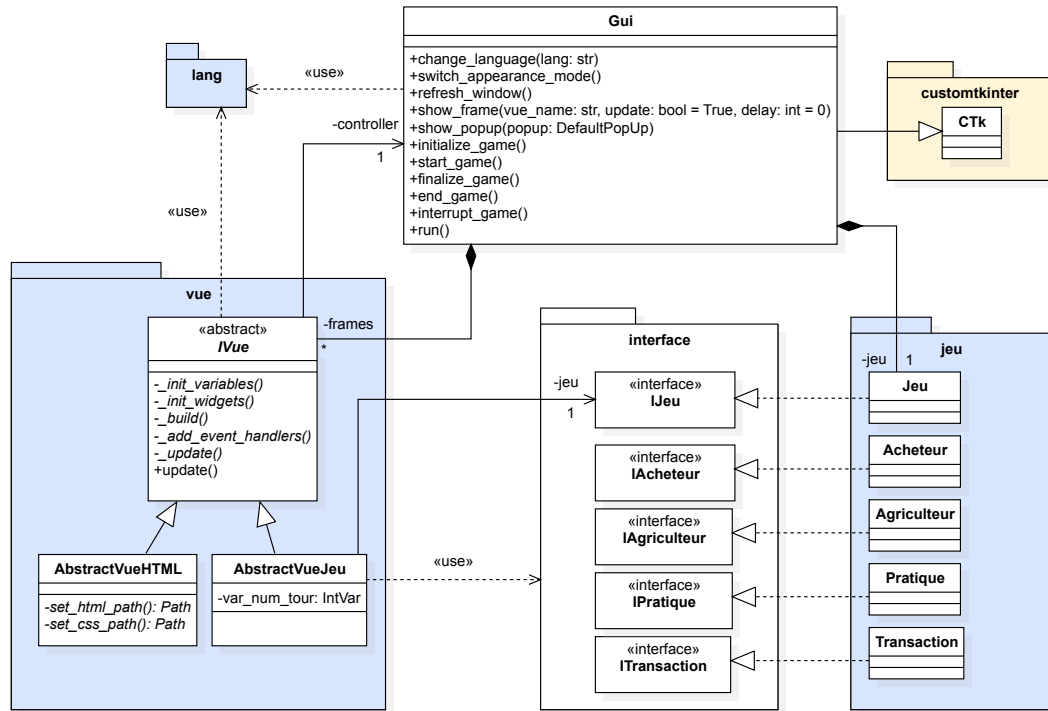


FIGURE 4.20 – Diagramme de classes UML simplifié de l'application

Le **GUI** communique avec la mécanique du jeu par le biais d'objets de type **interface*** qui explicitent les méthodes nécessaires à son fonctionnement (actions et demandes utilisateur). Les objets principaux de la mécanique implémentent directement l'interface associée. L'affichage est assurée par les classes **vues**.

La conception de l'interface graphique fait intervenir beaucoup d'héritage. En effet, sachant qu'une vue ou scène possède souvent la même forme, le même style de base, et peut conserver les mêmes attributs ou comportements qu'une scène de base avec quelques différences, l'héritage semble être une approche judicieuse. Par ailleurs, certains composants sont des versions spécifiques d'un certain **widget*** de base. L'héritage est aussi intéressant à ce niveau-là⁴. Dans l'exemple ci-dessous, la classe `DefaultButton` hérite de `CTkButton` en appliquant un style personnalisé. Certaines propriétés du **widget** de base proposé par la librairie `customtkinter` sont redéfinies. Quant à son parent, celui qui le construit, il est spécifié dans les paramètres du constructeur lors de son initialisation.

4. Il faut différencier le composant qui s'occupe de construire le **widget** (parent ou master) et l'objet dont hérite le **widget**.

```

1 class DefaultButton(CTkButton):
2     def __init__(self, parent, text: str = "Button", command: Callable |
      None = None, *args, **kwargs):
3         super().__init__(parent, text=text, command=command, *args, **
      kwargs)
4         self.style: Style = style
5         self.configure(
6             font=(self.style.BUTTONS_FONT, self.style.BUTTONS_FONT_SIZE,
      self.style.BUTTONS_FONT_WEIGHT),
7         )

```

FIGURE 4.21 – Exemple d'un widget réutilisable par le GUI : bouton par défaut

La fenêtre principale (*root*), représenté par la classe `Gui`, hérite directement de l'objet `CTk` de `customtkinter`, permettant d'initialiser une fenêtre. C'est principalement cette fenêtre qui va contenir l'ensemble des scènes ou *vues* du `GUI`, à l'exception des *pop-ups** qui sont des fenêtres à part mais qui seront créées par le composant *root*. La classe `Gui` a pour responsabilité d'assurer la création et la destruction des *vues*, l'instanciation de l'objet `Jeu`, et d'assurer la transition entre les *vues*, ainsi que de synchroniser ces transitions avec l'évolution des états du jeu.

Les différentes *vues* sont organisées dans le *package* `vue`. Toutes les *vues* héritent d'une classe abstraite `IVue` qui détermine les méthodes d'initialisation de la vue, appelées dans cet ordre respectif :

1. `_init_variables()` : pour initialiser toutes les variables de la vue, typiquement des objets de types `Variable` (objet `tkinter`) qui sont liés à des *widgets* pour mettre à jour leurs contenus textuels ou leur état.
2. `_init_widgets()` : permet de déclarer l'ensemble des *widgets* importants, nécessitant des *bindings* ou une gestion d'événement (faisant l'objet d'une interaction avec l'utilisateur). Ils sont souvent initialisés à `None` (*lazy loading*) et instanciés puis construits uniquement par la méthode `_build()`.
3. `_build()` : permet de construire et placer les *widgets* sur la vue, en utilisant les trois méthodes de placement disponibles de `tkinter` (`place`, `grid`, ou `pack`). Dans notre cas, `grid` est souvent utilisé pour la disposition générale de la vue sous forme de grille, `pack` pour empaqueter des *widgets* dans un conteneur (souvent des *widgets* de type `CTkFrame`), et `place` est rarement utilisé mais sert à placer un *widget* relativement à son parent (celui qui le construit).
4. `_add_event_handlers()` : permet d'ajouter les événements associés aux *widgets* ou de créer des *bindings* qui vont déclencher un événement.
5. `_update()` (appel optionnel selon les paramètres renseignés par la méthode `show_frame` de la classe `Gui`) : permet de mettre à jour les variables et les *widgets* si nécessaire.

Le `GUI` est constitué de 3 types de *vues*. Les *vues* de base, qui correspondent aux *vues* qui héritent directement de `IVue` et qui sont affichées en dehors du jeu (section "Menu" et "Paramètres"). Les *vues* du jeu, qui héritent de `AbstractVueJeu`, vont contenir l'instance du jeu (`IJeu`). Ce sont toutes les *vues* affichées lors du déroulement

du jeu, de la création de l'instance du jeu à sa destruction. Ensuite, les **vues HTML***, qui héritent de **AbstractVueHTML**, s'occupent de charger une page **HTML** stockée en local (avec possibilité de renseigner une feuille de style) via la librairie **tkinterweb**. Ces **vues** sont consacrées aux parties documentation (sections "Règles du jeu" et "A propos").

4.8.4 Responsive

Les méthodes **set_windows_scaling** et **set_widget_scaling** proposées par *customtkinter* permettent d'agrandir la fenêtre et les composants du **GUI**. Or, sous *Windows* et *macOS*, cela est fait automatiquement par *customtkinter* en fonction des configurations de zoom de l'utilisateur, sauf en cas de désactivation en spécifiant la méthode **deactivate_automatic_dpi_awareness**. Le cas à traiter est *Linux*. Le facteur d'agrandissement a été défini en fonction du **DPI*** actuel de l'utilisateur et du **DPI** de base (zoom à 100%). Cette valeur de base est fixée à 96, valeur courante pour la plupart des machines actuelles.

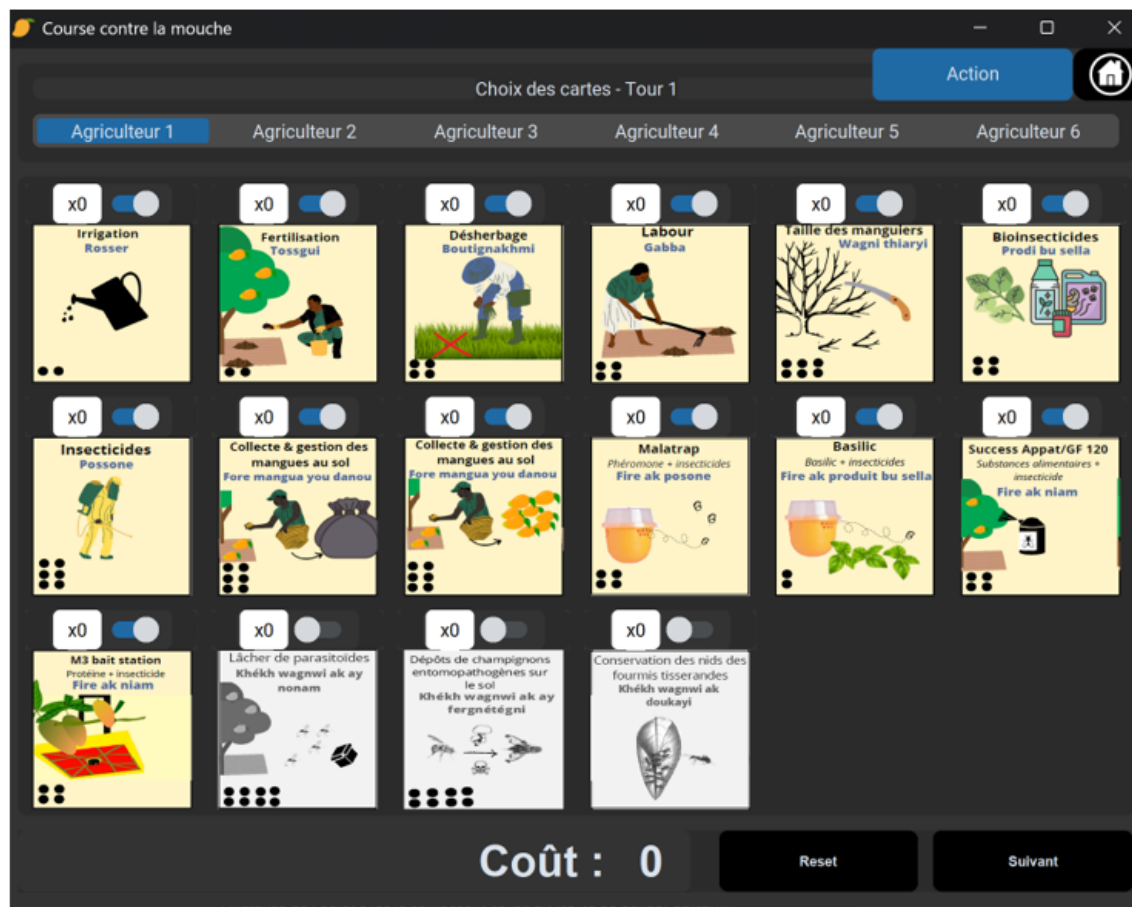
La gestion du **responsive*** est principalement associée aux options utilisées lors du placement des **widgets** avec les fonctions **grid**, **place**, ou **pack**.

