SCFE: Report 1 (First Presentation)

Matthieu Stombellini
Mathieu Rivier
François Soulier
Rakhmatullo Rashidov

Contents

1 Introduction

In this report, you will find details on the progress that has been made between the validation of the book of specifications for our project and the first defense. It provides details on the solutions that have been implemented and what should be done for the second defense.

One of the main goals of this first period was to get all of the members of the team comfortable with relatively new tools, as well as to create a solid basis for the project we will use extensively to build the application itself.

Before reading this document, and especially the task progression part, do note there are two Mat(t)hieus involved in this project, Matthieu Stombellini (with two T's, who is also the project leader) and Mathieu Rivier (with only one T).

1.1 Project subject reminder

Salamanders' Console File Explorer, or SCFE for short, aims to be a file explorer in the console that has the following goals:

- Usefulness, providing helpful information and adding something to the table when compared to regular file explorers,
- Beginner-friendliness, so that everyone has a chance of using the tool with all of its features with relatively low level of knowledge, especially when compared to regular CLI workflows,
- Power-user friendliness, so that more advanced users can get work done with SCFE,
- Approachable, by which we mean not in-your-face impressive but not letting you in without any clue of what you are supposed to do,
- Stability, as bugs are just an annoyance that no one appreciates, we will attempt to reduce them as much as possible.

It is built in C# using the `.NET Core` SDK, and, while its primarily targeted platform is Windows, it aims to be cross-platform. This decision was made both because it would increase the usefulness of the tool and because it is a necessity, as half of the members of our team use Mac computers.

The team behind this project is known as Salamanders' Lab, and is made of four motivated students: Matthieu Stombellini (project leader), Mathieu Rivier, François Soulier and Rakhmatullo Rashidov.

2 Book of Specifications follow-up

All the elements of the Book of Specifications were respected, and two of the technical uncertainties (such as which libraries we were going to use) have been solved, unless we see a particular need for more efficient replacements – which would be surprising, but we cannot realistically rule that out.

- For the Core UI, we created a library from the ground up (which we named `Viu` and will be detailed throughout this report). This allows us to have complete control over what is displayed and we can customize our interfaces exactly the way we want. The library itself is an integral part of our project but does not depend on other components, meaning that it can be entirely separated from our project and it will still work. This re-usability aspect makes Viu a perfect candidate for a future open-source release.
- For file manipulations, the basic classes provided by .NET have been more than enough up to this point in the project's development, although we did end up creating a wrapper around these classes, as they were not object-oriented representations. We might still use third-party libraries as an option, but we do not think they will be necessary.

An additional library – which you might see in screenshots or in the prototype – is being used, `JetBrains.Annotations`. It does not have any effect on what is actually ran and its only role is to provide better code error predictions (e.g. checking that no null values are given to parameters which require a non-null value) when automated tests are done on our code.

We were able to respect all of the requirements we placed upon ourselves throughout the first development of the project. We do not expect to have to break the book of specifications in any way, shape or form for the foreseeable future.

If we do need to break some of the features given in the Book of Specifications, these would be minimal (e.g. key bindings which end up being too cumbersome requiring simplification, mockup details...), or would otherwise go unnoticed for the end user, as they would be internal decisions related to the organization of the code base.

3 Task progression

The following part will present the progression of each task, also detailing who was in charge of which task.

3.1 Core UI

This task was the most important one for this first period, expecting 70% of overall completion (from 0%). It is the basis of the *entire* application.

3.1.1 Done

In order to actually build a console application, we needed to have something that could provide an easy way to display components, as well as handling inputs exactly as we wanted. Unfortunately, almost all of the C# libraries we could find lacked something crucial: it was often flexibility or cross-platform compatibility issues which made libraries unusable in our project. For the examples we thought about in our book of specifications:

- The `gui.cs`, while perfectly functional, was also extremely unclear, and not flexible enough. We also had doubts about its cross-platform capabilities.
- The `CursesSharp` library was just too low-level to be efficiently used, and also added numerous concerns over the portability and cross-platform capabilities of our code, the library requiring very specific bindings to system libraries which differed depending on the platform (UNIX or Windows).

