Rapport de stage

Développement d’une évolution d’un générateur de figures de Navon

IUT Informatique de Dijon,

[Titre du cours]

# Remerciements

Je tiens à remercier Madame Poulin Charronnat Bénédicte, la Directrice du LEAD[[1]](#footnote-1) pour m’avoir accepté au sein du laboratoire.

Je remercie les doctorants avec lesquels j’ai passé mon stage, pour leur accueil, leur bonne humeur et pour m’avoir permis de m’intégrer facilement.

Je remercie Monsieur Bard, mon maître de stage, pour sa confiance, pour m’avoir partagé ses connaissances et l’autonomie qu’il m’a offert pendant ce stage, me permettant d’affirmer mes compétences.

Je souhaite adresser mes remerciements au corps professoral et administratif de l’IUT Informatique de Dijon, pour la qualité de l’enseignement et plus particulièrement Madame Sologny pour m’avoir suivi durant ce stage.

Table des matières trop drôle

# Présentation du contexte

## Présentation générale

## Présentation du service

## Définition de la mission

### La problématique

Avant de vous présenter la problématique je souhaiterais vous expliquer ce qu’est une « Figure de Navon » : Une figure de Navon c’est un élément de grande taille (une lettre par exemple) composé d’éléments de plus petite taille comme :

INSERER UNE IMAGE

Une a

Au commencement, ce projet n’était qu’un sujet d’une discussion quelconque entre Monsieur Bard et Monsieur Witt. Lorsque celui-ci faisait ses figures de Navon, il perdait beaucoup de temps à former lettre par lettre, une figure de Navon. Une expérience ne se faisant pas qu’avec une seule figure de Navon, la préparation lui prenait une éternité. C’est à partir de ce moment-là que se développa l’idée d’un logiciel permettant de faciliter la tâche. N’ayant trouvé aucun logiciel, ils décidèrent de donner ce sujet en tant que projet à ESIREM. Deux premières versions furent développées. Les deux étaient très différentes l’une de l’autre et aucune n’atteignait les attentes voulues.

### La tâche à effectuer

Le projet était initialement nommé : « Développement d’une évolution d’un générateur de figures de Navon » : Je devais reprendre le code des précédents étudiants sur ce projet et l’améliorer afin qu’il atteigne les attentes voulues. Mon projet s’est rapidement renommé « Développement ~~d’une évolution~~ d’un générateur de figures de Navon » : Les codes ne pouvaient être suivis par d’autres personnes que les programmeurs eux-mêmes : il n’y avait aucun commentaire, le code possédait des erreurs non gérées. Ce qui avait été fait était intéressant, mais je ne pouvais pas reprendre ces travaux. J’avais donc décidé de repartir sur une base nouvelle. L’idée de ce projet est que l’utilisateur ait la possibilité de créer facilement des figures de Navon simples, telles que des lettres de l’alphabet et la possibilité de dessiner une figure de Navon. Et pour les deux façons, il doit avoir la possibilité de changer plusieurs options, principalement :

* Changer le caractère local
* Changer le caractère local en une « image locale » : Que la figure dessinée soit remplie de petites images.
* Changer la densité : le nombre l’éléments locaux (caractères ou images) utilisés afin de faire la figure

## Méthode retenue

### Les différentes solutions envisagées

Les solutions envisagées étaient celles recherchées et testées par les précédents groupes :

* La création d’une figure de Navon par géométrie analytique : On créer une figure de Navon à partir des coordonnées des éléments utilisés.
  + Par exemple : Un « A » majuscule est composé de trois segments.

INSERER UNE IMAGE

* Une grille avec des cases cliquables correspondantes aux emplacements où des lettres doivent apparaître.

INSERER UNE IMAGE

Il fallait dans un premier temps, tester ces possibilités, trouver comment celles-ci étaient améliorables, s’il y avait moyen d’implémenter le point le plus important : la densité.

La densité était, pour ces essais, le point problématique qui n’avait pas été géré.