Pour **grid**, l'option **sticky='nsew'** permet de spécifier au **GUI** que le **widget** reste collé au bord nord, sud, est et ouest de son parent, peu importe la taille de la fenêtre. Pour **pack**, l'option **fill='both'** ou ('x' ou 'y') permet au **widget** d'occuper tout l'espace disponible dans une direction (horizontalement, verticalement, ou les deux). L'option **expand=True** permet d'étendre l'espace d'occupation du **widget** en fonction de ce qui est disponible au parent.

Pour **place**, les options **relwidth=[entre 0 et 1]**, **relheight=[entre 0 et 1]**, permettent de spécifier la taille du **widget** en fonction de son parent.

Ultérieurement, il a parfois été utile d'utiliser un *binding* avec la séquence **<Configure>** pour mettre à jour un composant lorsque son parent change de taille, comme pour l'affichage des cartes illustrés ci-dessous (cf. Figure 4.22).

Mode fenêtré :



Mode plein écran :

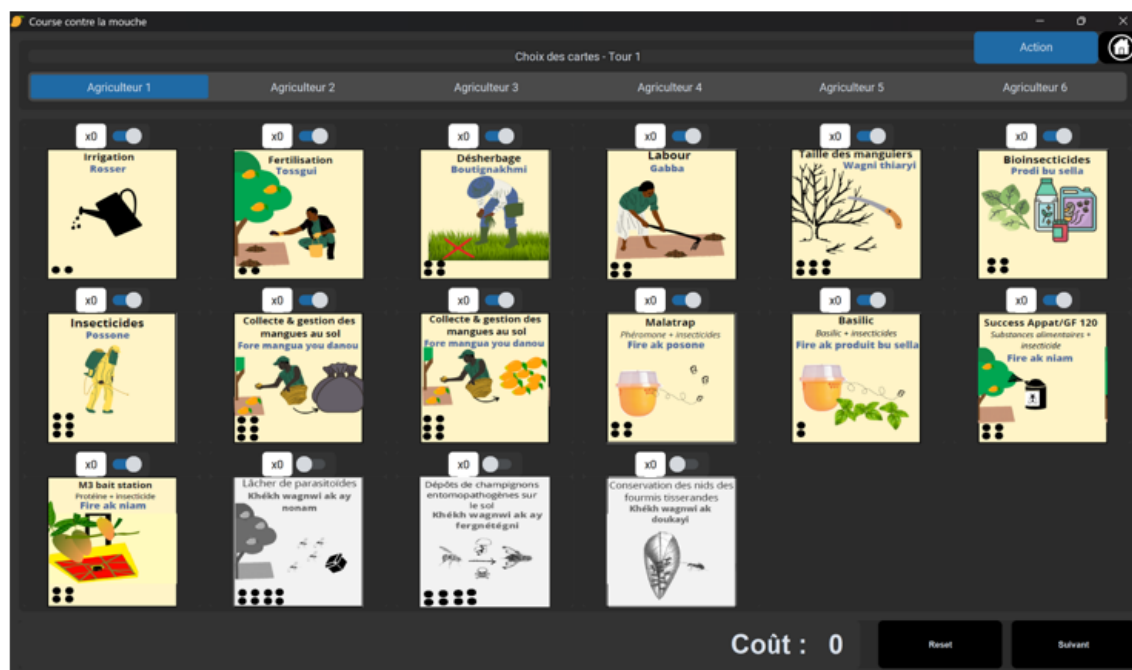


FIGURE 4.22 – Redimensionnement de l’affichage des cartes (2880x1800, zoom 225%)

4.8.5 Widgets et styles

L'interface graphique utilise principalement les composants de *customtkinter*. Certains composants de l'application sont des regroupements de plusieurs *widgets* de base. Ces *widgets* réutilisables sont définis dans un fichier `defaultwidgets.py`. Certains de ces composants peuvent être construits différemment lors de leur création, en fonction des appels de fonctions réalisés. Cette implémentation repose sur l'appel de méthodes en cascade, qui retournent l'instance de l'objet courant, en y ajoutant le composant construit (ce qui s'apparente à un *builder pattern*).

```

1  class DefaultFrame(CTkFrame):
2      def __init__(self, parent):
3          super().__init__(parent)
4          self.style : Style = style
5          self.header_frame : CTkFrame|None = None
6          self.label : DefaultLabel|None = None
7          self.indicator : DefaultLabel|None = None
8          self.info_button : DefaultInfoButton|None = None
9          self.entry : CTkEntry|None = None
10         self.configure(bg_color="transparent", fg_color="transparent")
11
12         def with_label(self, label_text: str = "Label"):
13             self._create_header_frame()
14             if self.label is None:
15                 self.label = DefaultLabel(self.header_frame, text=label_text)
16                 self.label.pack(side="left")
17             return self
18
19         def with_indicator(self, label_text: str = "Label", textvariable=None):
20             self._create_header_frame()
21             if self.indicator is None:
22                 self.indicator = DefaultLabel(self.header_frame, text=
23                     label_text, textvariable=textvariable)
24                 self.indicator.pack(side="right")
25             return self
26         ...

```

FIGURE 4.23 – Exemple d'un widget utilisant l'appel de méthodes en cascade

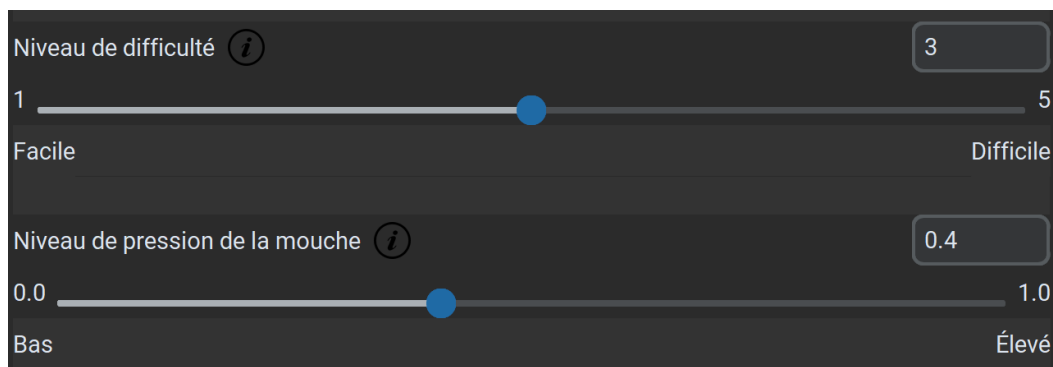


FIGURE 4.24 – Composants réutilisés : `DefaultFrameSlider(DefaultFrame)`

Certains composants plus complexes constitués d'une agrégation de **widgets** simples, sont spécifiques à une vue, comme par exemple le tableau récapitulatif des choix de pratiques, ci-dessous. Ces composants sont directement définis dans la vue associée.



FIGURE 4.25 – Composant complexe : RecapFrameTableMatrice

L'équipe de chercheurs a l'intention de faire appel à un prestataire extérieur pour packager le jeu (version physique et stylisation de la version informatique). En conséquence, les éléments graphiques, tels que les images, ainsi que leur format (couleur, texte, police, etc.) doivent être facilement modifiables et remplaçables. En ce qui concerne les **vues** de documentation (sections "A propos" et "Règles du jeu"), elles doivent également être facilement modifiables (texte, images, et disposition) tout en offrant un degré de personnalisation décent. Mon objectif est donc de mettre en place un système qui permet le remplacement simplifié des éléments graphiques du **GUI** et du style, sans avoir à modifier la construction des scènes dans le code, tout en respectant un certain degré de personnalisabilité.

Concernant les **vues** de documentation, la première solution envisagée pour leurs implémentations était d'utiliser une librairie qui permet de lire et afficher un fichier **Portable Document Format (PDF)**. Cependant, toutes les librairies fonctionnelles et maintenables de **Python** sont soumises à des licences permissives. Elles utilisent quasi tous la librairie **PyMuPdf** sous licence **GPL**. En revanche, il existe une librairie libre nommé **PyPDF2** qui permet la manipulation des fichiers **PDF**. Néanmoins, l'utilisation de cette dernière, implique la conversion de chaque élément lu, dans des **widgets** appropriés (text, images, etc.); et de coder un lecteur **PDF** fonctionnel pour **tkinter**. En prenant en considération le temps du projet et les priorités, ce choix-là n'a pas été retenu.

La solution choisie est donc le format **HTML** et **CSS***. Il répond aux critères demandés, sans pour autant avoir besoin de modifier le code source **Python**. Il suffit de changer les pages **HTML** et les feuilles de style **CSS** selon les besoins. Cependant, quelques contraintes sont à respecter pour assurer le bon fonctionnement. Un document a été rédigé spécifiant ces conditions à respecter lors de la modification de ces pages **HTML** locales : mêmes noms de fichier, renseigner les liens relatifs à la racine du projet, balises supportées, etc.

En effet, pour assurer que ces fichiers soient chargés correctement avec les images et les feuilles de style, des fonctions ont été écrites pour injecter dynamiquement les

bons chemins des fichiers au sein des fichiers **HTML**. Le moteur web de **tkinterweb** a besoin d'utiliser le protocole `file://` et des chemins absolus pour accéder correctement à des fichiers locaux de la machine. Ainsi, lors de l'exécution, le code vient convertir les chemins relatifs en chemins absolus, précédés du préfixe `file://` ou `file:///` en fonction de la plateforme. Cette méthode fonctionne indépendamment de l'utilisateur et vient remplacer le chemin absolu à chaque exécution (cas où l'utilisateur change le répertoire de l'application). Cette implémentation est réalisée grâce à la librairie **Beautiful Soup** qui nous octroie un parser **HTML**.

Le thème de base du **GUI** est chargé via un fichier de configuration `app_theme.json`. Dans ce fichier, concernant les couleurs des composants, deux valeurs sont renseignées correspondant respectivement au mode clair et au mode sombre de l'application : `[couleur_light, couleur_dark]`. Ce fichier définit les propriétés des **widgets** de base du module `customtkinter`. Certains composants de l'application, étant des regroupements de plusieurs **widgets**, sont configurés différemment. Leurs propriétés de base sont alors écrasées via l'injection d'une classe **Style** définie dans le fichier `style.py`.