This is why we decided to make our own library, based solely on the `Console` API available in `.NET`.

This sub-project of SCFE is one of its most important components: in order to match the flexibility and performance requirements we had in the Book of Specifications, a lot of work had to be done to make it as smooth as possible. The library was given a name, `Viu`, in order to separate it from code related to the application itself. The idea was that `SCFE` depended on `Viu`, not the other way around. While this separation might be seen as a step away from the original goal of the project, it is very much the opposite: the way in which we built `Viu` makes implementation of more high-level features of SCFE way easier.

The `Viu` library is heavily inspired by the Swing toolkit in Java: the validate-then-print workflow, component hierarchy and layout strategies resemble Swing, but no code has been taken from it, and, when actually creating interfaces with `Viu`, only some of the functions share characteristics with the Java system.

The main ideas developed throughout this first period were the creation of visual components (labels, text fields, tables, buttons...) as well as the implementation of various layout strategies. All of this was made harder by the fact that the interface always has a varying size, and layout strategies had to be flexible enough such that we would never have to touch low-level layout code and manipulating positional coordinates directly when building SCFE on top of `Viu`. Thanks to this and some light multithreading to catch when the window is resized, components dynamically resize themselves as the space around them changes (see figures 1 and 2 in the appendix).

`Viu` includes multiple "layout strategies" (see figure 3), allowing us to have components shown exactly as we want them to be. All strategies are able to smartly place components in them, always ensuring that they perfectly fit in, dynamically wrapping their components if necessary. The strategies illustrated in figure 3 are:

- The Border Strategy, laying out 4 components at each border and one in the middle
- The Line Strategy, organizing components into a horizontal or vertical line
- The Flow Strategy, putting components one after the other, wrapping them like a text if necessary, although components are of course not restricted to text only and can be any `Viu` component.

The most useful components were created, including simple texts (figures 1 and 2), text fields (which listen to user input), buttons and tables among others. As with layout strategies, they can all dynamically resize themselves. Tables can even resize each column individually with different widths for their content (provided that it has different sizes at its disposal) in order to either grow or shrink: an example of this in action can be seen in figures 4 and 5.

For extra flexibility, and if we wish to step away from the basic Console API for something that allows us to have a more fine-grained control over the output, `Viu` comes with its own abstraction layer above any console related code. Components (almost) never call the `Console` API directly, only calling our abstraction layer, for which the concrete implementation is given by a simple pass-through to the basic `Console` API.

`Viu` components were mainly built by Matthieu, who had experience with the Swing system (hence the resemblance), for the code that lays component out, the general appearance of components and the overall hierarchy of the `Viu` system. The user input side of `Viu` is described in its own task.

3.1.2 To Do

There is not much left to do for the low-level side for the interface. Adding a few extra features here and there for individual components will be done as needed when developing the remainder of the project, but this will not represent much.

Additional components also need to be coded, but will only be created when necessary so as not to focus on useless elements we may never use in the project, and the modularity of `Viu` makes this step fairly easy anyways.

For coding the actual application's interface: thanks to the great flexibility of the UI system, the work that is left to do is to actually use the components for SCFE and link them to all of the other tasks.

Between the first and second defenses, this task will go from 70% to 90%. This task is *on schedule*.

3.2 Navigation modes

This task was marginal for this first period, going from 0% to 20%, since it heavily relies on low-level work done in Core UI and User Input tasks.

3.2.1 Done

`Viu` only includes very few shortcuts (mostly arrow keys and enter key to navigate around the UI). A few additional shortcuts were added, making use of the Input Map and Action Map systems described in the User input section of this document. For the Action Map, most of the work done was hooking up a few shortcuts to action names in the NAV mode, as well as preparing the table used in the prototype to receive constantly changing input bindings. This was *not* fully implemented as it was unnecessary at this stage, but has proved itself to be a fairly thorough test for the input system to see how flexible it would be.