### Les matériels et logiciels utilisés

Afin de réaliser ce projet, voici ce que j’ai utilisé :

#### Python 3.9

Pour ses bibliothèques diverses et son implémentation simple, j’ai décidé de faire mon projet en Python. La version utilisée est la 3.9, c’est à l’heure actuelle la version la plus stable qu’il existe. Le Python possède plusieurs bibliothèques qui m’ont été très utile dans la réalisation de ce projet, des bibliothèques puissantes et assez simple d’utilisation :

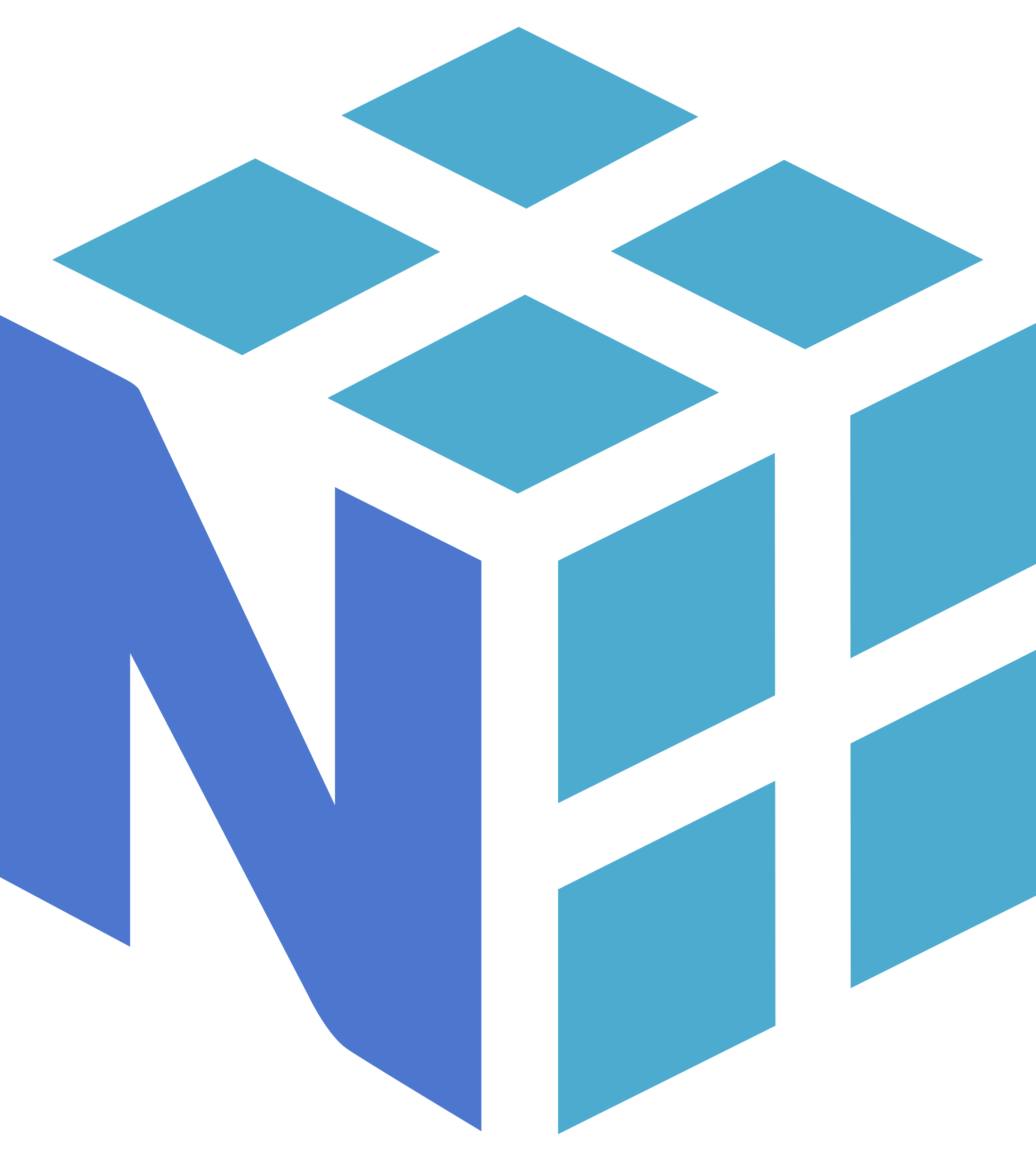


Image : Numpy

Numpy : une bibliothèque permettant la création de tableaux et matrice à N dimensions, propose diverses fonctions mathématiques applicables directement aux tableaux créés et ce, sans réduire les performances du programme.