N.B. Un document de tutoriel a été rédigé spécifiant les modalités de modification des différents éléments facilement modifiables du **GUI** (cf. Annexe **F**).

4.9 Packaging

Un des éléments du cahier des charges est que l'application soit packagée sous la forme d'un exécutable sur les trois principales plateformes (*Windows*, *macOS* et *Linux*), avec *Windows* comme priorité (plateforme la plus utilisée par les utilisateurs). Pour cela, il faut compiler le projet en code binaire sur les trois plateformes, en tenant compte des différences entre les différentes distributions *Linux*, qui peuvent ne pas être compatibles au niveau binaire, ainsi que des architectures de machines.

Parmi les options de compilation multiplateforme possibles (une même bibliothèque permettant la compilation sur les différentes plateformes), les principales retenues sont : *PyInstaller*, *cx_Freeze* et *Nuitka*. Des difficultés ont été rencontrées lors des tests de compilation avec *PyInstaller* et *cx_Freeze* sous *Windows*. En effet, l'exécutable généré était considéré comme un logiciel malveillant par les antivirus. Même s'il s'agit d'un faux positif, cela reste un inconvénient pour l'utilisateur sous *Windows*, qui devrait désactiver son antivirus pour lancer l'application.

Le choix s'est donc tourné vers *Nuitka*, un outil qui compile le code source **Python** en code source *C* optimisé, avant de le convertir en exécutable binaire.

Pour faciliter cette étape, un script **Python** a été codé pour générer la commande *Nuitka* à exécuter pour compiler le projet, en fonction du système d'exploitation. En effet, la commande s'avère complexe en raison des fichiers de données externes utilisés par l'application (config, ressources, etc.), mais aussi d'une complication avec le module **tkinterweb** utilisé par l'application. Plus de détails sur le processus, le choix et les démarches figurent dans un document `package.md` rédigé, qui explique les modalités de packaging de l'application avec *Nuitka* (cf. Annexe **G**).

Lors de la compilation, deux options sont disponibles :

- Compilation en un seul fichier exécutable (version *portable*). Néanmoins, cela entraîne un temps de lancement plus long et empêche la persistance des données entre les exécutions de l'application, du fait que les fichiers de configuration générés au lancement sont stockés dans un répertoire temporaire.
- Compilation en un dossier contenant l'exécutable et ses fichiers nécessaires (version *standalone*). Cette configuration permet un lancement nettement plus rapide, mais nécessite (recommandé) une étape supplémentaire pour sa distribution : la création d'un installateur (du moins sous *Windows*), afin de permettre à l'utilisateur d'installer facilement l'application.

Le logiciel libre choisi pour créer cet installateur sous *Windows* est *Inno Setup*. Les modalités de création de l'installateur figurent dans le même document `package.md` (cf. Annexe G).

Par ailleurs, pour chaque dépendance utilisée, je vérifie sa licence et si son utilisation commerciale est autorisée. L'ensemble de ces informations est renseigné dans un fichier `LICENSE.md` à la racine du projet. Ce fichier est affiché à l'utilisateur lors de l'installation de l'application via l'installateur. La première version du projet (V1.0 - 03/2025) a été compilée sous *Windows 11 x64* et sous *Linux Ubuntu 22 LTS (x64)*.

4.10 Tests et validation

L'ensemble des fonctions du modèle *Python* a été testé via des tests unitaires en utilisant le module `unittest`. Le script R disposait d'un jeu de données prérempli, correspondant à la session 3 réalisée lors des tests sur le terrain en Casamance, au Sénégal, en 2023. Ce jeu de données a permis d'alimenter ma batterie de tests avec des résultats cohérents. Ensuite, en raison de l'évolution du modèle et du changement de comportement de certaines fonctions, les tests ont dû être adaptés. Cependant, les tests initiaux ont permis de visualiser clairement les répercussions de ces changements et de s'assurer que le code fonctionne correctement. Parmi les tests unitaires, certains vérifient l'existence d'éléments du code (présence d'une classe), tandis que d'autres sont des tests d'assertions (vérification des types de retour, validation des valeurs retournées par les fonctions, bon comportement d'une méthode). Des objets et fichiers *Mocks* ont également été créés pour les tests.

```

1  # Les pratiques choisies pour ce test sont identiques aux itk de
    # départ du script R backtrotk_05.R
2  def test_calc_rdt(self):
3      with self.subTest("ag01"):
4          expct = 12.072049689441
5          res = self.modele.calc_rdt(self.ag1)
6          self.assertAlmostEqual(res, expct, places=6)
7      with self.subTest("ag02"):
8          expct = 14.7864906832298
9          res = self.modele.calc_rdt(self.ag2)
10         self.assertAlmostEqual(res, expct, places=6)
11         ...

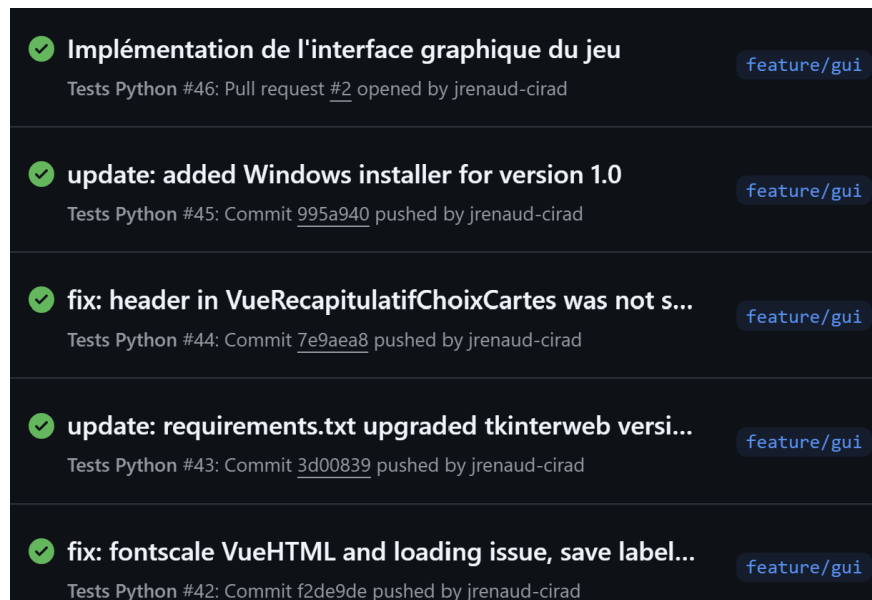
```

FIGURE 4.26 – Exemple de test unitaire : fonction `calc_rdt` du modèle

Ces tests unitaires sont exécutés automatiquement à chaque *push* et *pull request* via la *pipeline** de tests configurés au préalable avec *GitHub Actions*. La machine distante exécute les tâches suivantes dans l'ordre respectif :

1. Chargement du projet (*checkout*)
2. Installation de la bonne version de *Python* (ici 3.12)
3. Activation d'un environnement virtuel
4. Installation des dépendances du projet
5. Exécution des tests unitaires

Si une de ces tâches échoue *GitHub Actions* nous fait savoir via son interface utilisateur.

FIGURE 4.27 – Exemple de tests validés avec *GitHub Actions*

En ce qui concerne le *GUI*, il est difficile de tester les fonctions qui relèvent de l'affichage et qui dépendent sur le fait qu'une instance de la fenêtre soit créée. D'autant plus que la machine distante fonctionne en *headless* (sans interface graphique), ce qui signifie qu'instancier une fenêtre *tkinter* fera échouer la *pipeline*. Ainsi, les tests du

GUI se résume à des tests utilisateurs et des retours fait lors des réunions hebdomadaires. Une fois la première version de l'application développée, des tests fonctionnels ont été rédigés pour confirmation et validation par mes co-encadrants. L'application a été testée sous *Windows* et *Linux*. En raison de contrainte matériel et de la difficulté de se procurer une machine virtuelle *macOs*, l'application n'a pas encore été testée sur cette plateforme.

```
TESTS FONCTIONNELS
- L'interface graphique se lance et le menu de départ s'affiche
- Chaque scène s'affiche correctement (pas de dépassement de texte, taille police lisible...)
- Le changement de langue est fonctionnel. Le mode plein écran ne déforme pas les scènes
- Tous les boutons sont fonctionnels
- Le modèle affiche les bons résultats en fonction des conditions initiales et des cartes jouées
- Le mode transaction affiche les scènes de transactions après l'affichage des résultats du tour
- Le mode spatialisation fait varier le bonus
- Les pages HTML dans les sections 'A propos' et 'Règles du Jeu' s'affichent correctement (avec style CSS et images)
- Les fichiers de sorties sont bien générés dans le répertoire choisi lors de l'initialisation si la sauvegarde a été activée
- Les fichiers de sorties correspondent bien aux données de la partie
- La taille de la police est lisible peu importe les configurations de zoom de la machine (du moins avec 100%, 150% et 200%)
- La modification et la sauvegarde des paramètres dans 'Paramètres' sont bien prises en compte
- L'application est plus ou moins fluide (sauf transition entre Initialisation du Jeu et Début Jeu, environ 2s, le temps que l'application charge les scènes du jeu)

TESTS MODIFICATIONS (tuto dans /doc : customize.md, language.md, package.md)
- Changer une image
- Changer le contenu d'une page HTML (ajouter du texte et une image)
- Changer le style d'une page HTML (fichier CSS associé)
- Ajouter une langue sans forcément traduire
- Compiler l'application en suivant le guide, ouvrir l'exécutable et tester. (étape 1 et 2)
- Windows only pour le moment : Créer un installateur à partir du fichier généré par la compilation en suivant le guide (étape 3)
```

FIGURE 4.28 – Tests fonctionnels rédigés pour la version 1.0 de l'application

4.11 Difficultés rencontrées

Cette section liste les principales difficultés rencontrées au niveau du code lors de la réalisation.

