SCFE: Rapport de projet (français)

Matthieu Stombellini

Mathieu Rivier

François Soulier

Rakhmatullo Rashidov

Contents

1 Introduction

Ce document est le rapport final de notre projet de second semestre : SCFE (Salamanders' Console File Explorer). Il fournit des détails sur l'idée d'origine, le processus de réalisation du projet, ainsi qu'une description du résultat final.

1.1 Termes techniques utilisés dans ce document

Ce document peut être lu par des personnes qui ne sont pas habituées aux termes liés au développement logiciel. Voici une courte liste de définitions compréhensibles pour les mots que nous utiliserons tout au long de ce document :

- Librairie: une collection de fonctions, méthodes ou autres ressources réutilisables créée par des développeurs pour d'autres développeurs
- Framework: une librairie donnant une base beaucoup plus consistante et complète, souvent utilisée comme échafaudage pour des applications entières
- Implémentation: une manière d'encoder les définitions de fonctions. Par exemple, si les définitions étaient le résumé d'un livre, l'implémentation serait le contenu du livre
- `.NET`: un ensemble de frameworks, de langages informations et d'outils par Microsoft pour le développement d'applications
- `.NET Framework`: l'implémentation la plus ancienne des standards `.NET`, qui ne fonctionne que sur Windows
- `Mono`: un projet tiers qui vise à rendre `.NET Framework` disponibles sur des machines sur Linux ou Mac OS
- `.NET Core`: une implémentation relativement récente du standard `.NET` qui est entièrement compatible avec tous les systèmes d'exploitation

(Windows, Linux, Mac OS) et beaucoup plus véloce que `.NET Framework`

- `C#`: un langage de programmation créé pour le `.NET`. D'autres langages de `.NET` existent: par exemple `F#` ou `VB.NET`
- `UI` et `GUI`: acronymes de User Interface (interface utilisateur) et Graphical User Interface (interface graphique utilisateur)

1.2 Description générale du projet

Le but de SCFE est de fournir un explorateur de fichiers facile à utiliser pour tous, des débutants aux utilisateurs plus avancés.

L'idée est née d'un ensemble de problèmes dans la manière habituelle de manipuler les fichiers : d'une part, les explorateurs de fichiers graphiques (comme Finder sous Mac OS ou File Explorer sous Windows) sont très clairs, fournissent beaucoup d'informations, mais sont assez lents à utiliser car ils nécessitent l'utilisation de la souris, et peuvent aussi être un peu limités quand il faut utiliser des fonctionnalités plus avancées. Bref, c'est une excellente option pour les débutants et les personnes qui aiment être dans un environnement assez indolore. D'autre part, les consoles et les terminaux sont des options très populaires et, bien qu'ils ne soient pas des explorateurs de fichiers en soi, ils sont toujours utilisés comme tels. Ils contiennent toutes les différentes fonctionnalités des explorateurs de fichiers et bien plus encore, tout en étant extrêmement utiles pour la productivité, la prochaine action de l'utilisateur n'étant qu'à quelques pressions de touche. C'est une excellente option pour les utilisateurs qui aiment faire les choses rapidement et qui n'ont pas peur d'avoir une courbe d'apprentissage assez raide, mais c'est une très mauvaise option pour les débutants, qui doivent apprendre beaucoup de nouvelles commandes et doivent faire face à un certain manque de clarté dans la plupart des endroits. Il faut souvent chercher sur Internet pour comprendre comment faire quelque chose que vous pourriez faire en seulement quelques clics sur un explorateur de fichiers classique.

Nous avons donc décidé de créer un outil qui offrirait le meilleur des deux mondes : un outil facile à utiliser mais qui vous permet de faire des choses très rapidement, facile à apprendre et toujours à vos côtés au cas où vous auriez oublié comment effectuer une action. Il est également assez intuitif et ne nécessite pas beaucoup de connaissances pour pouvoir accéder à toutes ses

fonctionnalités, même les plus puissantes.

Dans l'ensemble, nous voulions créer un logiciel qui soit avant tout utile, quelque chose que nous voudrions utiliser nous-mêmes. Parce que nous le trouvons utile pour nous, nous pensons que d'autres le trouveront aussi utile.

Pour augmenter ce "facteur d'utilité", l'un de nos objectifs était de rendre SCFE multiplateforme, c'est-à-dire qu'il pourrait être utilisé sur n'importe quelle plateforme : Windows, Mac OS et Linux. Il s'agissait d'une contrainte majeure qui a causé quelques problèmes lors de la réalisation du projet.

Telles sont les valeurs fondamentales que nous avons suivies lors de la création de SCFE, et le résultat final les respecte autant que possible.

Nous avons décide de construire SCFE comme une application console pour de multiples raisons :

- Cela facilite la compatibilité multiplateforme et, dans l'ensemble, les consoles sont disponibles sur n'importe quel ordinateur, ce qui signifie que nous n'avions pratiquement aucune dépendance autre que le framework de base.
- Cela nous donnait une flexibilité quasi illimitée
- Il est beaucoup plus facile de travailler dans une console que dans une interface graphique complète, et les consoles donnent en général une expérience plus propre et fluide, notamment dûe au fait qu'une console soit beaucoup plus légère qu'une application complète de bureau
- Cela nous donnait une meilleure intégration pour les utilisateurs habitués aux environnements console. Les autres utilisateurs peuvent toujours ouvrir l'application avec une icône normale.

La base du fonctionnement de SCFE est son architecture modale. Il y a trois modes d'utilisation principaux que nous avons développés et étendus

pendant le projet. Les modes sont décrits dans la section des résultats finaux ainsi que dans notre cahier des charges.

1.2.1 Fonctionnalités

Voici quelques fonctionnalités que nous voulions implémenter:

- Ouverture de fichier basés sur l'application par défaut du système (par exemple ouvrir des fichiers .docx avec Word)
- Compatibilité avec les opérations de fichier de base (renommer, supprimer...)
- Sélection de multiples fichiers, copier et déplacer plusieurs fichiers à la fois
- Support pour aller directement dans un dossier donné
- Compatibilité multiplateforme avec Windows, Mac OS et Linux
- Compatibilité avec les commandes en console
- Compatibilité avec le logiciel de versionnage Git
- Système modal: mode `NAV` pour naviguer, mode `SEA` (search) pour rechercher, mode `COM` pour entrer des commandes de console
- Facile à utiliser avec des commandes intuitives
- Documentation simple mais complète
- Fluidité globale sur toute l'application

1.2.2 Détails techniques

SCFE est basé sur le framework `.NET Core` et est écrit dans le langage de programmation `C#`. Nous avons choisi le premier pour sa grande compatibilité multiplateforme et le second pour sa proximité avec d'autres langages que nous connaissions déjà.

SCFE utilise également des librairies listées ci-dessous:

- `JetBrains.Annotations`, une librairie qui nous donnait des indications pendant le développement pour une meilleure détection d'erreurs. Elle est uniquement utilisée pendant le développement et n'a pas d'impact pour l'utilisateur final
- `MoreLINQ`, une librairie qui nous donne plus de raccourcis pour gérer les collections de données en `C#`. Elle a rendu certaines requêtes partculièrement facile à faire comme par exemple "Donne moi la valeur maximum de cette séquence". Ce type de requête a pû être effectuée en seulement une ligne de code grâce à cette librairie.
- `LibGit2Sharp`, une librairie que nous avons utilisé pour avoir l'intégration avec le logiciel de contrôle de versionnage (VCS pour Version Control System) Git. Il s'agit *de facto* de la librairie officielle pour gérer des dépôts Git en utilisant le framework `.NET Core`

SCFE a été codé en utilisant l'environnement de développement intégré (IDE) Rider, et a été testé sur Windows, Mac OS ainsi que Linux via le système WSL sur Windows.

Pour distribuer l'application, nous avons choisi plusieurs solutions:

- Un installateur sur Windows, avec le framework `.NET Core 3` inclus
- Une version complète pour Mac OS et Linux avec le framework `.NET Core 3` inclus
- Une version légère pour Windows qui n'inclue que les fichiers `.dll` de base et un `.exe` pour lancer l'application. Cette version requiert qu'une version correcte de `.NET Core` soit installée sur la machine

2 Evolution du projet

Cette section couvrira la progression du développement de l'application. Elle sera découpée en étapes (avant la première soutenance, entre la première et la seconde...) et donnera des détails pour chaque étape. Voici la progression pour chaque tâche que nous avions prévu dans notre cahier des charges.

Task	Soutenance 1	Soutenance 2	Soutenance finale
Interface de base	70%	90%	100%
Modes de navigation	20%	70%	100%
Gestion de fichiers	30%	70%	100%
Entrée utilisateur	40%	50%	100%
Intégration	0%	30%	100%
Site web	0%	50%	100%
Documentation	20%	50%	100%

- "Interface de base" correspond au travail de bas niveau sur l'interface utilisateur, au développement de composants de base et de la librairie Viu sur laquelle SCFE est basée.
- Modes de navigation correspond à l'implémentation des différents modes qui peuvent être utiliser pour naviguer à travers l'application.
- Gestion de fichiers correspond à l'implémentation d'opérations sur les fichiers.
- Entrée utilisateur correspond à la gestion et au routage des appuis sur les touches du clavier de l'utilisateur
- Intégration correspond à l'intéropérabilité avec les applications, outils et systèmes existants
- Site web correspond au développement du site web
- Documentation correspond à l'écriture de la documentation pour notre

application

2.1 Avant la première soutenance

L'objectif de cette période était d'habituer tout le monde aux différents outils avec lesquels ils auraient à interagir au cours des prochains mois, ainsi que de créer les différentes bases dont nous avions besoin très tôt : la bibliothèque Viu et le système de fichiers que nous voulions mettre en place étaient de loin les plus importantes étapes.

2.1.1 Interface de base

Pour créer une application console, nous avions besoin d'un moyen simple d'afficher les composants et de gérer les entrées exactement comme nous le souhaitions. Malheureusement, presque toutes les librairies C# que nous avons pu trouver manquaient de quelque chose de crucial : c'était souvent des problèmes de flexibilité ou de compatibilité multi-plateforme qui rendaient les bibliothèques inutilisables dans notre projet.

Pour rappel, la compatibilité multiplateforme était l'une de nos priorités dans ce projet. A l'origine, deux des quatre membres de l'équipe étaient des utilisateurs Mac, mais un autre membre a également remplacé sa machine Windows par un Macbook. Avec 75% de l'équipe utilisant activement le système Mac OS, il était évident que la compatibilité multi-plateforme *devait* être présente.