The existing shortcuts are mapped to their corresponding action name, in a dictionary, ready to be implemented into the main application once it gets out of the simple prototype state. This task was realized by François.

3.2.2 To Do

This task will receive the most work between the first and second defense. The goal here will be to fully implement hooks between shortcuts and actions (i.e. complete the ActionMaps seen in the User Input task), as well as fully implement the navigation mode switcher. This will be done once the prototype becomes more functional, but should not require too much of an effort.

Between the first and second defenses, this task will go from 20% to 70%. This task is *on schedule*.

3.3 File I/O

This task was of fairly little important for this first period, going from 0% to 30%

3.3.1 Done

While file input and output is an essential part of SCFE, it was far from being a priority for the first period. We planned to primarily focus on the basis of what has become `Viu`, making everything else later.

As such, François coded an object-oriented representation of the file system entirely based on the various classes provided by .NET which, unfortunately, were either only using static methods and strings, or did not have enough features to satisfy our needs. Having our own implementation also means that we will be able to copy paste files or even entire folders using a simple function in the file representation.

In addition, the methods implemented in this task such as `Copy` or `Move` use the System.IO classes, in such a way that the File class from SCFE can represent either a file or a folder. That specific property was particularly difficult to handle in the implementation, because each and every case had to be taken into account, and thus imply a need for recognizing the kind of file (folder or file) that was being dealt with.

To make it clearer, the implementation of the SCFE File class includes the call of `System.IO` methods in `System.IO.File`, but also `System.IO.Directory` since SCFE File can represent both files and directories (but a path cannot represent both a file and a directory at the same time of course). Moreover, some work on the path of a folder or a file has been realized in order to determine the parent folder of the file. However, the implementation of the class is not complete, as specified in the 'To Do' section below.

It is important to note that the functionalities of the File class are not used to their full extent yet: it is only used to display the content of files. Indeed, they will be used much more consequently by the second presentation.

François was responsible for the entirety of the task, with outside help from Matthieu for fixing a few bugs and properly using .NET APIs.

3.3.2 To Do

This task will also see a significant increase in progression between the first and second defense. Most of the work will be based on adding access to all of the attributes of a file (creation date, modification date, type...), in a flexible object-oriented way. Full copy and paste methods will also be added to the interface, and file navigation will be fully functional at this point.

Furthermore, the functions will be reviewed in order to be thoroughly optimized. In this respect, the safety checks present in the functions might be modified. Also, the File class must provide the possibility of showing the size of the file, and it must also provide a sorting option of the files in the display, for instance a sorting by alphabetical order, by the size of the files, or by the date of creation and modification of the files. The various tools to make these features a possibility will be created before the second presentation.

Between the first and second defenses, this task will go from 30% to 70%. This task is *on schedule*.

3.4 User input

This task was important for this first period, going from 0% to 40%.

3.4.1 Done

The main goal of this task for the first period was to provide interactivity for `Viu`: reacting to key inputs and managing focus states of all components.

A focus system was added, which allows the user to navigate between the various elements of the interface and throughout the component hierarchy using arrow keys (or other custom key bindings). This was done through smart use of C# interfaces on all components, which can declare the fact that they can be focused and handle key presses by simply implementing an interface. The focus system can be compared to the logic behind GUI applications which allow them to determine which component to select next when pressing `Tab` or `Shit+Tab`.

To create this focus system, we simply extended layout strategies to also include logic that tell which component should be the next one to be focused when going left, right, up or down.

Another aspect of the UserInput task was the InputMap/ActionMap system. Once again inspired by the Swing system, this allows us to separate shortcuts from actions. The idea is that `InputMap`s provide a binding between shortcuts and action names, while `ActionMap`s provide a binding between the action

names and the actual action. This system provides a fantastic "buffer" layer between what the user presses and what the program does. In a nutshell, the InputMap links key presses to action names, and the ActionMap links the same action names to the actual implemented actions (see figure 6). This is the exact system that will be used for the Navigation Modes task to switch back and forth between different shortcuts without fundamentally changing the actual action themselves.