Le projet a permis de comprendre plus en détail le système d'importation en *Python* (relative et absolue) et l'intérêt du fichier `__init__.py` pour déclarer un package contenant plusieurs modules. Une des difficultés rencontrées était liée à l'importation circulaire (quand un module charge un autre module, qui en charge un autre, etc., et revient au module d'origine). Le code présentait cette erreur pour la raison suivante : la déclaration d'un type de variable nécessite le chargement du module. La solution pour garder le typage (aide au développeur, mais n'est pas pris en compte à l'exécution) est de faire appel à la bibliothèque `typing`, qui possède une constante booléenne `TYPE_CHECKING` (mise à `False` lors de l'exécution). Il faut cependant encadrer la déclaration type par deux guillemets simples. Cela permet de ne pas prendre en compte l'importation lors de l'exécution, tout en préservant le typage de la variable.

```
1 cannot import name 'Gui' from partially initialized module 'src.gui.Gui'  
  (most likely due to a circular import)
```

FIGURE 4.29 – Exemple de message d'erreur dû à une importation circulaire

```
1 if TYPE_CHECKING:  
2     from src.gui.Gui import Gui
```

FIGURE 4.30 – Solution envisagée face à l'importation circulaire

Le projet a également permis d'approfondir les connaissances du langage **Python**, et notamment le fait qu'aucun attribut n'est privé. Seules des conventions de nommage permettent de simuler la propriété des attributs (un préfixe `_` pour "protected" et `__` pour "private"). La convention "private" rend l'élément plus difficile d'accès (*name mangling*). Par ailleurs, certains décorateurs **Python** permettent de renforcer les concepts de la **POO**. Par exemple, le décorateur `@property` permet de déclarer un *getter*, qui rend la valeur accessible en lecture seule (*read-only*), sauf si un *setter* a été spécifié par `@nom_variable.setter`.

Une fonction `resource_path` permet de récupérer le chemin du répertoire du projet indépendamment du lieu d'exécution du code. Il est utile, pour s'assurer que l'application fonctionne toujours en mode portable (un seul exécutable qui génère les fichiers temporaires lors de son exécution).

L'étape de compilation avec *nuitka* sous une distribution **Anaconda** de **Python** résultait en un exécutable qui ne se lance pas. Ceci est dû à la manière dont cette distribution prend en charge les dépendances et les fichiers `.dll`, différemment d'une distribution **CPython** officielle. Le problème venait du fait que l'environnement virtuel a été créé avec le module `venv`. Ainsi, pour assurer la compilation avec *nuitka* sous **Anaconda Python**, il faut créer un environnement virtuel avec l'outil `conda` emballé avec la distribution.

Chapitre 5

Méthodologie du travail

5.1 Organisation du projet

Une réunion hebdomadaire permettait de faire le point sur les fonctionnalités développées et sur les changements à effectuer, ainsi que de planifier, par ordre de priorité, les tâches à réaliser pour la semaine suivante. Cette organisation suit à peu près le rythme d'une organisation itérative et incrémentale en méthodes agiles, avec une constante évolution du cahier des charges. Pour chacun des points hebdomadaires, une *todo-list* et un compte rendu ont été rédigés. La méthodologie suit un cycle de développement court comprenant les étapes de conception, réalisation, tests et validation.

5.2 Gitflow

Étant le principal contributeur de ce projet, je me suis contenté d'une organisation linéaire du *Gitflow* et de travailler directement sur une branche de développement créée à partir de la branche principale (**main**). Cette branche englobera toutes les fonctionnalités développées correspondant aux parties conséquentes du projet et fera l'objet d'un **pull request*** (équivalent de *merge request* de *GitLab*) pour sa validation avec mes co-encadrants en tant que *reviewers*.

Le projet comptabilise au total 4 branches de développement crée dans l'ordre suivant, les uns après les autres :

- **feature/mecanique_du_jeu** : développement de toute la partie fonctionnelle du jeu (entrées, sorties, relations entre entités, outils, modèle...) et d'une première version du jeu en mode **CLI** (*merged*)
- **feature/gui** : développement du **GUI** et packaging (*merged*)
- **feature/style** : affinage du style du **GUI**
- **feature/modele_depenses** : inclure les dépenses des agriculteurs et acheteurs en sortie

5.3 Activités annexes

- Participation à la session de test du jeu en interne (décembre 2024)
- Échange sur *Course contre la mouche* et les jeux sérieux (avec mes co-encadrants, Stéphanie RABAUD et Christophe LE PAGE, janvier 2025)
- Présence aux réunions d'**UPR Hortsys** (mars 2025)
- Oral de démonstration de l'application (réunion d'**UPR Hortsys**, avril 2025)

Chapitre 6

Conclusion

Le stage effectué au sein du CIRAD a été une concrétisation professionnelle des acquis de la formation que j'ai suivie à l'IUT, en cohérence avec ma spécialité. Il a permis de mettre en œuvre mes compétences techniques dans le cadre d'un projet concret. Sur le plan du développement informatique, j'ai pu mettre en application mes connaissances en conception, en POO, en gestion de code, en Python et en organisation du travail. Il m'a permis de découvrir le secteur d'activités de l'agronomie. Il m'a aussi permis d'évoluer dans un environnement pluridisciplinaire et de m'approprier les différentes étapes de développement d'une application informatique, de sa conception à sa validation, en passant par les tests et la réalisation.

Un des points d'amélioration du projet concerne l'implémentation dans le modèle du module "variétés" qui est en cours de développement par l'équipe de chercheurs. Ce module consiste en la décomposition de la production d'un agriculteur par type de variétés. Cela impliquerait d'adapter la conception de base, notamment au niveau du modèle si de nouvelles fonctions de calcul sont nécessaires, mais aussi au niveau de l'objet `Agriculteur`. Une idée de conception modulaire est d'extraire certaines données de l'agriculteur liées à la production dans des classes de type `Variete` et faire dépendre l'agriculteur de ces classes. Il faudra également adapter l'affichage des résultats en fonction des variétés, ce qui entraîne de modifier la construction de certaines vues en fonction de si l'option a été activée. Ce module devrait pouvoir s'intégrer dans l'application, mais nécessite un travail de conception et de `refactoring` préalable à partir de la structure existante.

D'autres perspectives d'amélioration du projet considérées, sont l'implémentation d'un système permettant d'enregistrer les dépenses des joueurs agriculteurs et acheteurs, une amélioration de l'ergonomie, ainsi qu'une redéfinition de l'interface graphique pour mobile pour une meilleure portabilité. La modularité du code et la séparation de la mécanique du jeu de l'interface graphique, ainsi que la documentation des différents éléments du projet, permettront, si nécessaire, la reprise du projet pour aborder ces différents points.

Signatures

Vu le 27/03/2025

A handwritten signature in black ink, consisting of a stylized 'V' followed by a series of loops and a long horizontal stroke.

Bibliographie

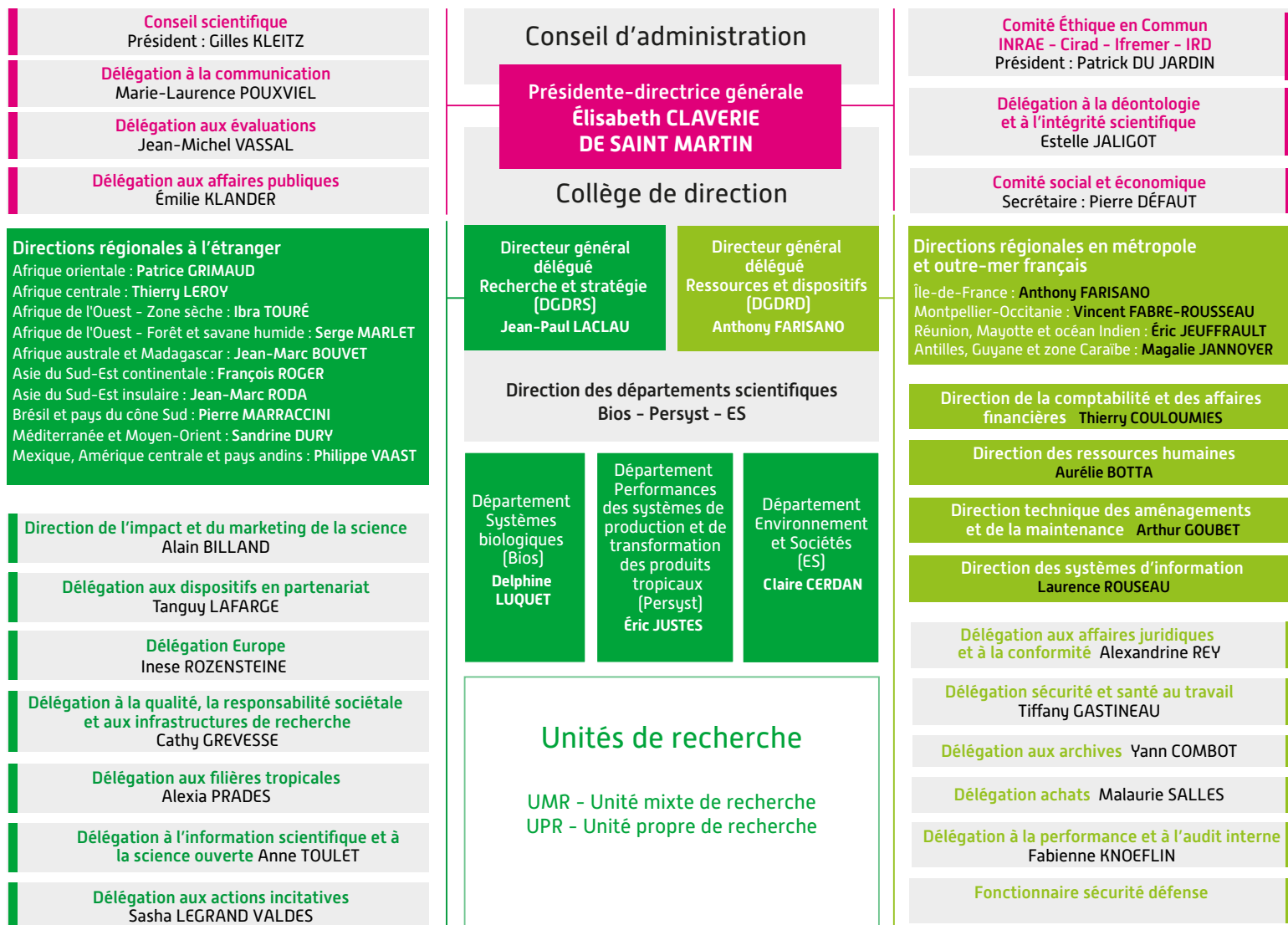
- [1] « L'ANR DISLAND : un projet de génétique du paysage pour informer la dispersion de la mouche orientale des fruits dans les paysages agricoles sénégalais -. » (), adresse : <https://passion-entomologie.fr/agroecologie-cirad-senegal/> (visité le 04/02/2025).
- [2] « Inférer la dispersion des ravageurs dans les paysages agricoles pour améliorer la gestion préventive, » Agence nationale de la recherche. (), adresse : <https://anr.fr/Projet-ANR-20-CE32-0012> (visité le 05/02/2025).
- [3] CIRAD. « Quand chercheurs et producteurs partagent leurs points de vue pour améliorer la production de mangues, » CIRAD. (22 fév. 2024), adresse : <https://www.cirad.fr/dans-le-monde/nos-directions-regionales/afrique-de-l-ouest-zone-seche/actualites/des-solutions-contre-la-mouche-orientale-des-fruits> (visité le 04/02/2025).
- [4] R. BELMIN, A. N'GOM, I. GRECHI, T. BRÉVAULT et F. REBAUDO, « "Course contre la mouche" : Un jeu de rôle pour apprendre et agir face à la mouche des fruits au Sénégal, » fr,
- [5] A. N'GOM, R. BELMIN, I. GRECHI, T. BRÉVAULT et F. REBAUDO, « Course contre la mouche : jeu de rôle pour une gestion concertée de la mouche orientale des fruits dans la filière mangue au Sénégal, » *Cirad-Agritrop*, 2023. adresse : <https://agritrop.cirad.fr/605285/>.
- [6] « HortSys - Fonctionnement agroécologique et performances des systèmes de culture horticoles. » (), adresse : <https://ur-hortsys.cirad.fr/> (visité le 05/02/2025).
- [7] R. C. TEAM, « R : A Language and Environment for Statistical Computing, R Foundation for Statistical Computing, Vienna, Ausitria., » 2023. adresse : <https://www.R-project.org/%3E>.
- [8] « R : Documentation. » (), adresse : <https://www.r-project.org/other-docs.html> (visité le 06/02/2025).
- [9] Z. BOBBITT. « How to use pmax & pmin in r (with examples), » Statology. (11 avr. 2022), adresse : <https://www.statology.org/r-pmax-pmin/> (visité le 06/02/2025).
- [10] E. KOSOUROVA. « Apply functions in r with examples [apply(), sapply(), lapply(), tapply()], » Dataquest. (31 mai 2022), adresse : <https://www.dataquest.io/blog/apply-functions-in-r-sapply-lapply-tapply/> (visité le 06/02/2025).
- [11] « Home - RDocumentation. » (), adresse : <https://www.rdocumentation.org/> (visité le 06/02/2025).
- [12] « pandas.DataFrame — pandas 2.2.3 documentation. » (), adresse : <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html> (visité le 06/02/2025).