Pour les exemples que nous avons mentionnés dans notre cahier des charges :

- La bibliothèque `gui.cs`, bien que parfaitement fonctionnelle, était aussi extrêmement peu claire, et pas assez flexible. Nous avions également des doutes quant à ses capacités multiplateformes.

- La bibliothèque `CursesSharp` était tout simplement trop bas niveau pour être utilisée efficacement, et a également ajouté de nombreuses préoccupations concernant la portabilité et les capacités multiplateformes de notre code, la librairie nécessitant des liaisons très spécifiques aux fonctions systèmes qui diffèrent selon la plateforme (Mac OS/Linux ou Windows).

C'est pourquoi nous avons décidé de créer notre propre bibliothèque, basée uniquement sur l'API `Console` disponible dans `.NET`. Il s'agit d'un ensemble de fonctionnalités simples, sans prise de tête et (pour la plupart) indolores qui ont fait leurs preuves. Le seul inconvénient était que ce système était un peu lent par rapport à d'autres méthodes qui appelaient directement les bibliothèques système.

Ce sous-projet de SCFE est l'un de ses composants les plus importants : afin de répondre aux exigences de flexibilité et de performance que nous avions dans le cahier des charges, beaucoup de travail a dû être fait pour le rendre aussi fluide que possible. La bibliothèque a reçu un nom, `Viu`, et a été complètement séparée du code de l'application elle-même. L'idée était que "SCFE" dépendait de "Viu", et non l'inverse. Bien que cette séparation puisse être considérée comme un éloignement par rapport à l'objectif initial du projet, c'est tout le contraire : la manière dont nous avons construit `Viu` facilite la mise en œuvre de fonctionnalités pour les utilisateurs de SCFE.

La bibliothèque `Viu` est fortement inspirée de la librairie Swing en Java : le système, la hiérarchie des composants et les stratégies de mise en page s'apparentent à Swing, mais aucun code n'en a été extrait, et, lorsqu'on crée des interfaces avec `Viu`, seules certaines fonctions partagent des caractéristiques avec le système Java, mais la librairie reste un travail totalement original.

Les principales idées développées tout au long de cette première période

ont été la création de composants visuels (texte, champs de texte, tableaux, boutons...) ainsi que la mise en œuvre de différentes stratégies de mise en page. Tout cela a été rendu plus difficile par le fait que l'interface a toujours une taille variable et que les stratégies de mise en page devaient être suffisamment flexibles pour que nous n'ayons jamais à toucher du code de mise en page de bas niveau et à manipuler les coordonnées de position directement lorsque nous construisons SCFE sur `Viu`.

Grâce à cela et à une légère parallelisation de processus (multithreading) pour récupérer le moment où la fenêtre est redimensionnée, les composants se redimensionnent dynamiquement à mesure que l'espace autour d'eux change.

`Viu` inclut de multiples "stratégies de mise en page", ce qui nous permet d'afficher les composants exactement comme nous voulons qu'ils soient. Toutes les stratégies sont capables de placer intelligemment des composants, en s'assurant toujours qu'ils s'intègrent parfaitement, en créant des retours à la ligne dynamiquement si nécessaire. Les stratégies illustrées dans la figure 1 sont :

- La Border Strategy (stratégie bordure), qui place quatre composants à chaque bord de la fenêtre et un au milieu
- La Line Strategy (stratégie ligne), qui organise les éléments dans une ligne verticale ou horizontale
- La Flow Strategy (stratégie flux), qui place les composants les uns après les autres, faisant des retours à la ligne comparables à du texte si nécessaire, bien que les composants qui peuvent être placés dans une stratégie flux ne soient pas nécessairement du texte et peuvent être n'importe quel élément de `Viu`.

Les composants les plus utiles ont été créés, y compris des textes simples, des champs de texte (qui écoutent les entrées des utilisateurs), des boutons

et des tableaux entre autres. Comme pour les stratégies de mise en page, elles peuvent toutes se redimensionner dynamiquement. Les tables peuvent même redimensionner chaque colonne individuellement avec des largeurs différentes pour leur contenu (à condition que ces colonnes aient plusieurs tailles disponibles) afin de se développer ou de se réduire.

Pour plus de flexibilité, et si nous souhaitons nous éloigner de l'API de base de la console pour quelque chose qui nous permet d'avoir un contrôle plus fin sur la sortie, `Viu` utilise sa propre couche d'abstraction au-dessus de tout code directement lié à la console. Les composants n'appellent jamais directement l'API `Console`, mais seulement notre couche d'abstraction, dont l'implémentation est donnée par un simple transfert vers l'API de base `Console`.

Les composants `Viu` ont été construits par Matthieu, qui avait de l'expérience avec le système Swing (d'où la ressemblance), pour le code qui définit les composants, l'apparence générale des composants et la hiérarchie générale du système `Viu`. Le côté entrée utilisateur de `Viu` est décrit dans sa propre tâche.

2.1.2 Modes de navigation

Nous avons construit `Viu` avec très peu de raccourcis disponibles dès le départ. Nous n'avons ajouté que les touches essentielles évidentes (par exemple, les touches fléchées et la touche Entrée). Quelques raccourcis supplémentaires ont été ajoutés, en utilisant les systèmes Input Map et Action Map décrits dans la section Entrée utilisateur. Pour les Action Map, l'essentiel du travail a consisté à lier quelques raccourcis aux noms d'actions en mode NAV, ainsi qu'à préparer le tableau utilisé dans le prototype pour recevoir des Input Map en constante évolution. Ceci n'a pas été entièrement mis en œuvre car cela n'était pas nécessaire à ce stade, mais s'est avéré être un test assez complet pour le système d'entrée, pour pouvoir voir à quel point il serait flexible.

Les raccourcis existants ont été liés à leur nom d'action correspondant, dans un dictionnaire, prêt à être implémentés dans l'application principale pour quand elle serait hors de l'état de simple prototype. Cette tâche a été réalisée par François.

2.1.3 Gestion de fichiers

Bien que la gestion de fichiers soient une partie essentielle de SCFE, elle était loin d'être une priorité pour la première période. Nous avions prévu de nous concentrer principalement sur ce qui est devenu `Viu`, en faisant tout le reste plus tard.

Ainsi, François a codé une représentation orientée objet du système de fichiers. Cette représentation est entièrement basée sur les différentes classes fournies par `.NET`. Malheureusement, ces classes fournies n'utilisaient que des méthodes statiques et des chaînes de caractères, et n'avaient pas assez de fonctionnalités pour satisfaire nos besoins. Notre implémentatio Cela signifiait qu'au lieu d'un code qui aurait l'air plutôt explicite comme `monFichier.AvoirNom()`, nous aurions eu un code beaucoup moins clair comme `CheminsDeFichier.AvoirNom(monCheminDeFichier)`. De plus, les erreurs envoyées par les classes `.NET` étaient soit non explicites, soit imprévisibles.

Le fait d'avoir notre propre implémentation nous a également permis de copier et coller des fichiers ou même des dossiers entiers en utilisant une seule fonction dans la représentation des fichiers. (par exemple `monDossier.CopierVers(autrePart)`). C'était une fonctionnalité qui était étonnamment absente des classes `.NET` de base.

De plus, les méthodes implémentées dans cette tâche telles que "Copier" ou "Déplacer" utilisent les classes System.IO telles que la classe `Fichier` de SCFE

peut représenter soit un fichier, soit un dossier. Cette propriété spécifique était particulièrement difficile à gérer lors de la mise en œuvre, car chaque cas devait être pris en compte et impliquait donc la nécessité de reconnaître le type de fichier (dossier ou fichier) qui était traité. `.NET` a deux classes spécifiques pour traiter soit les fichiers soit les dossiers, ce qui rend impossible de dire simplement “copier cette chose là”. Il faudrait dire “si c’est un dossier, copiez le là-bas, si c’est un fichier, copiez le là-bas”, ce qui ne serait pas pratique car les opérations sur les fichiers sont essentielles pour notre application.

Notre implémentation fusionne donc les fonctions `System.IO.File` et `System.IO.Directory`, avec quelques fonctionnalités supplémentaires, comme la détermination du dossier dans lequel se trouve un fichier, ou d’autres raccourcis pratiques.

Il est important de noter que les fonctionnalités de la classe Fichier n’ont pas été pleinement utilisées à ce stade : elles l’ont été uniquement pour afficher le contenu des fichiers. Nous avions l’intention d’utiliser plus largement ce qui a été codé pour la deuxième présentation.

François était responsable de l’ensemble de la tâche, avec l’aide de Matthieu pour corriger quelques bugs et utiliser correctement les fonctions `.NET`, puisqu’il fallait plonger dans la documentation fournie par Microsoft pour traiter correctement chaque cas.

2.1.4 Entrée utilisateur

L’objectif principal de cette tâche pour la première période était de fournir une interactivité pour `Viu` : réagir aux entrées clavier et gérer les états de sélection de tous les composants.

Un système de sélection (aussi appelé “focus” ou ciblage) a été ajouté, qui

permet à l'utilisateur de naviguer entre les différents éléments de l'interface et dans la hiérarchie des composants en utilisant les touches fléchées (ou d'autres raccourcis clavier personnalisés). Cela a été fait grâce à l'utilisation intelligente des interfaces C# sur tous les composants, qui peuvent déclarer le fait qu'ils peuvent être ciblés et gérer les pressions de touches par la simple mise en œuvre d'une interface. Le système de mise au point peut être comparé à la logique des applications normales qui leur permet de déterminer le composant suivant à sélectionner en appuyant sur "Tab" ou "Shit+Tab".

Pour créer ce système de sélection, nous avons simplement étendu les stratégies de mise en page pour inclure également la logique qui indique quel composant doit être le prochain à être ciblé lorsque l'utilisateur va à gauche, à droite, en haut ou en bas.

Un autre aspect de la tâche de saisie de l'utilisateur était le système InputMap/ActionMap. Encore une fois inspiré du système Swing du langage Java, cela nous permet de séparer les raccourcis des actions. L'idée est que les `InputMap`s fournissent une liaison entre les pressions de touches et les noms d'actions, tandis que les `ActionMap`s fournissent une liaison entre les noms d'actions et l'implémentation de l'action réelle. Ce système fournit une couche "tampon" fantastique entre la touche que l'utilisateur presse et ce que le programme fait. C'est le système exact qui a été utilisé pour la tâche des modes de navigation pour la deuxième période pour passer d'une gestion de raccourcis à l'autre sans changer fondamentalement les actions réelles elles-mêmes.