Finally, a few components which are entirely based on user interaction were added, namely text fields and buttons. These were entirely custom built by handling individual key presses, since we were not really able to rely on the "regular" way of reading text from the console. Text fields provide a sizeable implementation of shortcuts which are common in text editors in order to simply make the text field component easier to use. Examples of such key presses include the Delete key or Ctrl+Arrow shortcuts (to jump between words instead of just between letters/symbols).

Heavy testing for this part was crucial to make sure that the input system was robust.

Mathieu was responsible for most of the task, implementing the input system (under Matthieu's supervision in order to make it fit nicely into the existing component hierarchy) with ActionMap and InputMaps as well as the focus system (once again with help from Matthieu), and Rakhmatullo was in charge of the "input reaction" part of a few components, including buttons (e.g. performing a predetermined action when pressing a button).

3.4.2 To Do

Most of the work left to do for user input will end up being very important after more work is done on every other task, in order to handle more complex key shortcuts or components.

As such, this task will only receive minimum attention for the second period, switching to a "maintenance mode" where the main focus will be fixing bugs, adding user input related code to new components and adding a few small extra shortcuts to the text fields.

Between the first and second defenses, this task will go from 40% to 50%. This task is *on schedule*.

3.5 Tool & OS integration

3.5.1 Done

Nothing in particular was done for this task, as was planned (0%) in the book of specifications, since this is a more advanced part of SCFE which has absolutely no use if the basic application is not stable and functional.

3.5.2 To Do

Basic OS integration will be provided for the next period, with examples such as identification of "symlinks" (on OS which feature them) or helpfully shortening displayed directory paths (e.g. turning `/home/myname/mydir/myfile.txt` to just `~/mydir/myfile.txt`). Moreover, opening files is also a feature of the underlying OS, which will be implemented and usable for the second presentation. Integration of heavier tools like Git will come between the second defense and the final presentation, or might even start before if we have enough time.

Between the first and second defenses, this task will go from 0% to 30%. This task has *not been started yet.*

3.6 Website

3.6.1 Done

Not much for this task was done in particular, as was planned (0%) in the book of specifications, since creating a website on a project that has just started would have redirected our efforts to a time-wasting task.

Mathieu did gather some information and mockups on what could be done in order to gain some time for the second presentation, but no website has been published yet.

3.6.2 To Do

Mathieu will take care of creating the website for the next period and ensuring content to it and that it is hosted properly. He and Rakhmatullo will be responsible for pushing content to it, both when it comes to the regular content expected from the website and the documentation.

We will explore various options for hosting the website during the next period, as we have not chosen a particular one yet.

Between the first and second defenses, this task will go from 0% to 50%. This task has *not been started yet*.

3.7 Documentation

3.7.1 Done

Some code documentation for `Viu` was done as we intend to publish it later on, but most of the work that has been done for this part is simply collecting intentions from the book of specifications to make them into a user-readable format. This is done in Markdown at the moment, but will be published onto the website later on.

The current documentation task has been progressing quite slowly, and we hope to make it much more useful as soon as possible. Some research about Markdown in general was done by Rakhmatullo to get some training on this format.

Moreover, we also have to study ways of turning Markdown source files into full HTML pages ready to be published on the website. We are not sure of how we may do this, but the most sensible option right now seems to be creating a custom script to automatically build the HTML documentation from the Markdown sources.

This task was entirely worked on by Rakhmatullo.

3.7.2 To Do

Just like the website, the documentation will be pushed and published during the second period, as well as adapted to the state of SCFE, in order to not document features that have not been implemented yet.

We might also consider publishing `Viu` API documentation to the website, but we are not entirely sure about whether we want to do that or not, since that would also mean releasing the sources of `Viu`, and extremely thorough testing needs to be done before any kind of public release.