- [13] J. D. HUNTER, « Matplotlib : A 2D graphics environment, » *Computing in Science & Engineering*, t. 9, n° 3, p. 90-95, 2007. DOI : [10.1109/MCSE.2007.55](https://doi.org/10.1109/MCSE.2007.55).
- [14] « Getter and setter in python, » GeeksforGeeks. Section : Technical Scriptor. (), adresse : <https://www.geeksforgeeks.org/getter-and-setter-in-python/> (visité le 25/03/2025).
- [15] « Using a mutable default value as an argument — Python Anti-Patterns documentation. » (), adresse : https://docs.quantifiedcode.com/python-anti-patterns/correctness/mutable_default_value_as_argument.html (visité le 25/03/2025).
- [16] « Logging — logging facility for python, » Python documentation. (), adresse : <https://docs.python.org/3/library/logging.html> (visité le 25/03/2025).
- [17] « abc — Abstract Base Classes, » Python documentation. (), adresse : <https://docs.python.org/3/library/abc.html> (visité le 25/03/2025).
- [18] « PyInstaller Manual — PyInstaller 6.12.0 documentation. » (), adresse : <https://pyinstaller.org/en/stable/> (visité le 15/03/2025).
- [19] (), adresse : <https://pyyaml.org/wiki/PyYAMLDocumentation> (visité le 05/02/2025).
- [20] ANDREW, *Andereoo/TkinterWeb*, original-date : 2021-01-07T13:11:32Z, 20 mars 2025. adresse : <https://github.com/Andereoo/TkinterWeb> (visité le 18/02/2025).
- [21] « User Documentation — Nuitka the Python Compiler. » (), adresse : <https://nuitka.net/user-documentation/> (visité le 15/03/2025).
- [22] « Beautiful Soup Documentation — Beautiful Soup 4.13.0 documentation. » (), adresse : <https://www.crummy.com/software/BeautifulSoup/bs4/doc/> (visité le 18/02/2025).
- [23] « Annotation issues at runtime - mypy 1.15.0 documentation. » (), adresse : https://mypy.readthedocs.io/en/stable/runtime_troubles.html (visité le 25/03/2025).
- [24] « Documentation introduction | CustomTkinter. » (), adresse : <https://customtkinter.tomschimansky.com/documentation/> (visité le 12/02/2025).
- [25] « How to modify HTML using BeautifulSoup ? » GeeksforGeeks. Section : Python. (), adresse : <https://www.geeksforgeeks.org/how-to-modify-html-using-beautifulsoup/> (visité le 18/02/2025).
- [26] « Welcome to TkinterWeb's documentation! — TkinterWeb 4.2 documentation. » (), adresse : <https://tkinterweb.readthedocs.io/en/latest/index.html> (visité le 18/02/2025).
- [27] « Patrons de conception / Design patterns. » (), adresse : <https://refactoring.guru/fr/design-patterns> (visité le 02/03/2025).
- [28] « Qualité de développement (R3.04). » (), adresse : <https://mgasquet.github.io/R304-QualiteDeveloppement/> (visité le 02/03/2025).

Annexe A

Organigramme du CIRAD

Organigramme du Cirad - Décembre 2024



Annexe B

Diagrammes UML

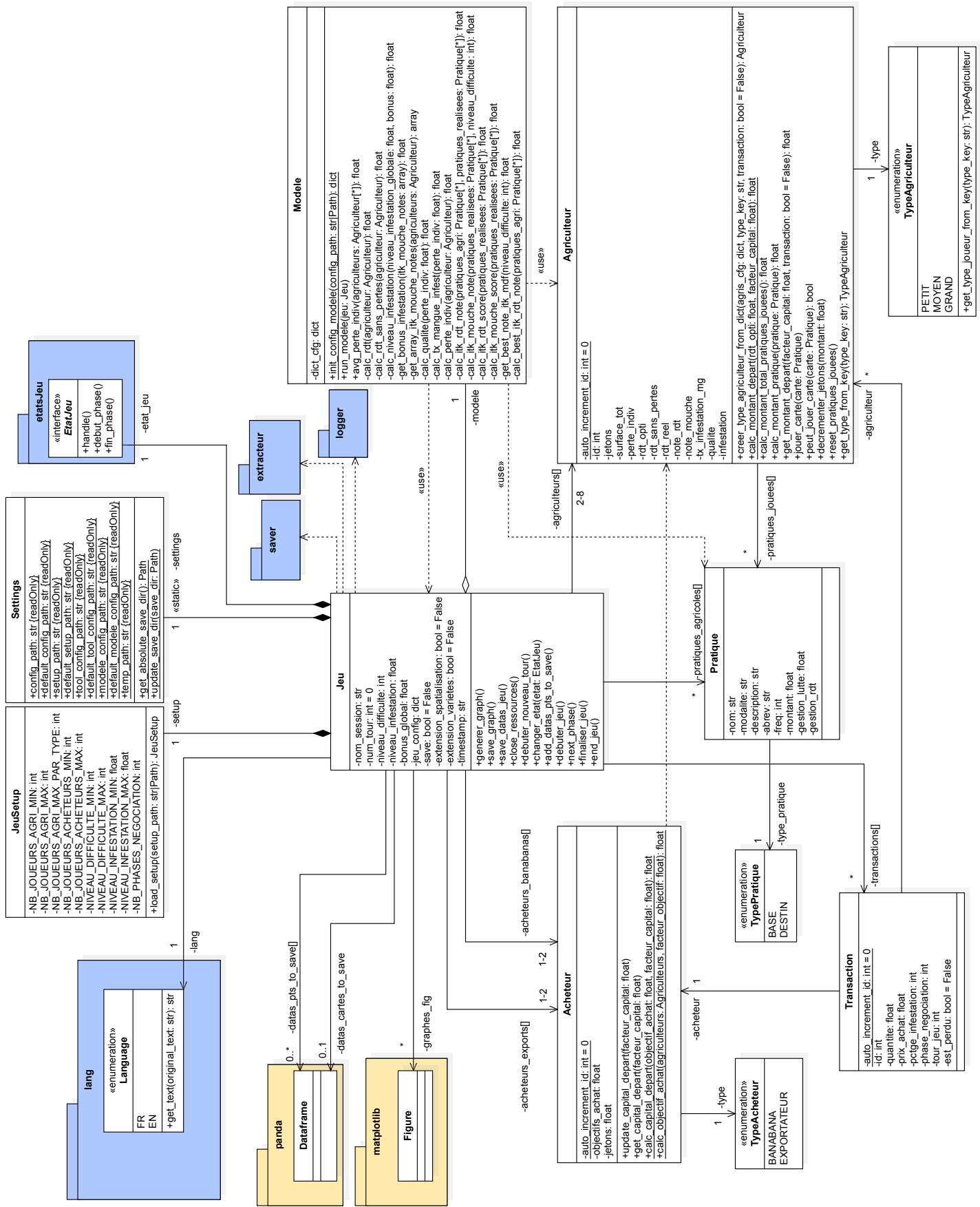


FIGURE B.1 – Diagramme de classes UML de conception de la mécanique du jeu (en bleu les packages internes et en jaune les packages externes)