Un diagramme (en anglais) décrivant le processus se trouve à la figure 2.

Enfin, quelques composants entièrement basés sur l'interaction de l'utilisateur ont été ajoutés, à savoir des champs de texte et des boutons. Celles-ci ont été entièrement réalisées sur mesure en utilisant des touches individuelles, car nous ne pouvions pas vraiment nous fier à la façon "régulière"

de lire le texte depuis la console. Les champs de texte fournissent une implémentation importante de raccourcis qui sont courants dans les éditeurs de texte afin de simplifier l'utilisation du composant. Parmi les exemples d'appui sur ces touches, citons la touche Supprimer ou les raccourcis clavier Ctrl+Flèche (pour passer d'un mot à l'autre au lieu de passer d'une lettre/symbole à l'autre uniquement).

Des essais intensifs pour cette tâche ont été cruciaux pour s'assurer que le système d'entrée était robuste.

Mathieu était responsable de la plus grande partie de la tâche, implémentant le système d'entrée (sous la supervision de Matthieu afin qu'il s'intègre parfaitement dans la hiérarchie des composants existante) avec ActionMap et InputMaps ainsi que le système de sélection (toujours avec l'aide de Matthieu), et Rakhmatullo était responsable de la partie "réaction" de quelques composants, y compris les boutons (par exemple effectuer une action prédéterminée quand on appuie sur un bouton).

2.1.5 Intégration

Rien n'a été fait spécifiquement pour cette tâche, comme cela était prévu dans le cahier des charges. Elle représentait une partie plus avancée de l'application et allait bien au-delà de la simple étape de "pose des bases" que nous étions en train d'effectuer pour la première période.

2.1.6 Site Web

Une fois de plus, peu de choses avaient été faites pour le site Web à cette période, l'objectif étant surtout de préparer une base pour toutes les autres tâches. Mathieu a recueilli quelques informations et a fait des maquettes afin

de gagner du temps pour la deuxième période.

2.1.7 Documentation

La majeure partie du travail effectué pour cette partie consistait simplement à recueillir les intentions du cahier des charges pour en faire un format lisible par l'utilisateur. Ceci est fait dans le langage Markdown pour le moment, mais sera publié sur le site web plus tard.

Rakhmatullo n'ayant jamais touché au format Markdown auparavant, il s'est penché sur la question et sur la façon de le faire fonctionner pour nous.

2.1.8 État général

A la fin de la première période, nous avions accompli beaucoup de choses : nous avions une base solide sous la forme de `Viu` ainsi que des prototypes qui présentaient les différents composants que nous avions réalisés. L'objectif était d'avoir quelque chose de prêt pour une transition en douceur vers une fonctionnalité complète.

Un prototype de l'application a été réalisé et se trouve sur la figure 3. Bien qu'il ne "fasse rien", il était possible de monter et de descendre dans l'interface et de sélectionner des fichiers, mais c'était tout. Il s'agissait plus d'une démonstration des capacités de la bibliothèque Viu que d'un produit réel.

2.2 Entre la première et la deuxième soutenance

Maintenant que nous avions une base pour l'application, y compris une bibliothèque faite maison pour afficher les interfaces utilisateur dans la console, le moment était venu de créer l'application par-dessus tout cela.

2.2.1 Interface utilisateur de base

Lors de la construction de l'application principale, nous avions rencontré quelques bugs inattendus avec le code que nous utilisions pour construire des interfaces utilisateur graphiques dans la console. L'objectif pour cette période était, logiquement, de les corriger, ainsi que d'ajouter des fonctionnalités ici et là. Certains composants sont devenus plus intelligents avec une logique qui gère correctement le fait d'être "écrasé". Les composants se comporteraient mal s'ils n'avaient pas assez d'espace autour d'eux : chaque composant attend et demande normalement une certaine quantité d'espace qui lui sera attribué. Si la console était trop petite, il était impossible pour le composant de s'afficher correctement, et un certain nombre de problèmes pouvaient survenir : soit le texte "débordait" sur la ligne suivante, soit l'application entière se plantait avec une erreur plutôt inquiétante (mais assez stupide). Ceci a été corrigé et la plupart des composants supportent maintenant l'écrasement et peuvent afficher une ellipse (...) à un endroit configurable : les chaînes peuvent être coupées à gauche (....comme ceci), à droite (comme ceci...) ou au centre (comme...ceci), selon ce qui a du sens. Un autre composant qui a été amélioré est le composant de tableau (qui est utilisé comme liste de fichiers dans SCFE) : il est maintenant possible de faire défiler la liste s'il n'y a pas assez de place vertical pour l'afficher.

Une autre amélioration apportée au cours de cette période a été l'introduction du multithreading approprié à l'interface utilisateur. Dans les

tests précédents, toutes les actions effectuées sur l'interface étaient effectuées à partir de plusieurs processus en même temps. Ce type de "concurrence" (modifier la même valeur depuis plusieurs processus en parallèle) pourrait être extrêmement problématique à l'avenir et entraîner des plantages sous Linux et Mac OS. Durant cette période, tous les composants devaient maintenant être affichés sur un seul processus (que nous appellerons le thread graphique, thread étant le mot exact pour parler de ce type de processus), ce qui signifie que les opérations liées à l'affichage et au rafraîchissement de l'interface utilisateur étaient toutes effectuées sur un seul processus, éliminant beaucoup de bugs que nous avions. La façon de procéder consistait simplement à limiter l'accès à la couche d'abstraction développée dans la première période : les composants ayant besoin d'accéder à cette couche pour réellement afficher les choses dans la console, nous avons simplement limité la disponibilité de ladite couche aux opérations effectuées dans le thread graphique.

L'interface elle-même a été entièrement construite en utilisant `Viu` : un excellent test pour la base qui est devenue plus solide et plus stable en conséquence. Cela faisait partie de cette tâche, mais consistait surtout à tout assembler comme des briques Lego.

Le travail sur cette tâche a été entièrement réalisé par Matthieu.

2.2.2 Modes de navigation

Deux des trois modes prévus ont été mis en œuvre : le mode "NAV"igation ainsi que le mode "SEA"rch pour la recherche.

Le mode `NAV`, qui est plus proche des explorateurs de fichiers graphiques habituels, consiste à utiliser les touches fléchées et les touches HJKL pour se déplacer dans l'interface. De nombreux raccourcis ont été implémentés pour effectuer des actions sur les différents fichiers. Quelques exemples d'opérations

entièrement mises en œuvre et fonctionnelles pour la deuxième soutenance :

- Appuyer sur R pour renommer un fichier
- Appuyer sur N pour créer un nouveau fichier
- Appuyer sur Maj+N pour créer un nouveau dossier
- Appuyer sur la touche Suppr pour supprimer un fichier
- Appuyer sur C pour copier un fichier, X pour couper un fichier et V pour coller un fichier

Afin de rendre l'application plus sécurisée, les opérations destructives ne sont pas effectuées (par exemple, coller lorsqu'un fichier existe déjà avec le même nom dans le dossier de destination), montrent une erreur, ou demandent une confirmation. De cette façon, la suppression exige que l'utilisateur tape "yes" et appuie sur Entrée dans la zone de texte pour s'assurer qu'il veut supprimer le(s) fichier(s) qu'il a sélectionné.

Le mode `SEA` a aussi ces raccourcis, bien qu'ils nécessitent l'utilisation de la touche Ctrl (par exemple, alors que vous pouvez simplement appuyer sur R pour renommer un fichier en mode `NAV`, vous devez appuyer sur Ctrl+R en mode `SEA`). En effet, les touches de lettres ordinaires ne redirigent pas vers les actions habituellement attendues : elles sautent plutôt vers la zone de texte en bas de l'interface et permettent à l'utilisateur d'entrer des lettres pour rechercher des fichiers spécifiques. L'utilisateur peut alors :

- Appuyer sur Entrée pour directement ouvrir un fichier ou un dossier si un seul élément a été trouvé
- Si plusieurs fichiers correspondent à la recherche, appuyer sur Entrée resélectionne la zone des fichiers, ce qui permet à l'utilisateur de choisir quel fichier il veut vraiment
- Appuyer sur Echap annule la recherche, et vide la zone de texte

Le mode `SEA` correspond plus à ce que les utilisateurs de la console attendent un accès rapide aux fichiers en utilisant les noms de fichiers. Il est à noter que la recherche effectuée n'est pas récursive : seuls les fichiers du dossier courant sont recherchés. En effet, l'intérêt du mode `SEA` est de naviguer le plus rapidement possible dans les fichiers en entrant leur nom, et la recherche dans toute l'arborescence des fichiers serait extrêmement fastidieuse et ralentirait l'application dans son ensemble. Afin d'éviter cela (et aussi d'éviter la confusion à l'apparition de fichiers aléatoires se situant dans des dossiers enfouis profondément émergeants dans une simple recherche), entrer du texte en mode `SEA` ne recherche que dans le répertoire courant. De cette façon, il est préférable de considérer le mode `SEA` comme un "filtre d'affichage des fichiers" plutôt qu'un mode de recherche lourde et complète.

Presque toutes les actions décrites ci-dessus sont prises en charge pour plusieurs fichiers : appuyer sur Espace ou S permet à l'utilisateur de sélectionner des fichiers, et la plupart des actions agissent en conséquence (par exemple, copier plusieurs fichiers dans le presse-papiers).

Tout cela a été implémenté en utilisant le système d'Input Map (transformer les touches de clavier en noms d'action) et Action map (transformer les noms d'action en véritables fonctions qui exécutent l'action) qui est décrit dans la partie concernant les progrès réalisés avant la première soutenance. Nous disposions maintenant d'un système très flexible qui est capable d'échanger à chaud les correspondances entre raccourcis et noms d'action tout en laissant les actions réelles intactes.

De plus, des actions comme le collage ou la suppression peuvent prendre beaucoup de temps. Afin de les accompagner correctement, nous avons utilisé le multithreading pour effectuer l'opération sur un processus séparé tout en permettant aux autres processus de gérer librement les entrées de l'utilisateur.

Le processus dans lequel la suppression se produit indique alors au processus graphique soit d'afficher un message, soit de réafficher le dossier entier si le contenu a changé.

En passant, il est intéressant de noter qu'au sein de l'équipe de développement, nous avons chacun nos propres préférences sur la façon de naviguer dans SCFE : certains préfèrent le mode `SEA` pour un accès rapide à la plupart des fichiers tandis que d'autres préfèrent le mode `NAV` pour la facilité d'utilisation et un accès rapide aux actions !