4 Intentions for the second defense

Now that we have detailed each task's progression, we need to discuss the overall progression of the project.

4.1 Current state

Right now, the project has a very solid basis, and the implementation of the application can only be made easier down the line.

We have a fully working prototype (see figure 7) which handles key inputs well, and shows accurate representations of folders (folder navigation being limited, it only displays the current directory with no way of actually navigating back and forth). The creation of the prototype was made satisfying by the framework we created, with no major drawbacks in its implementation.

The current prototype also has a lot of gaps to be filled, in such a way that we know exactly *where* to add new features. This means that development of more advanced features should be relatively painless, or, at least, much easier than the full creation of `Viu`.

4.2 Goal

One of our priorities for the second defense is to turn all of this ground work into a usable product. We are on the right track, as the prototype was very easy to create yet worked very well without the need to fiddle with low-level code. As such, our efforts will especially be put towards making all of the different tasks work together and unifying them under the application.

If we stay realistic, the application for the second presentation will be pretty barebones, but fully functional. Polishing it and adding the hardest features to code will be our primary goal *between the second and final presentations*. For the mean time, we are focused on *making it work*.

The goals for each task have been described in the "To Do" section for each task – please refer back to them for more details. They are all along the lines of "improve and use in the main application", refining each piece and adding them to the general puzzle that SCFE is.

To recap, the main points we want to have covered for the second presentation are:

- Having a functional interface with working file navigation and actions,
- Completing the file implementation to get a good overview of the project,

- Fixing all bugs found in `Viu` in order to make it as robust as possible,
- Having a website and documentation which showcase the project and allow users to download the resulting pre-release application.

We believe that, while challenging, fulfilling all of these goals will be feasible for the following presentation.

We will have time restrictions for the second presentation, as it will come very soon, the delay being even worse between the second and final presentations. We will try to be as efficient as possible in order to not drown under work before the final presentation. This is nothing out of the ordinary though and is not that much of a threat for the project's progression or integrity.

5 Personal appreciation

While the task description provided was rather neutral and did not express how we felt about the project, as it was more about *what was done* rather than *what we think about what was done*, the following section will express the personal opinions of the members of Salamanders' Lab.

5.1 Matthieu Stombellini

I am fairly happy with what was produced in this first period. I do wish that task progression was a bit faster, and, unfortunately, I underestimated the huge task that creating a console interface system would be, causing problems for distributing other tasks to my teammates at first. We still managed to work on the project successfully, and the result is there: a full toolkit that, while remaining lightweight and usable, is quite complete and more than enough for the tasks that still need to be done. I have had various very frustrating problems with C# build systems during the first period, and ensuring that the project could be built successfully every time was very painful due to various bugs or surprising behaviors on Mono (which half of our team uses since they are Mac users).

Problems for testing the code and very strange behavior of terminals on Mac systems do make the whole development schedule slower than I wished it was, but I am still determined to make this project as cross-platform compatible, both for personal satisfaction and to actually be able to use this project in machine rooms!

5.2 Mathieu Rivier

The beginning of the project has been quite productive for the team. We have been able to build a solid basis for the app which will certainly facilitate the implementation of later features and insure a great cross platform compatibility from the ground up. For me, this project has been a great way to apply and understand better the many principles that we study during TPs and see in the TD classes, while learning many new things about development in general. Furthermore the collision and great team spirit make of this project a great learning experience and a unique opportunity for us to make something useful at our humble level. I think that this project will bring to each of us some extremely important lessons, that are: Team work and communication as our own strengths might not always be adapted for a particular task when someone's else could be; responsibility as we all need to do our part in due time in order for the project to advance at its normal pace and more. I really

enjoy working on the user input part of the project as it is what allows the user to interact with the application. I am looking forward to start working on the website that will provide our project a whole new level as it will finally be online for the public to see and start interacting with (even though that might mainly consist of the members of the group...). Overall I am very happy with how the project is turning out to be and think that our project will truly be of great use in our daily lives.