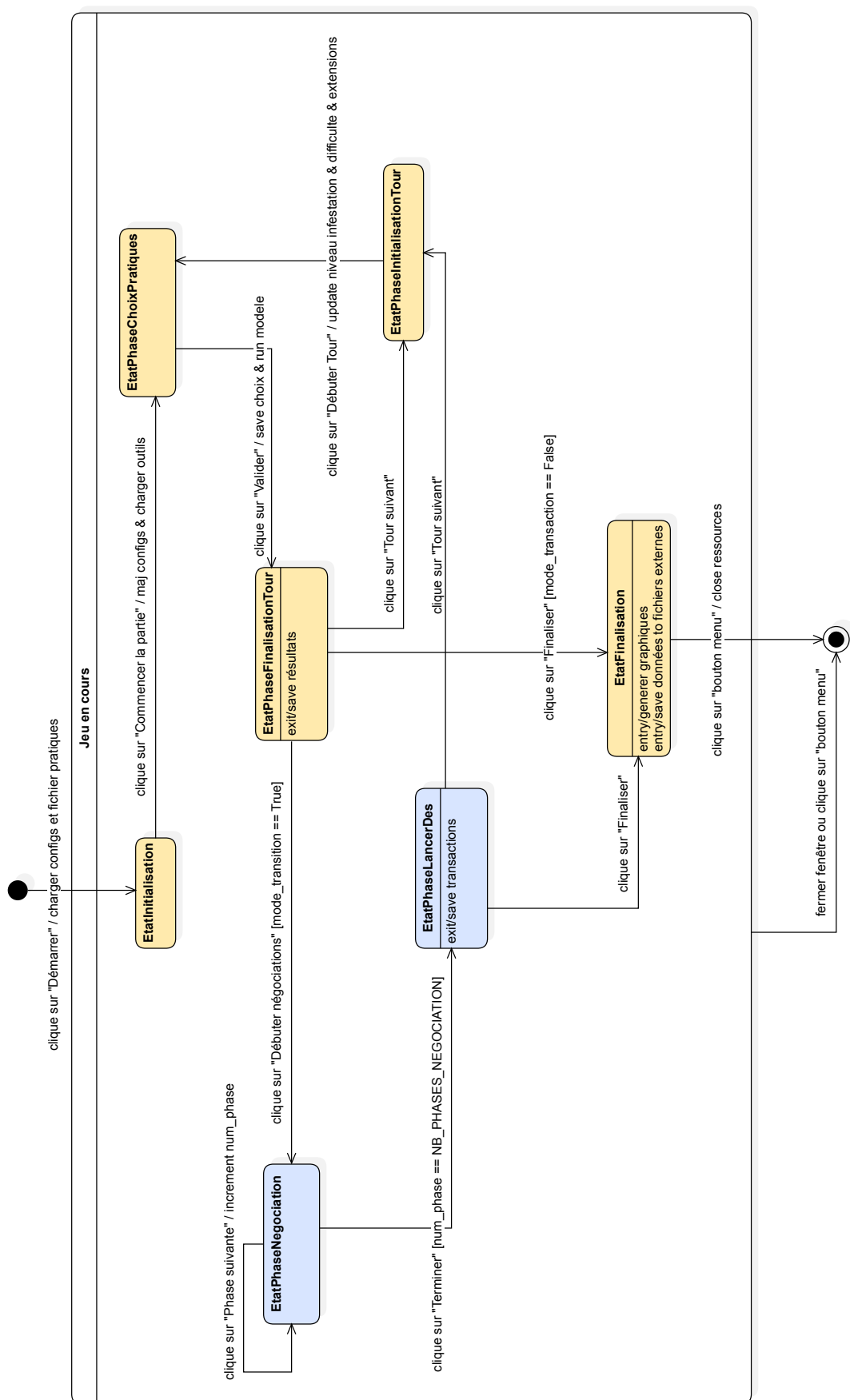


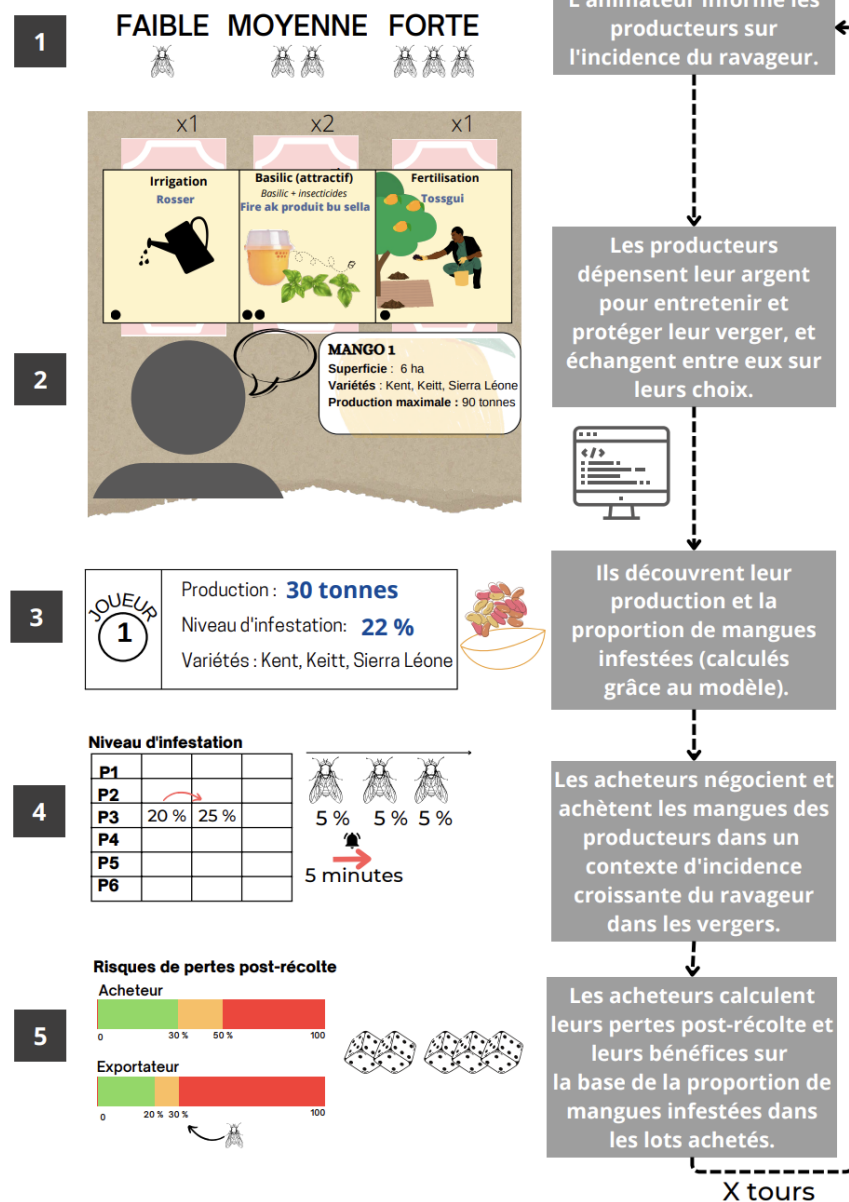
FIGURE B.2 – Diagramme etats-transitions de l'objet Jeu

Annexe C

Déroulement du jeu

Déroulé d'une partie

Initialisation du jeu, tours de jeu et débriefing



Extrait du poster du jeu [5]

Annexe D

Protoype Figma

Menu et Paramètres

Race against the flies

Start

Rules

Settings

Quit

EN

General

Modele

Game

Language
EN

Results save directory
./data/

Logs directory
./logs/

Save

Initialisation de la partie



Nom de la partie Reset Default

SESSION_CASAMANCE

Difficulté



Niveau Infestation 0-1

0.4

Joueurs Agriculteurs 2-6



Joueurs BanaBana 2-6



Joueurs Exportateurs 2-6



Sauvegarder

Sauvegarder les données du jeu en fin de partie



Extensions

- ☒ Spatialisation
Information sur la spatialisation
- ☒ Variétés
Information sur les variétés
- ☒ Pénalités
Information sur la pénalité

Config. Agriculteurs



	Surface	Rdt Optimal
Agriculteur 1	50	20T
Agriculteur 2	50	20T
Agriculteur 3	50	20T
Agriculteur 4	50	20T
Agriculteur 5	50	20T
Agriculteur 6	50	20T

Démarrer

Choix des pratiques



TOUR N

Choix des pratiques



Difficulté : 3

Infestation globale : 0.3

Select Joueur



7 \$



Valider

Récapitulatif du choix des pratiques et Résultats du tour



TOUR N

Choix des pratiques



TOUR N

Résultats

Difficulté : 3

Infestation globale : 0.3

Joueur1	1 irr, 2 fer, 1 ins	7 \$
Joueur2	1 irr, 2 fer, 1 ins	8\$
Joueur3	1 irr, 2 fer, 1 ins	7 \$
Joueur4	1 irr, 2 fer, 1 ins	8\$
Joueur5	1 irr, 2 fer, 1 ins	7 \$
Joueur6		0\$

Go back

Confirmer

	Infestation	Rendement
Joueur1	31%	19 T
Joueur3	20%	26 T
Joueur4	21%	30 T
Joueur5	26%	17 T
Joueur6	19%	30 T

Prochain tour

Terminer la partie

Annexe E

Tutoriel traduction

Le document ci-dessous est issu du fichier `language.md`

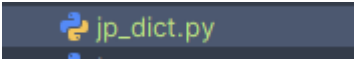
Traduction et langues

1. [Ajouter une nouvelle langue](#)
2. [Changer la traduction d'une langue existante](#)
3. [Supprimer une langue](#)

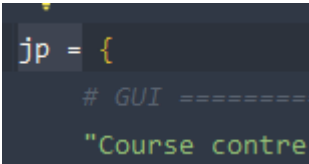
Ajouter une nouvelle langue

Les captures d'écran montrent l'ajout d'une langue à travers un exemple : le japonais (JP)

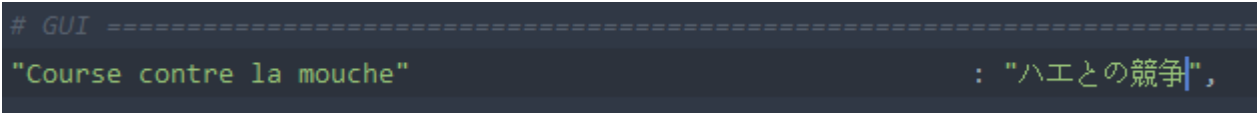
1. Faites une copie du fichier [src/lang/template_dict.py](#) ou d'un des dictionnaires de langue existant en le renommant '[abrev_langue]_dict.py'. Placez le dans le même dossier [src/lang/](#)



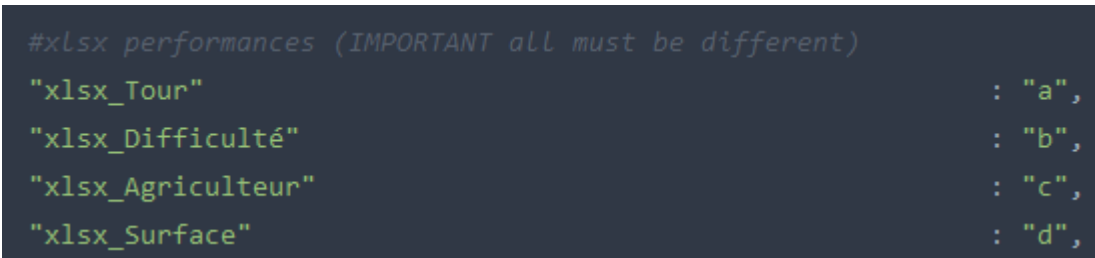
2. Remplacez le nom du dictionnaire (1ère ligne du fichier) de template à [abrev langue]



3. Traduisez chacun des termes en remplissant/remplaçant toutes les valeurs de DROITES (attention certaines valeurs doivent différer des autres, suivre les commentaires dans le code)