Ce travail a été fait par Matthieu et François. Puisque Matthieu était principalement responsable de la tâche de l'interface utilisateur de base et François de la tâche de gestion de fichiers, Matthieu s'occupait de lier le mode de navigation aux composants de l'interface ainsi qu'aux éléments les plus complexes comme le multithreading, tandis que François s'occupait de lier les différentes actions à notre système de gestion de fichier. Toutes les actions ont deux "branches" : montrer à l'utilisateur qu'une action est en cours ou s'est produite, et effectuer l'action sur les fichiers. Matthieu s'est occupé de la première, tandis que François s'est occupé de la seconde.

2.2.3 Entrées/sorties de fichiers

Afin d'avoir un affichage plus complet des différentes propriétés de chaque fichier, l'implémentation actuelle devait pouvoir obtenir plus d'informations à partir des fichiers, par exemple la date de dernière modification.

Le système avait également besoin de trier et de filtrer les fichiers pour plusieurs raisons :

- L'ordre des fichiers retournés par les fonctions de `.NET` n'est ni stable ni pratique, et peut être assez chaotique pour les gros dossiers.

- *TOUS* Les fichiers sont affichés par défaut, ce qui signifie que les dossiers du système qui ne sont même pas lisibles par qui que ce soit d'autre que le système étaient visibles, ce qui a fini par ajouter plus d'encombrement à l'interface utilisateur et causer plus de confusion quant aux fichiers qui étaient accessibles et utiles.

Nous nous sommes donc retrouvés avec un système assez souple de filtres re-dirigeables et de "triage intelligent".

- Le filtre de base pour afficher les fichiers dépend de quelques variables, y compris si les fichiers cachés doivent être affichés ou non, ou si nous sommes en mode `SEA` et devons appliquer les termes de recherche ou non. La détection des fichiers cachés se fait à la fois à la manière Windows et Linux/Mac OS : le fichier peut avoir l'attribut "Caché" (comme sous Windows) ou peut commencer par un point (comme sous Linux et Mac OS).
- **Les conditions de tri que nous utilisons tiennent compte de quelques attributs** d'abord si le fichier est un dossier ou un fichier réel, et ensuite les noms des fichiers. Ceci n'est pas personnalisable pour le moment, mais nous pourrions permettre à l'utilisateur de choisir s'il veut trier par taille, date de modification, ou autre.

Des modèles de couleurs ont également été mis en œuvre. Ils consistent simplement à différencier les fichiers (en blanc) des dossiers (en vert) et les fichiers cachés (plus foncés) des fichiers ordinaires (teintes normales).

L'implémentation des fichiers s'est dotée d'un nouveau système de représentation relative des chemins d'accès aux fichiers, qui est expliqué dans la partie "Intégration".

Le travail sur cette tâche a été fait par François.

2.2.4 Entrée utilisateur

La zone de texte au bas de SCFE est le moyen de base pour l'utilisateur d'interagir avec l'application. Il est basé sur le même composant de zone de texte qu'auparavant, et la plupart du travail effectué cette fois-ci a été de correctement sélectionner et désélectionner afin d'éviter que l'utilisateur n'y entre du texte lorsque ce n'est pas prévu.

Lorsque la zone de texte reçoit une touche "Entrée" ou reçoit une nouvelle lettre, elle effectue des actions qui dépendent de l'état de l'application : si une action est en cours (par exemple, l'utilisateur entre le nouveau nom d'un fichier car il a déclenché la séquence "renommer" en appuyant sur "R"), alors la zone de texte redirige l'entrée vers le gestionnaire d'action actuel, qui est différent pour chaque action. Si aucune action n'est en cours, la zone de texte soit ne fait rien (en mode `NAV`), soit filtre la vue du fichier (en mode `SEA`) comme n'importe quelle recherche le ferait. En mode `NAV`, la zone de texte n'est généralement pas accessible, et si elle l'est, elle n'a pas d'action réelle (sauf lorsque des actions comme le renommage sont en cours bien sûr).

De plus, le "système d'entrée" (c'est-à-dire les InputMap et ActionMap) a été relié à la table des fichiers de l'application durant cette tâche, permettant à l'utilisateur d'effectuer des actions avec son clavier. Cependant, le but de cette tâche n'était pas de définir les différents raccourcis, car il s'agissait d'un travail relevant des modes de navigation.

Ce travail a été fait par Mathieu.

2.2.5 Intégration

Afin de réduire l'encombrement de l'interface et d'éviter des chemins de fichiers extrêmement longs, dans certains cas, le chemin peut être

"relativisé". Le seul cas actuellement implémenté est lorsque l'utilisateur se trouve dans un dossier qui est un sous-dossier de son dossier personnel. Par exemple : Sous Windows, si mon nom d'utilisateur est "MonNom", au lieu de "C:/Users/MonNom/MonDossier", SCFE affiche "/MonDossier", ce qui est plus utile (et utilise un standard bien connu qui est de remplacer le dossier personnel de l'utilisateur par") Le nom du dossier est déduit de ce que le système d'exploitation indique en tant que répertoire personnel, il fonctionnera donc pour tout nom d'utilisateur sous les systèmes d'exploitation correctement pris en charge par `.NET Core`.

Parce qu'un explorateur de fichiers qui ne peut pas ouvrir des fichiers est tout à fait inutile, SCFE fournit une intégration légère avec le système sous-jacent en étant capable d'ouvrir des fichiers. Ceci utilise l'action par défaut de l'OS et ouvre la même application qui s'ouvrirait si l'utilisateur double-clique sur le fichier dans l'explorateur de fichiers normal du système.

Cela a été fait par Rakhmatullo.

2.2.6 Site Web

À ce moment, alors que nous avions quelques idées pour le site Web, Mathieu s'est chargé de la création et de la maintenance du site Web. Le site Web est disponible à l'adresse suivante :

https://salamanders.dev/

A ce moment, il ne disposait que d'informations de base sur le projet : ce qu'il est, qui sont les membres de l'équipe et quelques espaces réservés pour pouvoir y ajouter du contenu plus tard. Il est totalement "responsive" (les éléments bougent et s'adaptent lorsque la taille de la fenêtre change) et fonctionne sur mobile. A ce stade, il était prêt à accueillir davantage de contenu.

Son design est proche de celui que nous avons utilisé tout au long des

documents et des présentations : des accents noirs et jaunes avec des logos pour SCFE et Viu. Viu n'avait pas de logo à l'origine, mais comme il était séparé de SCFE dans la base de code, nous avons pensé qu'il pouvait être considéré comme un autre produit de notre équipe. Le logo Viu a été créé pour l'occasion.

Le site Web a été réalisé en HTML et CSS pures (mais propres) à l'aide de la boîte à outils Bootstrap et est hébergé sur OVH. Le certificat HTTPS a également été obtenu via OVH. Cela nous donne un contrôle total sur le contenu du site Web, bien que la création et le téléversement puissent parfois être un peu fastidieux.

La documentation est également hébergée sur le site Web et est bien intégrée au thème actuel. Une page de téléchargements est également présente mais n'avait pas réellement fourni les téléchargements à l'époque : ceux-ci ont été ajoutés pour la dernière soutenance.

Le travail sur le site ainsi que le thème de la documentation ont été entièrement réalisés par Mathieu.

2.2.7 Documentation

L'utilisation de base de l'application a été documentée et comprenait des informations sur l'utilisation simple de l'application dans les manières d'utiliser l'application les plus élémentaires.

La documentation explique clairement l'utilisation des modes `SEA` et `NAV`, en s'assurant d'expliquer quand et où faire quelque chose d'une manière claire afin de ne pas complètement perdre les nouveaux venus dans des détails inutiles. Il fournit également des raccourcis clavier utiles, qui sont également disponibles à partir de l'application.

L'application elle-même dispose également d'une bonne quantité d'informations : l'utilisateur peut ouvrir des "panneaux d'options" spéciaux qui affichent toutes les options disponibles pour un fichier, ainsi que les raccourcis clavier qui permettent d'effectuer l'action. Cette aide est créée dynamiquement, de sorte que si nous voulons avoir des raccourcis clavier personnalisables à l'avenir, les raccourcis affichés dans l'application reflètent les raccourcis clavier choisis - mais aussi pratique que cela soit, l'aide reste minimale, à la fois parce qu'elle ne doit pas interrompre le flux de travail et parce que la console est un mauvais environnement pour la lecture du texte, principalement en raison du manque d'espace et de décorations appropriées des polices.

La documentation qui se trouve à l'extérieur de l'application est écrite en Markdown et est disponible sur le site web dans l'onglet Documentation. Elle est bien sûre convertie en HTML avant d'être publiée, et quelques ajustements manuels au code HTML ont dû être effectués. La documentation en ligne a été faite par Rakhmatullo, tandis que la documentation dans l'application a été faite par Matthieu. Le thème de la documentation sur le site web a été fait par Mathieu.

2.2.8 État général

À ce moment-là, l'application existait vraiment avec presque toutes ses fonctionnalités. Grâce à notre base solide, nous avons été en mesure d'accumuler à un rythme assez rapide toutes les fonctionnalités simples que nous souhaitions. Trois captures d'écran sont disponibles dans les figures 4, 5 et 6, montrant respectivement l'application fonctionnant sous Windows, sous Windows en mode recherche et sous Mac OS. La compatibilité multiplateforme, étant l'une de nos priorités, a fonctionné, mais a eu quelques bugs ici et là qui ont été corrigés dans la période qui a suivi.

De plus, les sites Web et la documentation avaient commencé à prendre forme et à voir le jour. Ils étaient légers, car ils n'étaient pas notre priorité pour cette période, mais ils fonctionnaient bien.

2.3 Entre la deuxième et dernière soutenance

Beaucoup de travail avait été fait entre la première et la dernière défense : à tel point qu'il ne restait plus grand-chose à faire pour la défense finale, mis à part les fonctions les plus avancées, et la correction des bogues habituelle.

2.3.1 Interface de base

La plupart du travail effectué pour cette tâche durant cette période consistait à remanier et à corriger les problèmes qui se passaient dans les coulisses. Quelques fonctionnalités internes supplémentaires ont été ajoutées, principalement pour faciliter le multithreading et le bon aller-retour entre le processus graphique et les autres processus qui s'occupaient de la gestion des fichiers. Un exemple de telles fonctionnalités est une fonction "RequêteSynchronisée" tout-en-un, appelée dans les processus de gestion de fichiers, qui est chargée de bloquer le processus de fichier, de demander à l'utilisateur une entrée, d'attendre cette entrée, de la renvoyer et de libérer le processus précédemment bloqué, sauf si l'utilisateur annule l'action (par exemple, en appuyant sur la touche Echap).