5.3 François Soulier

First of all, I am fairly impressed by the strength of the motivation we share regarding the project, because there is a real interest on the creation and realization of this project. Then, I would like to highlight the fact that the group cohesion is also very strong, which very important in a project like this, because reliability on each member of the group represents a big part of efficiency. Regarding the project and the progress that we did, I see myself very confident about it, especially concerning the schedule. Indeed, everything is going as planed and see absolutely no reason why it shouldn't be the case later on. To go into further details, I really enjoy working on the file manipulation, as it presents a real liberty about the methods that can be used. In addition, I am looking forward to improving that part, in order to make it more flexible and optimized. Nevertheless, I appreciate getting myself into the navigation modes, meaning working closer to the visual and graphic interface – which is actually the area where cohesion is the major factor since every member is involved. On the whole, and regardless of our work efficiency, I am very enthusiastic about the entirety of our project, because I find it a real coherence and interest, both from a programming and an algorithmic point of view.

5.4 Rakhmatullo Rashidov

Throughout the first part of the project, I have been doing the documentation using Markdown. To be honest, before starting, I did not know how to use Markdown – However, right now, I am comfortable with it since I have finished some part of the documentation using it. Although I did not have many tasks to perform for this period, I checked the other parts of the code we are implementing as I was very curious. All in all, I have been learning a lot. I have always loved learning new things, especially, coding. I am happy that we have made a lot of progress and I think our group is the best!

6 Conclusion

To conclude, we are all proud of the result of this first work period, and, while the hardest part has been done for some tasks, we are well aware that this is only the beginning. We will work as much as we can during this second period to ensure the delivery of a light yet functional version of our application.

One of the most exciting aspects of this project is realizing that what we have made will be useful, even outside of SCFE – the `Viu` system can be taken as a prime example for something we will be able to re-use in personal projects in the future. Thanks to our constant motivation of *creating something useful*, we are able to give a new perspective to a project that would otherwise have been forgotten past the second semester.

We are excited for the next evolutions of our project, as the second presentation will be a perfect way of showing what we have learnt and showcasing our take on how to create a modern, functional and simple console application.

7 Appendix

These images are related to the various tasks seen above and either showcase features of `Viu`, or how `Viu` works internally, as well as showing the current prototype for the SCFE application.