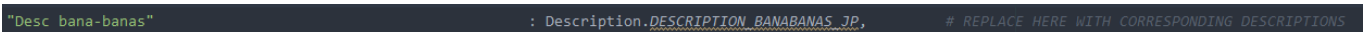


Exemple de traductions où chaque champ doit être différents



ATTENTION : alterer les valeurs de gauches (clés du dictionnaire) correspondant aux valeurs renseignées dans le code source, risque de faire dysfonctionner l'application

4. Pour les descriptions longues : remplacez sur la variable correspondant à la description, l'abréviation par celle de votre langue (en majuscules) :



Puis dans [src/lang/description.py](#) rajoutez la nouvelle variable et sa description :

```
class Description(StrEnum): 32 usages  🧑 igrechi +1 *
    DESCRIPTION_BANABANAS_FR = "Commerçants qui approvisionnent le marché domestique avec tous types de variétés"
    DESCRIPTION_BANABANAS_EN = "Retailers supplying the domestic market with all types of varieties"
    DESCRIPTION_BANABANAS_JP = "国内市場にあらゆる種類の商品を提供する小売業者"
```

5. Une fois traduit, allez dans `src/lang/Language.py`, dans la classe `Language` ajoutez `[ABREV_LANGUE] = [abrev_lang]_dict.[abrev_lang]`. N'oubliez pas d'importer le nouveau dictionnaire dans cette même classe.

```
class Language(DictEnum):  🧑 Julien Renaud *
    FR = fr_dict.fr
    EN = en_dict.en
    💡 JP = jp_dict.jp

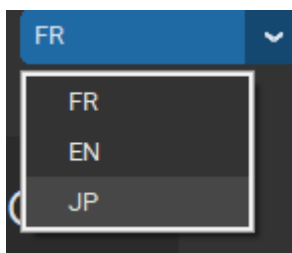
    def get_text(self, original_text: str) -> str:  🧑 Julien Renaud
        return self.value.get(original_text) or "Translation not found"
```

Mise à jour des imports :

```
from . import fr_dict, en_dict, jp_dict
```

N.B. En faisant cela, vous créez une nouvelle classe (énumération) de type `Language` qui a pour valeur le nouveau dictionnaire que vous venez de créer et traduire. Pour s'assurer que le dictionnaire reste du bon format, vous pouvez lancer les tests suivants depuis votre environnement virtuel à la racine du projet, via la commande : `python -m unittest test.test_Language.TestLanguage`

6. Visualisez le résultat :





Changer la traduction d'une langue existante

Il suffit de remplacer les valeurs de droites dans le dictionnaire associés

Supprimer une langue

Il suffit de supprimer la valeur dans [src/lang/Language.py](#), les descriptions associés dans [src/lang/description.py](#) et de supprimer le fichier contenant le dictionnaire associé.

ATTENTION : l'application a besoin d'au moins une langue pour afficher correctement, veuillez à ne pas supprimer toutes les langues.

Annexe F

Tutoriel personnalisation

Le document ci-dessous est issu du fichier `customize.md`

Ressources et Style

1. [Modifier les images](#)
2. [Modifier les pages HTML](#)
3. [Modifier le style GUI](#)

Modifier les images

Les images et icônes de l'application sont répertoriées dans le dossier [ressources/ico/](#) et dans [ressources/img/](#). Pour remplacer une image, il suffit d'écraser l'ancienne image par la nouvelle image souhaitée.

ATTENTION : pour garder un affichage correct, veillez à respecter les proportions des images. Il faut également garder le même format ([.png](#) pour la plupart, supporte la transparence).

Information complémentaire : pour les images des cartes dans [ressources/img/cartes/](#), il est possible de configurer le ratio (proportion) des cartes. Il suffit de modifier la valeur [RATIO_CARTE](#) dans la classe [StyleCarte](#) du fichier [src/gui/style/style.py](#).

Modifier les pages HTML

Afin d'afficher du contenu lié à la documentation du jeu : sections '*Règles Du Jeu*' et '*A propos*', le choix s'est tournée vers l'affichage sous forme de pages **HTML** stockées local. Ces pages peuvent être accompagnées d'une feuille de style **CSS** et peuvent contenir des images (le code actuel ne permet pas d'injecter du **JS**). Ces pages se trouvent dans le dossier [ressources/html](#). Elles peuvent également être traduites dans les langues existantes à condition de respecter les règles suivantes :

- Les chemins spécifiés dans les fichiers HTML, pour la feuille de style et les images doivent être **relatif au repertoire du projet** (le code s'occupera ensuite d'injecter les chemins absolus lors de l'exécution).


```

<!DOCTYPE html>
<html Lang="fr">
  <head>
    <meta charset="utf-8"/>
    <meta content="width=device-width, initial-scale=1.0" name="viewport"/>
    <title>
      À Propos de Nous
      🌟/title>
    <link href="ressources/css/aPropos.css" rel="stylesheet"/>
  </head>
  <body>
    <header>
      <div class="container">
        
        
        <h1>

```

- Les fichiers HTML doivent être définis dans le répertoire `/ressources/html/[langue]/` ou `[langue]` est remplacé par l'abréviation correspondant à la langue. Les fichiers CSS sont définis dans `/ressources/css/`
- Les fichiers HTML pour les sections 'Règles Du Jeu' et 'A propos' doivent être nommés respectivement `reglesDuJeu.html` et `aPropos.html`. De même pour le CSS, `reglesDuJeu.css` et `aPropos.css`.

ATTENTION : le moteur web utilisé par `tkinterweb` ne supporte pas certaines balises HTML (la balise video par exemple) et attributs CSS (les gradients par exemple). (en savoir plus sur <https://tkinterweb.readthedocs.io/en/latest/index.html>)

Modifier le style GUI

Le thème de base du GUI est chargé via le fichier `ressources/themes/app_theme.json`. Pour les couleurs, deux valeurs sont renseignées `[couleur_light, couleur_dark]`. Ce fichier définit les propriétés des widgets de bases. Certains composants de l'application sont des regroupements de plusieurs widgets de bases. Ils sont ainsi configurés différemment et leurs propriétés de bases sont écrasées par les configurations définies dans le fichier `src/gui/style/style.py`.

Toutes ces valeurs sont modifiables sans trop altérer la disposition des composants (attention tout de même pour la taille des polices).

Les couleurs peuvent être spécifiées en hexadécimal ou en notation prise en charge par `tkinter` ("white", "black", "red", "green", "blue", "cyan", "yellow", "magenta" ...).

Annexe G

Tutoriel packaging

Le document ci-dessous est issu du fichier `package.md`

Distribuer le projet

- [Prérequis](#)
- [Etape 1 : Installation de Nuitka](#)
- [Etape 2 : Compilation du projet](#)
- [Etape 3 : Création de l'installateur \(mode standalone\)](#)

Le but de ce document est de renseigner les différentes étapes pour convertir le projet *Python* en un répertoire contenant un **exécutable**, et d'en faire un **installateur** distribuable. Pour cela, nous utilisons le package *Python* multiplateforme *Nuitka*. Il s'agit d'un compilateur optimisé qui convertit le code source *Python* en code source C optimisée avant de le convertir en code machine (binaire). Le choix d'utiliser *Nuitka* à la place d'autres solutions connues telles que *PyInstaller* ou *cx_freeze* s'appuie sur les critères suivants :

- Optimisation de la compilation
- Rapidité du lancement de l'exécutable pour le mode **standalone** (0 à 3s)
- Non-signallement de l'exécutable comme un logiciel malveillant par les antivirus (faux positif)

Le troisième critère est un point important. Lors des tests réalisés avec *PyInstaller* (6.12.0) et *cx_freeze* (7.3.0) sous *Windows 11*, l'exécutable créé (notamment lorsque que la compilation se fait avec le mode **--noconsole** ou **--windowed**, le fichier **.exe** a été signalé comme un logiciel malveillant (Trojan) et directement mis en quarantaine par l'antivirus lors de son lancement (Antivirus testé : *WithSecure Elements*, *McAfee* et *Windows Defender*). Ceci est un problème récurrent et à avoir avec la manière dont *PyInstaller* compile le code source ([StackOverflow](#), [PyInstaller Github Issues](#)). Il existe des alternatives pour contourner ce problème en utilisant une version antérieure du package ou bien en compilant soit même le bootloader de *PyInstaller* ([Alternatives](#), [Bootloader](#)). Malgré ces alternatives, les exécutables générés par *PyInstaller* sont souvent lent au lancement (de l'ordre de 5 à 10s). Afin de faciliter ce processus et d'éviter l'utilisateur de mettre à jour sa liste d'exclusion ou de désactiver son anti-virus pour lancer l'application, nous choisissons d'utiliser *Nuitka* (<https://nuitka.net/>).

Pour le mode **standalone**, pour packager le tout avec un installateur, il faudra utiliser un logiciel tiers qui dépend de la plateforme. Nos choix :

- Sous *Windows* : *InnoSetup*
- Sous *Linux* : *A venir...*
- Sous *MacOs* : *A venir...*

N.B. Les fichiers binaires générés par l'étape de compilation sont plateformes spécifiques. Il faudra donc compiler sur les 3 principales plateformes (*Linux*, *Windows* et *MacOs*).

Prérequis

- Python et les dépendances du projet sont installés dans un environnement virtuel **.venv** (voir [README](#))
- (*Linux*) package **patchelf** installé sur la distribution : **apt/dnf/yum install patchelf**

Etape 1 : Installation de *Nuitka*

Dans votre environnement virtuel exécutez : **python -m pip install nuitka==2.6.8**

N.B. les executables de la version 1.0 ([installers/](#)) de l'application ont été réalisées avec la version 2.6.8 de *Nuitka*.

Etape 2 : Compilation du projet

Pensez à remettre les configurations par défaut si nécessaire pour les fichiers de configs [config/](#) avant de packager (copiez-collez les valeurs des configs par default dans les fichiers configs associés).

Pour générer la commande de compilation, vous pouvez executez le script de [get_build_cmd.py](#) depuis la racine du projet : `python -m get_build_cmd <mode>` (remplacez `<mode>` par `standalone` pour générer un dossier de distribution ou par `portable` pour générer un seul exécutable)

- Mode `standalone` : lancement nettement plus rapide, optimisé, intégré dans l'OS, permet l'edition et la sauvegarde des fichiers de configurations, souvent accompagné d'un installeur.
- Mode `portable` : application portable (pas d'installeur), un seul fichier, **les changements de configurations ne sont pas sauvegardés**.

Copiez la sortie du script et executez la commande, la compilation devrait se lancer.

Ce processus peut prendre plusieurs dizaines de minutes (notemment la première fois où le cache n'est pas encore mis en place)

Nuitka vous demandera également d'installer un compilateur C si ce n'est pas déjà fait. Il suffit de repondre avec `y` lorsqu'il vous le sera demandé.