De plus, quelques problèmes importants ont été corrigés pour Viu : à cause des bugs dans `.NET Core 2.2`, qui est le framework que nous utilisions jusqu'ici, nous avons été obligés d'effacer et de réimprimer l'interface utilisateur complète à chaque fois que l'utilisateur fournissait des données par le clavier sur Mac OS et Linux. Nous avons donc rendu Viu compatible avec `.NET Core 3` tout en gardant la compatibilité avec la version 2.2. Nous ne pouvions pas simplement abandonner le support de la version 2.2, car la version 3 n'est disponible qu'en pré-version et devrait être disponible en version stable vers fin 2019.

Une autre fonctionnalité qui a été ajoutée est la possibilité de changer l'ordre des colonnes à travers un fichier de configuration. Ce fichier de configuration,

situé sur `~/.SCFE/columns.txt`, prend la forme du nom des colonnes séparées par des virgules. La mise en page par défaut est `name,git,size,date`. Bien qu'il s'agisse d'un format de fichier de configuration simple, nous comprenons qu'il peut être un peu difficile à modifier pour les débutants, mais nous avons décidé de ne pas créer un écran de configuration complet pour notre application en raison des contraintes de temps et de la surcharge d'une telle fonctionnalité pour un simple réordonnement des colonnes. Nous n'avons pas beaucoup de colonnes disponibles dans SCFE : l'ajout de ce mécanisme de réorganisation n'est qu'une commodité et non une caractéristique particulière du programme.

Matthieu était responsable de l'ensemble de cette tâche.

2.3.2 Modes de navigation

Les modes de navigation ont fait l'objet d'une certaine attention dans cette partie, mais qui est restée légère. Il s'agissait surtout d'ajouter quelques raccourcis supplémentaires et de fournir des liens pour l'intégration de Git. Par exemple, une fonction "Tout sélectionner" et une fonction "Inveser la sélection" ont été ajoutées.

De plus, l'utilisateur peut maintenant changer la méthode de tri des fichiers à l'aide de deux raccourcis : `Maj+S` fait défiler les différentes méthodes de tri disponibles (par nom, puis par extension de fichier, puis par taille, puis par date, puis de nouveau par nom), et `Maj+Q` inverse l'ordre. L'inversion d'ordre ne se contente pas de "retourner la liste de fichiers" : elle le fait de manière plus intelligente. Par exemple, dans le tri des noms (qui est le tri par défaut), les dossiers vont d'abord de A à Z, puis les fichiers vont de A à Z. Dans le tri des noms inversé, les dossiers sont toujours en premier mais vont de Z à A et seulement après les dossiers viennent les fichiers qui vont aussi de Z à A. Ainsi, les dossiers restent toujours en haut, ce qui ne serait pas le cas si l'on inversait

simplement la liste originale.

François et Matthieu ont coopéré à cette tâche. Matthieu a relié les raccourcis actuels à l'interface tandis que François a apporté les modifications nécessaires au système de tri que nous avions déjà en place. Ces modifications sont détaillées dans la section suivante.

2.3.3 Gestion de fichiers

Cette tâche a vu un nouvel ajout que nous n'avions pas prévu dans le cahier des charges parce que nous n'étions pas sûrs de sa faisabilité. L'une des failles majeures de notre application était l'absence d'une fonction de "rechargement à chaud" : si l'utilisateur (ou un autre programme) crée, supprime ou modifie un fichier, il est impossible pour SCFE de le détecter et de se rafraîchir. L'utilisateur devait appuyer manuellement sur le raccourci `Maj+R`, ce qui déclenche un rechargement du dossier.

Nous avons donc ajouté une nouvelle fonctionnalité dans cette période : l'auto-rechargement. Grâce à une fonctionnalité de `.NET` appelée "surveillants de système de fichiersé", nous sommes capables de détecter tout changement dans le dossier qui est actuellement ouvert dans SCFE. Si un changement est détecté, nous rechargeons automatiquement l'interface utilisateur pour afficher la modification, tout en continuant à sélectionner le fichier qui était précédemment sélectionné. Ce processus est très rapide et rend SCFE extrêmement utile dans de nombreuses autres situations. Il permet également de retirer deux pressions de touche de l'utilisateur, celles qui auraient été nécessaires au rechargement manuel du dossier, ce qui est toujours un ajout agréable puisque l'application est axée sur la productivité.

Certaines fonctionnalités ont été modifiées ici et là pour permettre l'ajout de nouvelles options de tri que l'utilisateur peut faire défiler, comme décrit

dans la partie Modes de navigation dans cette période.

La méthode de tri initiale a également été modifiée pour favoriser le tri naturel. La façon originale et classique de trier les noms est d'un point de vue purement alphabétique. Ainsi, les noms “Hello1”, “Hello2” et “Hello10” seraient triés comme “Hello1”, “Hello10” et “Hello2”, ce qui n'est pas logique, puisque 10 est plus grand que 2. La façon logique de trier les chaînes est appelée “tri naturel” et remplace la méthode de tri originale que nous avions en place. De cette façon, le tri est moins déroutant et plus agréable pour l'utilisateur, car nous pensons qu'il est beaucoup plus propre.

Ce travail a été fait par François.

2.3.4 Entrée utilisateur

Il n'y avait pas grand-chose à faire dans cette partie. Le codage du mode `COM`, que nous détaillerons plus loin dans ce document, est basé exactement sur le même mécanisme qui nous a permis de demander aux utilisateurs de fournir des détails pour des opérations comme le renommage de fichiers. Nous avons réutilisé ce système pour permettre à l'utilisateur d'entrer une commande, puis d'acheminer cette commande en fonction de ce qui a été codé dans la partie Intégration.

Une capacité supplémentaire a été ajoutée au composant de champ de texte, que nous utilisons dans SCFE comme boîte de texte disponible à tout moment en bas de l'écran. Il est maintenant capable de masquer son entrée dans le cas où l'utilisateur saisit des informations sensibles : ceci a été ajouté afin de permettre l'intégration avec Git pour demander un mot de passe à l'utilisateur de telle manière que le mot de passe ne soit pas affiché en clair à l'écran.

Certains bogues que nous avons rencontrés avec des touches qui n'étaient

pas détectées ont été corrigés en mettant simplement à jour notre version `.NET Core` vers les pré-versions de la version 3.

Ce travail a été fait par Mathieu.

2.3.5 Outil et intégration du système d'exploitation

A ce stade du projet, l'intégration était *la* priorité. Nous avions un explorateur de fichiers assez basique, maintenant nous avions besoin de fonctionnalités puissantes pour les utilisateurs avancés.

Cependant, comme le but de notre application est d'être un explorateur de fichiers, nous avons décidé de ne pas avoir d'intégration lourde avec chaque outil. L'idée est qu'une intégration légère de l'outil le plus utilisé serait plus que suffisante, et les utilisateurs qui ont besoin d'effectuer des actions plus complexes devraient utiliser l'outil directement au lieu de l'utiliser via SCFE.

Avec cet état d'esprit, nous avons entrepris d'ajouter les deux caractéristiques que nous trouverions les plus utiles : Intégration de Git et un mode COM pour exécuter des commandes dans la ligne de commande.

2.3.5.1 Intégration Git

Git est un "système de contrôle de version" : un outil qui permet aux utilisateurs de gérer leurs fichiers (généralement du code) organisés dans des dépôts, de les versionner (c'est-à-dire d'avoir des versions de fichiers sauvegardées), de naviguer dans l'historique des fichiers facilement et de manière fiable, tout en pouvant partager les fichiers et leurs modifications en ligne sur des sites comme GitHub et GitLab. Git est utilisé à l'EPITA pour soumettre du code pendant les travaux pratiques de programmation, et nous l'utilisons depuis le tout début du projet pour collaborer sur le code. C'est l'outil le plus utilisé

dans sa catégorie, donc l'intégrer dans SCFE serait un avantage évident pour les développeurs.

L'intégration de Git dans SCFE est simple mais extrêmement utile. Elle est basée sur la librairie LibGit2Sharp, et offre à la fois la possibilité de visualiser le dépôt Git et les actions sur celui-ci. Dans un référentiel Git, SCFE charge et cache automatiquement l'état du référentiel et affiche l'état de chaque fichier dans une colonne séparée. Cette colonne n'apparaît que dans un dépôt Git et est automatiquement cachée partout ailleurs.

De plus, quelques actions communes ont été ajoutées : la mise en place et la suppression d'un fichier (qui indique à Git qu'il doit être ajouté à la prochaine sauvegarde), la création d'un commit (c'est-à-dire la création d'une sauvegarde), ainsi que le "push" et la "pull" (c'est-à-dire envoyer et recevoir des sauvegardes depuis un serveur). Ces actions ne sont disponibles qu'à un niveau assez élémentaire, et nous avons opté pour la sécurité. Ces opérations ont différentes variantes qui nécessitent une entrée supplémentaire de la part de l'utilisateur, et Git peut généralement détecter les conflits entre les fichiers : en cas de conflit ou d'entrée supplémentaire, nous informons simplement l'utilisateur qu'une erreur se produit, et nous nous arrêtons là. Ceci fait suite à l'idée que les utilisateurs de SCFE ne devraient pas être dans SCFE pour effectuer des actions compliquées.

La figure 7 donne un exemple de l'utilisation de Git.

Il existe encore une autre façon de lancer des commandes plus complexes à partir de SCFE, sous la forme du mode COM.

2.3.5.2 Mode COM

Le mode COM est le troisième mode et probablement le plus simple : il suffit d'appuyer sur Ctrl+Entrée (ou Maj+M), de taper le nom de la commande

que vous souhaitez lancer comme dans un terminal normal, puis d'appuyer sur Entrée pour lancer la commande. Comme toute autre saisie de texte dans SCFE, il est aussi possible d'appuyer sur Echap pour annuler l'action.

La commande elle-même est redirigée vers un shell (un logiciel de terminal) sous-jacent, qui est PowerShell sous Windows et bash sous Mac OS et Linux. Si le processus écrit quelque chose dans sa sortie (par exemple pour afficher un message), le message est intercepté et acheminé vers l'interface où tous les messages sont généralement affichés sur SCFE.