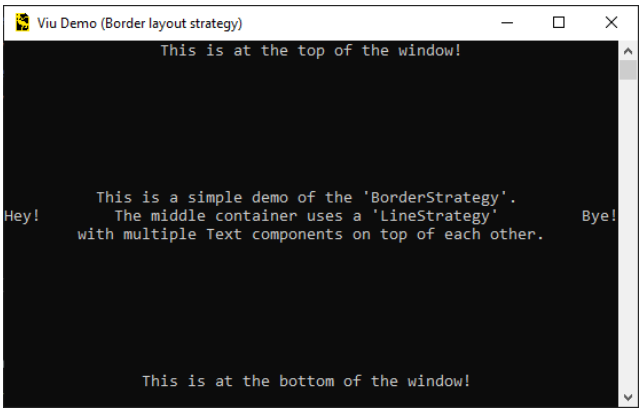

Figure 1: A simple UI in a small form factor

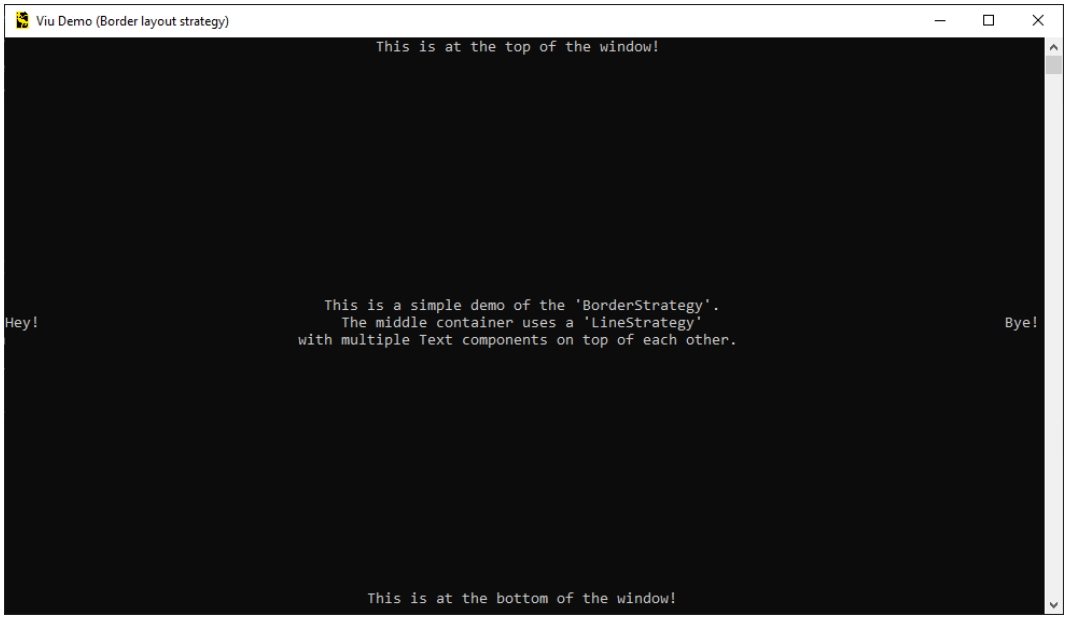

Figure 2: The same UI after the window is resized

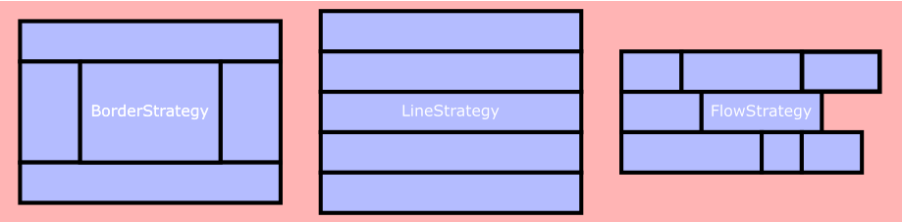

Figure 3: Layout strategies for components (Border, Line and Flow strategies)

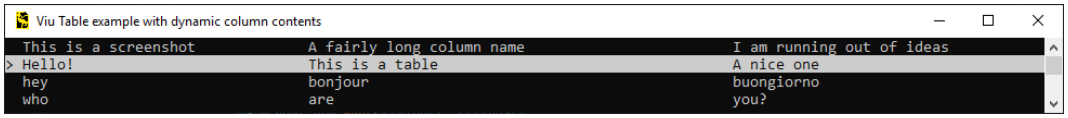

Figure 4: The Table component if the window is larger than needed (columns grow)

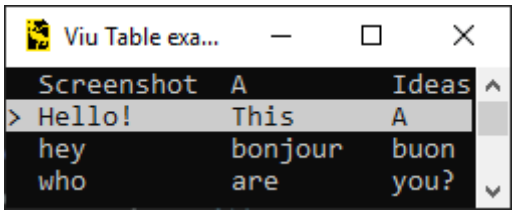

Figure 5: The Table component if the window is shorter than required (columns shrink)