Une fois compilé, lancez l'executable et assurez-vous que tout fonctionne.

En savoir plus

Nuitka compile le code source *Python* en code source *C* avant de le convertir en binaires. Cependant, le projet nécessite des fichiers externes pour fonctionner (qui ne se terminent pas par l'extension `.py`). Il faut donc spécifier à *Nuitka* de les inclure dans le répertoire final. Les répertoires contenant les fichiers à inclure sont [config](#), [ressources](#) et [data](#). L'option `--include-data-dir="[CHEMIN_DU_DOSSIER]"="[CHEMIN_DESTINATION]"` permet d'inclure ces répertoires de données.

La difficulté repose sur les dépendances utilisées qui eux mêmes utilisent des fichiers de données externes. Heureusement que pour la plupart des librairies connues *Nuitka* se charge d'inclure ces fichiers automatiquement via leurs plugins. Cependant, pour le package `tkinterweb` utilisé par notre application, il faudra l'indiquer malgré tout. Bien que spécifié dans les configurations et la documentation de *Nuitka* à ce sujet (<https://nuitka.net/posts/nuitka-package-config-kickoff.html>), l'application ne se lance pas et renvoie l'erreur `_tkinter.TclError: can't find package Tkhtml` (même souci renseigné par ce commentaire : <https://github.com/Nuitka/Nuitka/issues/1998#issuecomment-1386433893>). Après avoir minutieusement examiné le fichier `utilities.py` du package, il semblerait que le module charge le package `Tkhtml` (<http://tkhtml.tcl.tk/>) via un fichier `.dll` sous *Windows* (ou `.so` sur *Linux*, `.dylib` sur *macOs*). Pour charger ce fichier, le programme utilise la variable

```
ROOT_DIR = os.path.join(os.path.abspath(os.path.dirname(__file__)), "tkhtml")
```

et la fonction suivante pour repérer le repertoire contenant le fichier.

```
get_tkhtml_folder() : return os.path.join(ROOT_DIR, "binaries")
```

Tout se passe comme prévu lorsque l'on démarre l'application via le script *Python*. Le problème est lorsque le programme se lance sous la forme d'un executable où le comportement diffère. En effet le bout de code `os.path.dirname(__file__)` va retourner le repertoire dans lequel se trouve l'executable. Par conséquent, il va chercher dans son repertoire un dossier `tkhtml`. Sauf que ce dossier n'existe pas. Il faut donc inclure ce dossier et son contenu à la racine de l'executable (en faisant de la sorte, on se restreint de modifier le code source du package `tkinterweb` pour éviter tout autre dysfonctionnement).

En utilisant l'option `--include-data-dir="[CHEMIN_DU_DOSSIER]"="[CHEMIN_DESTINATION]"` *Nuitka* exclu les fichiers `.dll` (et autres binaires) (<https://nuitka.net/user-documentation/user-manual.html>). Il faut donc inclure le fichier en question avec l'option `--include-data-files="[CHEMIN_DU_FICHIER]"="[CHEMIN_DESTINATION]"`

"Nuitka does not consider code to be data files and will not include DLLs or Python files as data files" (<https://nuitka.net/user-documentation/user-manual.html>)

Pour repérer le dossier dans lequel `tkinterweb` réside depuis votre environnement virtuel : `pip show tkinterweb`

```
(.venv) D:\Mes Programmes\course_contre_la_mouche>pip show tkinterweb
Name: tkinterweb
Version: 4.2.0
Summary: HTML/CSS viewer for Tkinter
Home-page: https://github.com/Andereoo/TkinterWeb
Author:
Author-email:
License: MIT
Location: D:\Mes Programmes\course_contre_la_mouche\.venv\Lib\site-packages
Requires: pillow
Required-by:
```

Pour repérer le repertoire `tkhtml` et le fichier binaire utilisé par le package pour charger `tkhtml` :

- (Windows) `dir "[chemin vers tkinterweb]\tkinterweb*.dll"`
- (Linux) `find "[chemin vers tkinterweb]/tkinterweb" -type f -name "*.so"`
- (MacOs) `find "[chemin vers tkinterweb]/tkinterweb" -type f -name "*.dylib"`

```
(.venv) D:\Mes Programmes\course_contre_la_mouche>dir /s /b "D:\Mes Programmes\course_contre_la_mouche\.venv\Lib\site-packages\tkinterweb\*.dll"
D:\Mes Programmes\course_contre_la_mouche\.venv\Lib\site-packages\tkinterweb\tkhtml\binaries\tkhtml30.dll
```

Prenant tout cela en compte, on forme la commande finale suivante (ici mode `standalone`) : 69 sur 71

```
python -m nuitka --windows-console-mode=disable --enable-plugin=tk-inter --python-flag=no_site --standalone --include-data-dir="[CHEMIN_DU_REPERTOIRE_TKHTML]"=
[NOM_DU_REPERTOIRE_TKHTML] --include-data-files="
[CHEMIN_DU_FICHER_BINAIRE_TKHTML]"="[CHEMIN_RELATIF_DU_FICHER_BINAIRE_TKHTML]" -
--include-data-dir=".\\data"=data --include-data-dir=".\\ressources"=ressources --
include-data-dir=".\\config"=config app.py
```

Avec l'exemple de la capture d'écran ci-dessus, on a :

- `[CHEMIN_DU_REPERTOIRE_TKHTML]` : D:\Mes Programmes\course_contre_la_mouche.venv\Lib\site-packages\tkinterweb\tkhtml
- `[NOM_DU_REPERTOIRE_TKHTML]` : tkhtml
- `[CHEMIN_DU_FICHER_BINAIRE_TKHTML]` : D:\Mes Programmes\course_contre_la_mouche.venv\Lib\site-packages\tkinterweb\tkhtml\binaries\Tkhtml30.dll
- `[CHEMIN_RELATIF_DU_FICHER_BINAIRE_TKHTML]` : tkhtml\binaries\Tkhtml30.dll

Option à ajouter sous *macOs* : `--macos-create-app-bundle`

Pour rajouter une icone, inclure l'option :


- (*Windows*) `--windows-icon-from-ico=[CHEMIN_ICON_ICO]`
- (*MacOs*) `--macos-app-icon=[CHEMIN_ICON_ICO]`


N.B. Il est possible de spécifier si on le souhaite le chemin de destination du répertoire généré contenant l'exécutable en ajoutant l'option `--output-dir="[CHEMIN]"`. Sans cette option *Nuitka* place le répertoire à la racine du projet sous le nom de `app.dist`, ou un fichier `app.exe` ou `app.bin` pour le mode `portable`


Etape 3 : Création de l'installateur (mode standalone)

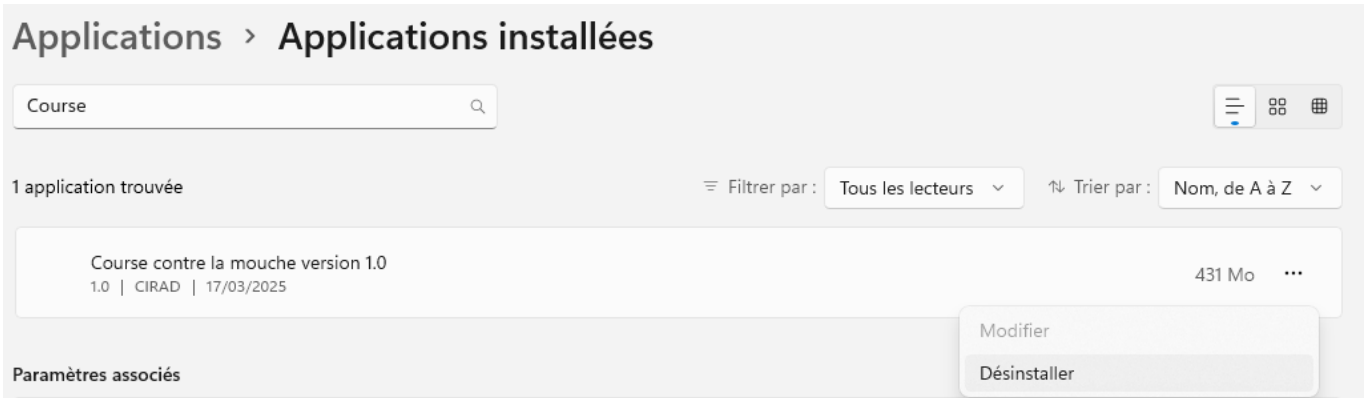
Sous *Windows* :

1. Installez le logiciel libre *InnoSetup* sur <https://jrsoftware.org/isdl.php>
2. Lancez le logiciel et cochez `Create a new script file using the Script Wizard` ou `Open an existing script file` si vous avez déjà créé un script, puis cliquez sur `OK`
3. Le wizard se lance, cliquez sur `Next`
4. Renseignez les informations concernant votre application puis cliquez sur `Next`
5. Cliquez à nouveau sur `Next`
6. Renseignez le chemin vers votre exécutable en cliquant sur `Browse`
7. IMPORTANT : Renseignez le dossier contenant votre exécutable en cliquant sur `Add folder...` puis sur `Oui` pour inclure tous les sous dossiers. Ensuite, cliquez sur `Next`
8. Décochez l'option `Associate a file type to the main executable` puis cliquez sur `Next`
9. Cochez les options qui vous conviennent puis cliquez sur `Next`
10. Renseignez le fichier de licence (`Licence.txt` à la racine du projet) puis cliquez sur `Next`
11. Cochez les options qui vous conviennent puis cliquez sur `Next`
12. Cliquez sur `Next`
13. Sélectionnez les langues souhaitées, puis cliquez sur `Next`
14. Renseignez le chemin du fichier de sortie de l'installateur, son nom et optionnellement son icone. Puis cliquez sur `Next`

15. Cliquez sur **Next**
16. Cliquez sur **Finish**
17. Cliquez sur **Oui** ou le bouton compiler  (CTRL + F9) pour débiter la compilation (cliquez sur bouton save (CTRL + S) si vous voulez sauvegarder le script).
18. Une fois compilé, lancez l'installateur et vérifiez qu'il installe bien le programme sur votre machine et que

 `course_contre_la_mouche_VERSION_1_0_setup.exe`
tout fonctionne.

N.B. Pour désinstaller, lancez le fichier `unins000.exe` dans le répertoire de l'application  ou bien via le panneau de configuration de *Windows* ("Ajoutez ou supprimer des programmes").



Note : si vous avez déjà installé l'application, et que vous exécutez à nouveau le même l'installateur (ou un installateur généré par un même script), il se peut qu'il annule l'installation (car même **AppId** dans script .iss). Dans ce cas, veuillez désinstaller l'application avant de pouvoir réinstaller avec le même installateur.