Ce système fonctionne bien et se fait sur de multiples processus à la fois, ce qui signifie que l'interface principale n'est pas bloquée lorsque la commande est exécutée en arrière-plan. Ce mode est assez basique et ne prend pas en charge les processus qui nécessitent l'intervention de l'utilisateur en cours d'exécution, mais il n'est pas destiné à remplacer un terminal complet. Il peut cependant être utile pour de petites tâches. Par exemple, nous pouvons générer les fichiers d'installation de SCFE depuis SCFE via une seule commande en mode COM.

La figure 8 donne un exemple de l'utilisation de ce mode.

Le travail sur l'intégration Git et le mode COM a été réalisé par Rakhmatullo avec l'aide de Matthieu. Le reste de l'équipe a également aidé à tester les nouveaux ajouts, ce qui était crucial puisque certaines fonctionnalités comme le mode `COM` reposent fortement sur des comportements dépendant de la plate-forme.

2.3.6 Site Web

Même si le site avait bien progressé entre la première et la deuxième défense, il y avait encore du travail à faire. Quelques animations ont été ajoutées et plus de thématiques ont été faites pour une apparence plus propre.

La page de téléchargement a également été enrichie de quelques nouveautés, dont une apparence inspirée de la page d'accueil de l'éditeur Atom (https://atom.io), ainsi que plus de détails sur le logiciel comme les notes de mise à jour et plus encore.

Le travail sur le site a été entièrement réalisé par Mathieu.

2.3.7 Documentation

D'autres parties du projet ont été documentées. Comme dans la période précédente, chaque nouvelle fonctionnalité est affichée dans un panneau spécifique dans SCFE qui affiche toutes les actions disponibles.

La documentation sur le site Web a également reçu quelques améliorations, la plupart documentant les nouvelles fonctionnalités qui sont apparues durant cette période, ainsi que d'autres fonctionnalités qui n'avaient pas encore été documentées alors qu'elles étaient déjà présentes.

La documentation a été faite par Rakhmatulo.

2.3.8 État général

À ce stade, l'application, le site Web et la documentation sont entièrement terminés. Les résultats finaux sont expliqués dans la section suivante.

L'application a également reçu un installateur pour Windows et pour Mac OS, une version légère (qui n'inclut pas le framework `.NET Core` et nécessite donc un téléchargement séparé) pour Windows, et les binaires pour Linux. L'installateur pour Windows a été réalisé par Matthieu, celui pour Mac OS par Mathieu.

3 Résultats finaux

Dans cette section, nous passerons en revue ce qui a été créé pour notre projet et les résultats finaux.

3.1 Une librairie : Viu

Comme nous l'avons mentionné précédemment, nous n'aimions aucune des diverses options qui s'offraient à nous.

Bien que ce ne soit pas le but principal de notre application, nous avons fini par créer une bibliothèque séparée pour créer des interfaces dans la console. La motivation derrière le fait de ne pas simplement réutiliser quelque chose qui existait déjà était que les autres bibliothèques ne nous donnaient pas la flexibilité que nous voulions, qu'elles avaient des problèmes sur des plates-formes autres que Windows, ou qu'elles avaient des bizarreries diverses que nous ne pouvions pas réparer facilement.

Viu fournit un excellent moyen de créer une application dans la console. L'interface est entièrement redimensionnable, multithreadée (elle supporte une exécution en parallèle particulièrement efficace) et possède une couche d'abstraction qui lui permet de fonctionner sur n'importe quel type de sortie et pas seulement une console. En remplaçant simplement cette couche d'abstraction par autre chose, nous pourrions théoriquement écrire l'interface utilisateur sur une imprimante, une interface graphique réelle (bien qu'elle garderait son apparence textuelle), une implémentation de console différente, ou à peu près n'importe quoi d'autre.

Viu dispose d'un système d'entrée puissant que nous avons décrit dans l'évolution du projet. Il nous a grandement aidés dans le processus de création

de l'application elle-même, en nous fournissant un moyen facile de permuter entre les raccourcis des différents modes de SCFE.

3.2 Une application : SCFE

L'application SCFE est devenue exactement ce que nous avions prévu, et plus encore.

L'une des fonctionnalités que nous avons pu ajouter, bien qu'elle n'ait pas été initialement prévue, est un système de rafraîchissement automatique. Lors de l'ajout ou de la modification de fichiers depuis l'intérieur de SCFE, l'application s'est initialement rafraîchie, s'assurant que les fichiers créés étaient affichés. Cependant, ce système ne prenait pas en compte les fichiers qui auraient pu être modifiés d'ailleurs, par exemple à partir d'une autre application ou d'un utilitaire en ligne de commande.

Dans une tentative d'ajouter quelques fonctionnalités supplémentaires, Matthieu s'est penché sur la possibilité d'utiliser la fonction de surveillance du système de fichiers de `.NET`, qui était étonnamment facile à ajouter puisque nous avions déjà implémenté des fonctionnalités similaires. En fin de compte, SCFE prend en charge le rechargement des dossiers à la volée lorsqu'ils sont modifiés depuis un autre endroit.

Quant à toutes les caractéristiques prévues, elles sont toutes là, y compris le système de mode. SCFE a été entièrement construit avec ce système de mode à l'esprit. Il existe trois modes utilisables, suivant une approche "modale" :

- Mode `NAV` (Navigation), pour naviguer facilement entre les fichiers. Les raccourcis sont faciles d'accès dans ce mode, en appuyant une seule fois sur une touche. Le mode `NAV` est conçu pour la productivité dans des dossiers avec un nombre limité de fichiers, et pour les personnes qui

préfèrent l'approche habituelle et plus lente de la navigation dans les fichiers, où on ne peut aller que de haut en bas entre les fichiers.

- Mode `SEA` (Search), pour trouver rapidement un fichier spécifique que vous recherchez. Les raccourcis de ce mode sont les mêmes que ceux du mode de navigation, la différence étant que le mode de recherche exige que la touche Ctrl soit maintenue enfoncée pour tous les raccourcis. En effet, en appuyant sur n'importe quelle touche sans la touche Ctrl, vous rechercherez des fichiers au lieu de déclencher un raccourci. L'utilisateur peut taper le nom d'un raccourci de n'importe où et accéder facilement et rapidement à ce fichier. Cela rend la recherche de fichiers extrêmement efficace et ressemble davantage à ce à quoi les utilisateurs de la console s'attendent.

- Mode `COM` (Commande), pour lancer des commandes plus complexes que celles que l'on entrerait normalement dans un terminal. Les commandes sont parfaites pour lancer des scripts ou d'autres processus qui sont plus techniques ou plus flexibles que ce que l'on peut habituellement faire en SCFE.

Il existe plusieurs façons différentes de travailler avec ce modèle :

- En utilisant uniquement le mode `NAV` avec le menu Options que nous avons implémenté. Ce menu donne accès à toutes les actions disponibles dans une liste et affiche tous les raccourcis associés. C'est un excellent moyen de démarrer avec SCFE et permet aux débutants et à ceux qui ne veulent pas apprendre tout ce qu'il y a à savoir sur l'application d'utiliser l'application dans toute son étendue.

- Utiliser le mode `NAV` avec tous les raccourcis et parfois passer en mode `SEA` pour naviguer dans de grands dossiers. Les raccourcis en mode `NAV` sont

particulièrement rapides à utiliser car ils ne nécessitent pas l'utilisation de la touche Ctrl. Un copier-coller, qui est généralement un Ctrl+C et Ctrl+V est réduit à un appui sur C pour copier et sur V pour coller. Bien sûr, si l'utilisateur oublie un raccourci, il peut ouvrir le menu d'options pour voir to les raccourcis. C'est la méthode utilisée par la plupart des membres de l'équipe.

- Uniquement en utilisant le mode `SEA`. Cela rend la navigation dans les dossiers extrêmement rapide, mais peut être plus pénible pour la mainpour lancer des raccourcis. De plus, vous devez connaître le nom de vos dossiers par cœur afin d'être vraiment efficace. Cette méthode est la plus proche de celle de la console pour naviguer dans les fichiers.

Le mode "COM", qui n'est pas vraiment un mode de "navigation" mais plutôt un mode d'"opération", peut toujours être utilisé en conjonction avec tous les autres modes sans aucun problème et dans tous les flux de travail.

SCFE fournit toutes les opérations régulières que l'on peut attendre d'un explorateur de fichiers, et plus encore :

- Intégration directe légère avec la ligne de commande avec le mode `COM`
- Intégration légère avec Git avec une latence moindre
- Compatibilité avec Windows, Linux et Mac OS avec peu de différences dans les opération
- Rechargement automatique des fichier

Les autres explorateurs de fichier ont rarement les quatre fonctionnalités en même temps, ce qui nous rend très fier du résultat.

Une autre note intéressante à propos de SCFE est que, même s'il est écrit dans un framework qui n'est pas nécessairement réputé pour sa vitesse, il

reste assez rapide et parfaitement utilisable pendant toute la durée de vie de l'application, ce qui est particulièrement important quand on passe par les dépôts Git. L'utilisation des dépôts Git est un peu pénible pour le système, car le fonctionnement du système Git nous oblige à recharger tout l'index du dépôt afin d'obtenir l'état actuel des différents fichiers et de montrer une représentation précise de l'état des fichiers.

3.3 Un site web : salamanders.dev

Avec SCFE, le site Web que nous avons créé, bien que simple, respecte toujours ce que nous considérons comme nos priorités : quelque chose de propre qui va droit au but mais qui ne vous perd pas en simplifications excessives.

Les différents liens de téléchargement et la documentation sont affichés sur le site Web, qui présente un style propre, minimaliste et moderne, tout en conservant notre thème graphique et notre police de caractères de choix (PT Mono). Les téléchargements sont conçus pour toutes les plates-formes que nous ciblons, et incluent un programme d'installation Windows que Matthieu a construit en utilisant l'outil NSIS (Nullsoft Scriptable Install System). NSIS est une solution qui a fait ses preuves depuis 19 ans et qui dispose d'une base d'utilisateurs exceptionnelle : presque tous les installateurs que nous avons aujourd'hui sont construits avec NSIS. De plus, NSIS est totalement gratuit, ce qui nous permet d'éviter d'ajouter des coûts à notre projet.

Le site Web lui-même, hébergé par OVH, est entièrement statique, ce qui est une excellente option car il ne nécessite aucune maintenance, reste flexible, et nous n'avons pas trop à nous soucier de bases de données ou d'autres composants Web plus complexes. Étant écrit en HTML pur, il est un peu lourd à gérer, mais Mathieu a de l'expérience dans l'écriture de sites Web en HTML pur et sait comment les gérer correctement.