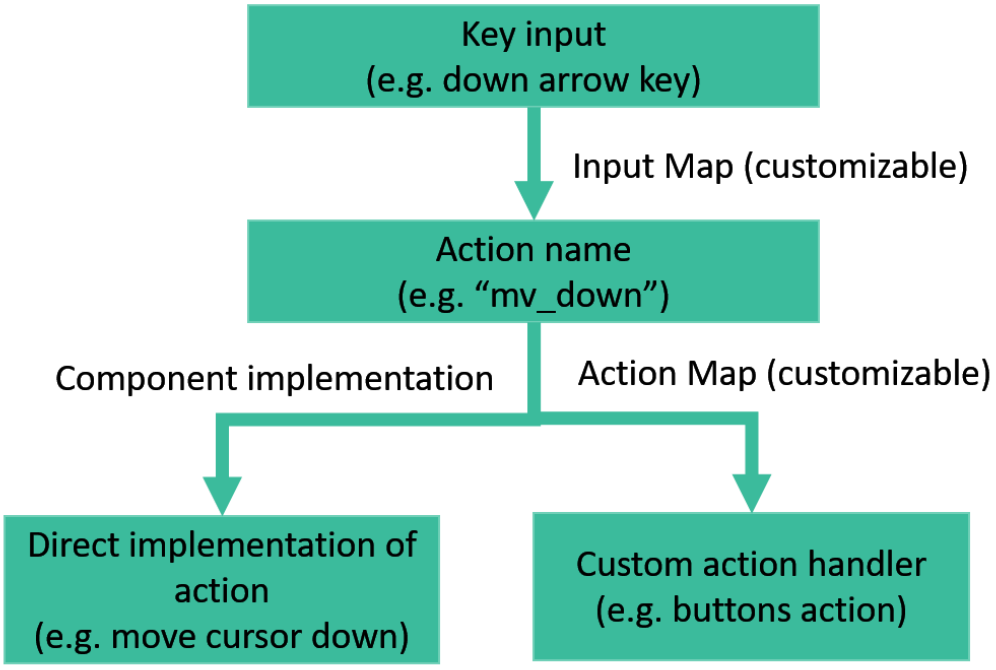

Figure 6: Illustration of how `InputMap` and `ActionMap` work

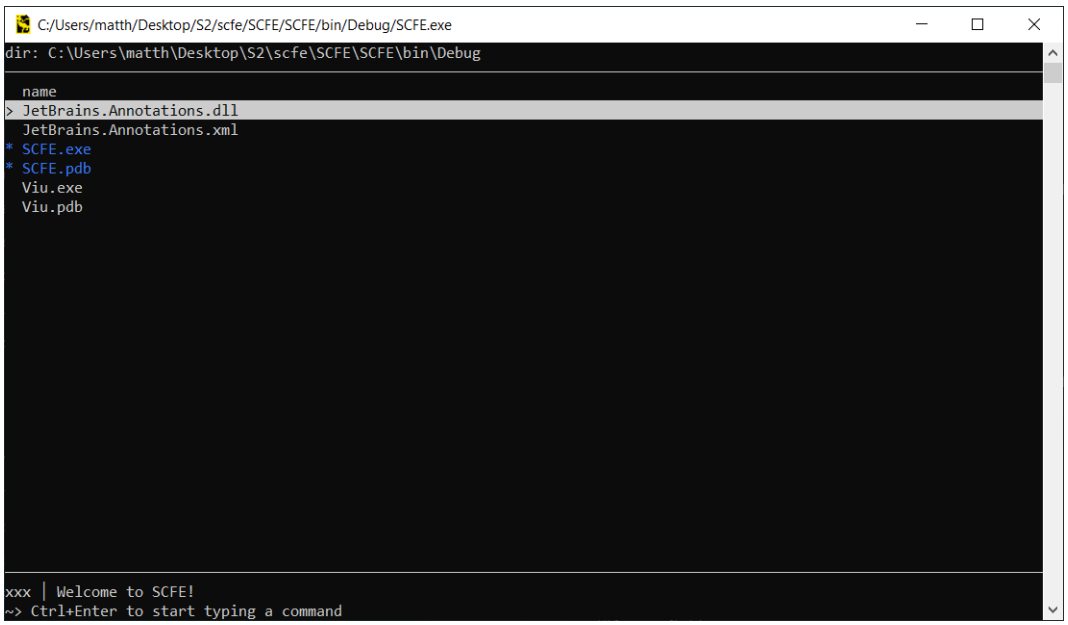

Figure 7: Current prototype of the application