4 Opinion général

4.1 Le bon

Dans l'ensemble, nous sommes très satisfaits de ce que nous avons réussi à produire.

Tout ce que nous avons fait était nouveau pour les membres du groupe : par exemple, la gestion d'une équipe était une première pour Matthieu, alors que Mathieu et François avaient très peu d'expérience en codage avant de rejoindre EPITA. Cela nous a donné de nouvelles expériences, ce qui est toujours une bonne chose.

Le projet nous a également permis d'utiliser les outils d'une manière plus avancée : alors que Git n'était utilisé que pour des soumissions simples dans notre cours de programmation, nous l'avons utilisé pour l'intégrer dans une autre application - et nous l'avons également utilisé pour suivre l'évolution du projet.

4.2 Le mauvais

L'un de nos objectifs les plus importants a été la compatibilité multiplateforme. Alors que tout fonctionnait bien sur les machines Windows, les autres membres de l'équipe ont eu beaucoup plus de mal à résoudre les problèmes. Les terminaux sur les plateformes non Windows ont tendance à "attraper" beaucoup de pressions de touches, ce qui rend certaines séquences de touches complètement inutiles. Toutes les séquences de touches qui impliquaient d'appuyer sur la touche Alt ne fonctionnaient tout simplement pas la plupart du temps sur les plateformes non-Windows. Il n'y a pas d'autre solution que de plonger vers une approche beaucoup plus basse pour communiquer avec le terminal afin

d'attraper ces touches, mais cela aurait été beaucoup trop long pour nous.

En fin de compte, les différents menus que nous avons mis en place pour sélectionner des actions à partir d'une liste finissent également par être un dernier recours si un raccourci échoue pour une raison quelconque sur une plateforme non-Windows. Il s'agit d'un problème très imprévisible puisque tous les systèmes auront des problèmes de touches différentes et, lorsque le système sous-jacent n'est pas responsable du problème, c'est le framework `.NET` que nous utilisons qui ne veut pas coopérer avec nous.

Sur un plan plus humain, Matthieu a eu beaucoup de mal à déléguer des tâches au reste de l'équipe, puisqu'il avait plus d'expérience dans ce genre de projet. Les fonctions de codage étaient généralement plus rapides s'il le faisait par lui-même plutôt que de laisser les autres le faire et d'avoir à les expliquer.

4.3 Dernières réflexions

Ce projet était une première pour nous tous, et nous sommes tous très fiers du résultat final. Nous croyons que nous avons atteint ce que nous avions à l'esprit lorsque nous avons commencé et que nous avons réussi à livrer un produit que nous voulons utiliser nous-mêmes. Ce fut une expérience intéressante qui nous a permis de manipuler des outils de développement logiciel en dehors des cours de programmation habituels que nous avons, et une expérience d'équipe plutôt nouvelle.

5 Annexe

Cette section contient des images additionelles et des captures d'écran de notre application

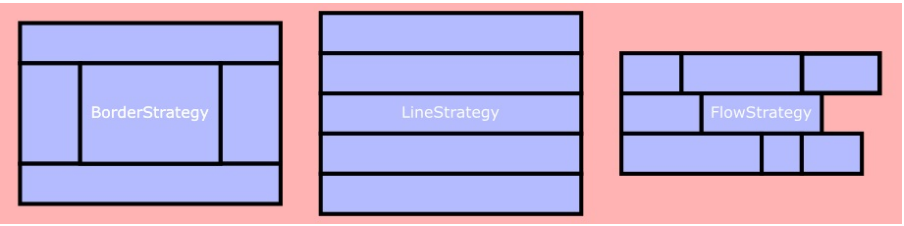

Figure 1: Stratégies de mise en page pour les composants (Border, Line et Flow Strategies)

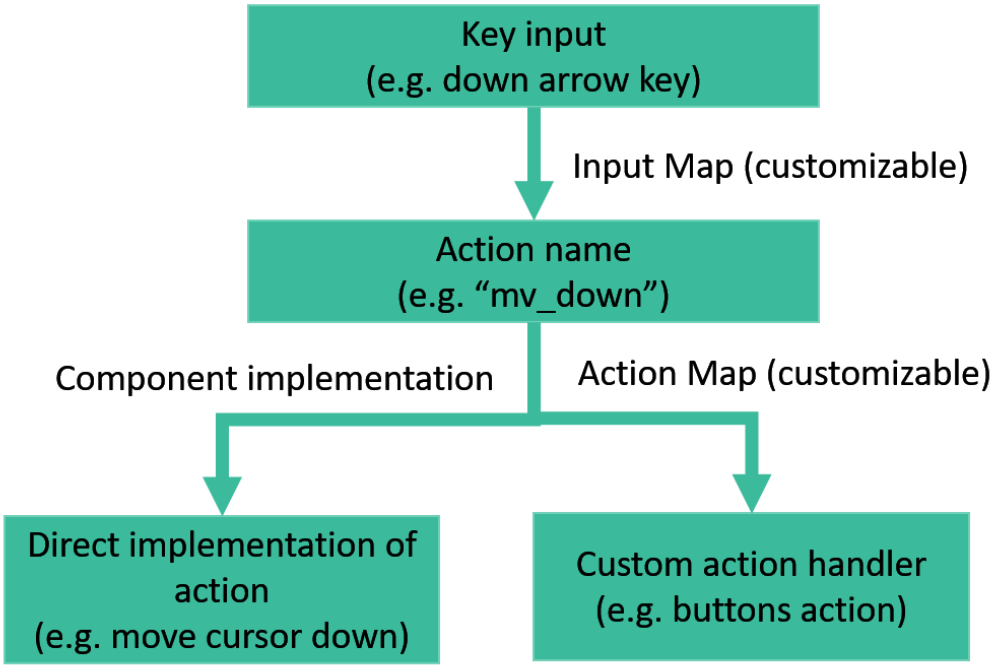

Figure 2: Illustration du fonctionnement des `InputMap` et `ActionMap`

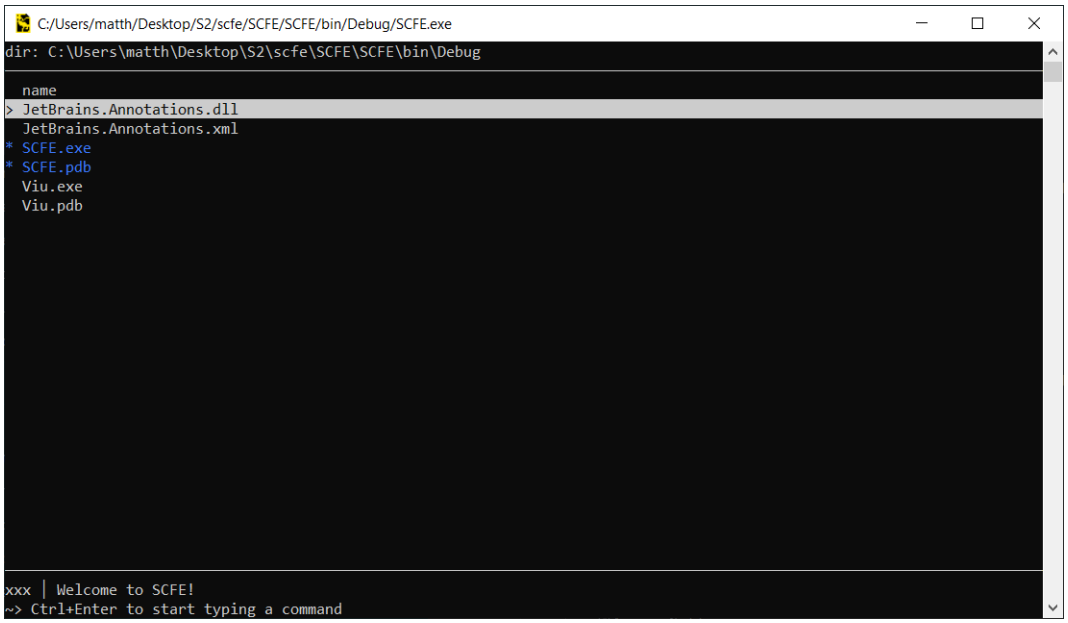

Figure 3: Prototype de l'application (Première soutenance)

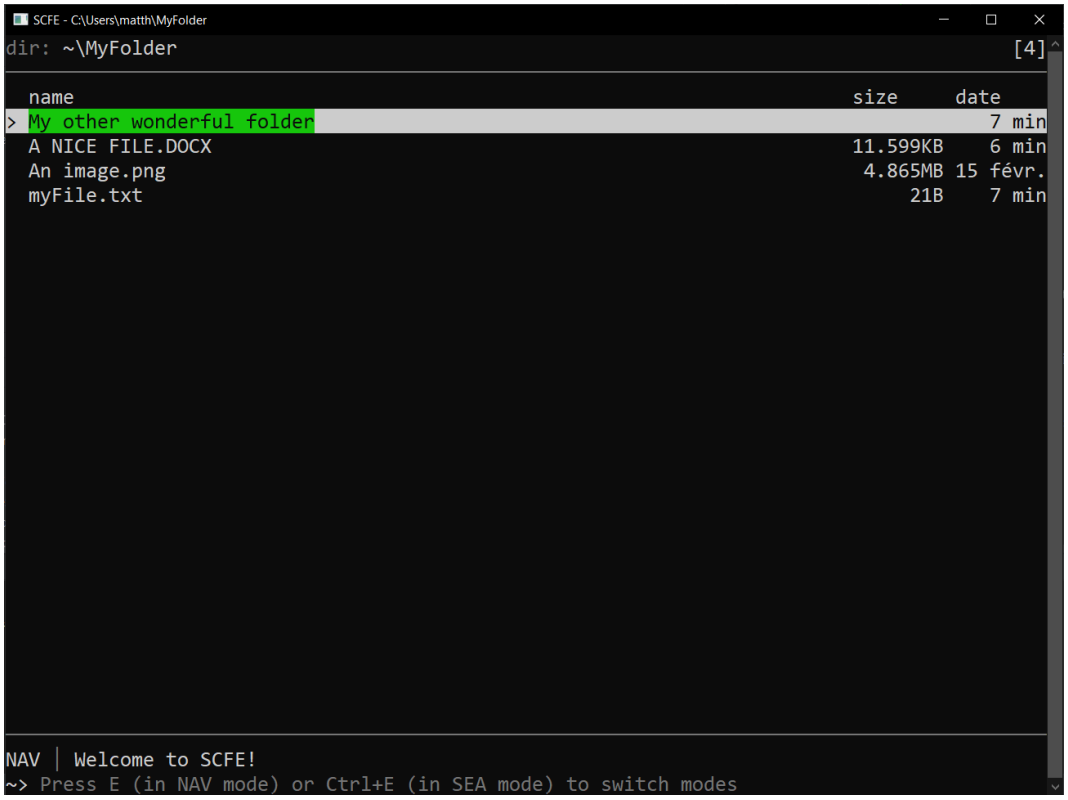

Figure 4: L'application sur Windows (Seconde soutenance)

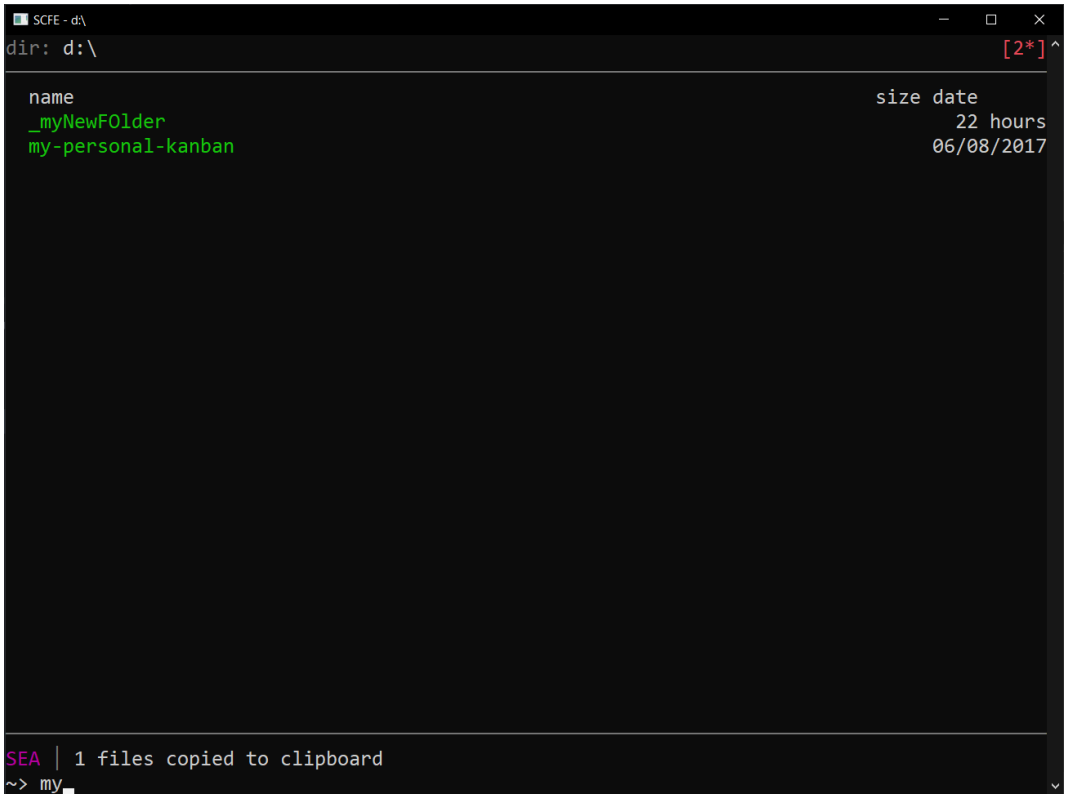

Figure 5: Utilisation du mode recherche (SEA)

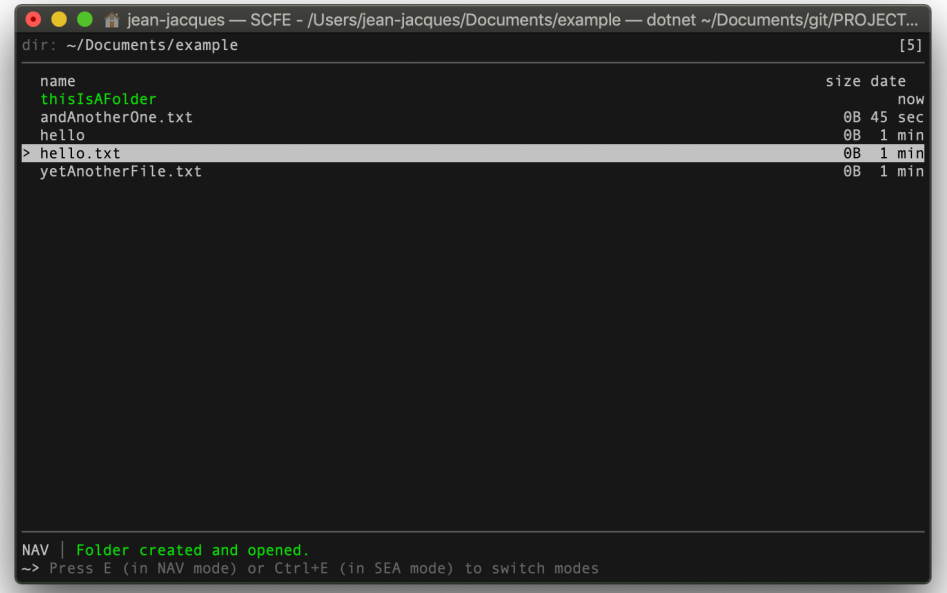

Figure 6: L'application sur Mac OS (Seconde soutenance)

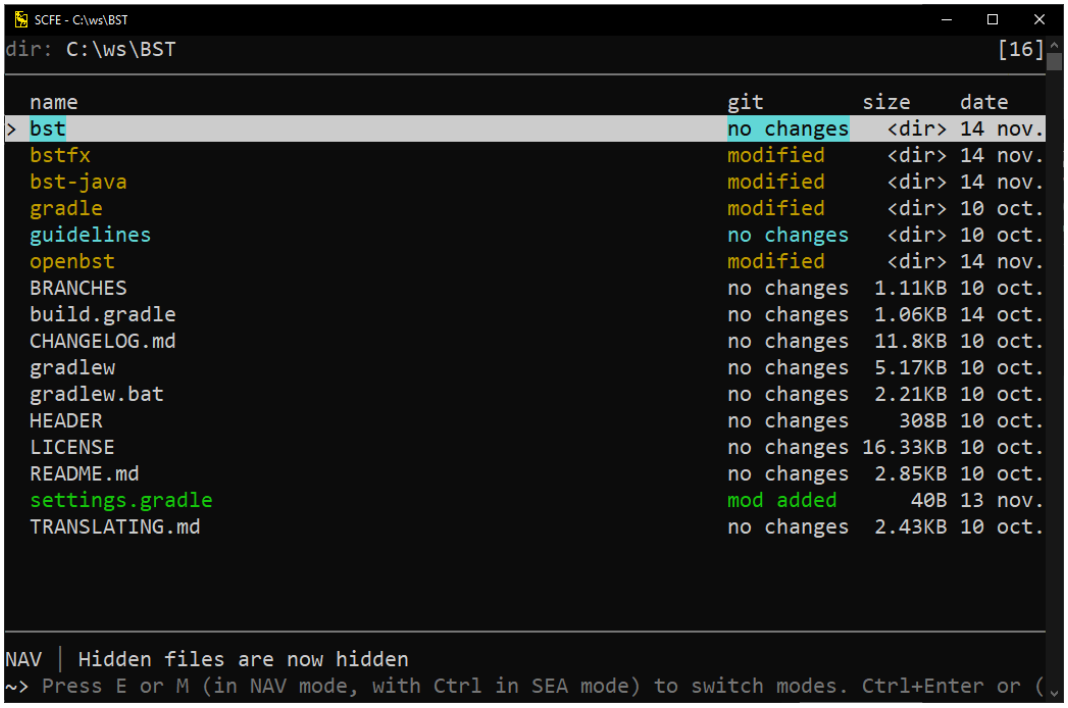

Figure 7: L'application finale dans un dépôt Git

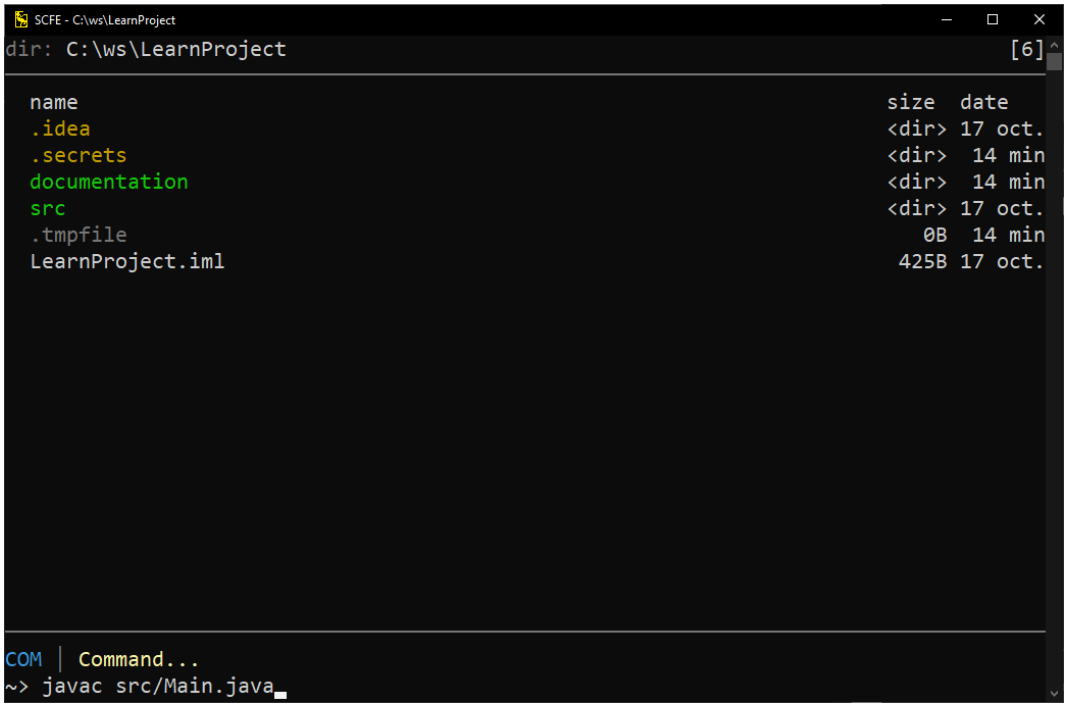

Figure 8: L'application finale en mode COM, dans un dossier avec des fichiers cachés