Image : SciPy

SciPy : une bibliothèque à usage scientifique, utilisant d’autres bibliothèques telles que Numpy citée précédemment. Cette bibliothèque a été utilisé pour sa fonction d’interpolation.



Image : Pillow

Pillow : une bibliothèque de gestion d’image « Python Image Library » me permettant de créer les images sur lesquelles sont dessinés les figures de Navon.



Image : Tkinter

Tkinter : une bibliothèque de graphique, m’ayant permis de créer l’interface homme-machine du programme.

#### Environnements de développements

Le premier EDI utilisé, principalement lors de la phase de test est Pycharm Community Edition :

Pycharm est un EDI utilisé pour programmer en Python, un EDI développé permettant d’analyser son code, d’utiliser un débugueur graphique et d’effectuer des tests entre autres. Il m’a permis, dans les premières semaines de développement, de test et de recherche, d’effectuer beaucoup de test, me permettant de découvrir les différentes bibliothèques citées plus haut.



Image : Pycharm



Image : Sublime Text

Le second outil, celui que j’ai utilisé la majorité du temps est Sublime Text afin de développer ce programme. Sublime Text est un éditeur de texte, simple d’utilisation qui permet une navigation et une visualisation simple des fichiers utilisés.

#### Conception

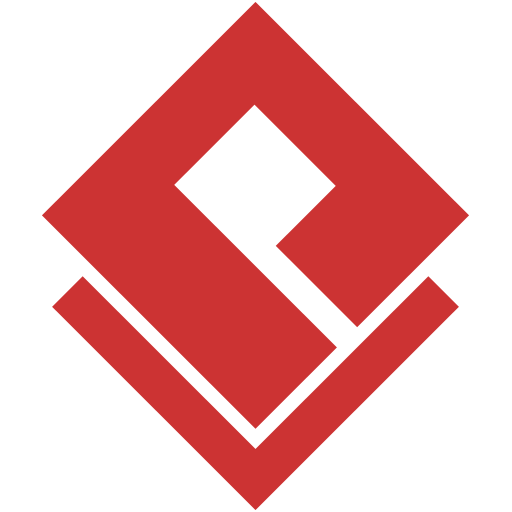


Image : Visual Paradigm

Visual Paradigm : Logiciel me permettant de faire les diagrammes UML[[2]](#footnote-2). Nous avions déjà utilisé ce logiciel en cours afin de créer la conception de projet.

GitMind : La licence de Visual Paradigm ayant expirée avant la fin de mon stage, j’ai repris toute la conception sur GitMind, un site permettant de faire différents diagrammes UML.



Image : GitMind

#### Suivi de l’avancée

Afin de suivre la progression globale du projet, j’ai utilisé plusieurs logiciels à ma disposition :

GitHub : Afin de faire des sauvegardes régulières de mon code et de faire un pas en arrière si besoin, mon avancée de la programmation du projet est enregistrée dans un dossier que seul moi peut accéder. Ces enregistrements permettent en cas de pépins (comme cela a pu m’arriver) de reprendre à un point précédent sans tout recommencer.



Image : GitHub



Image : OneNote

OneNote : Le programme de prise de note de Microsoft, me permet de faire un suivi quotidien de mon avancée dans mes recherches.



Image : GanttProject

GanttProject : Un logiciel me permettant de modéliser le diagramme de Gantt de ce projet.

## Application de la méthode et résultats

### Les différentes phases de la réalisation

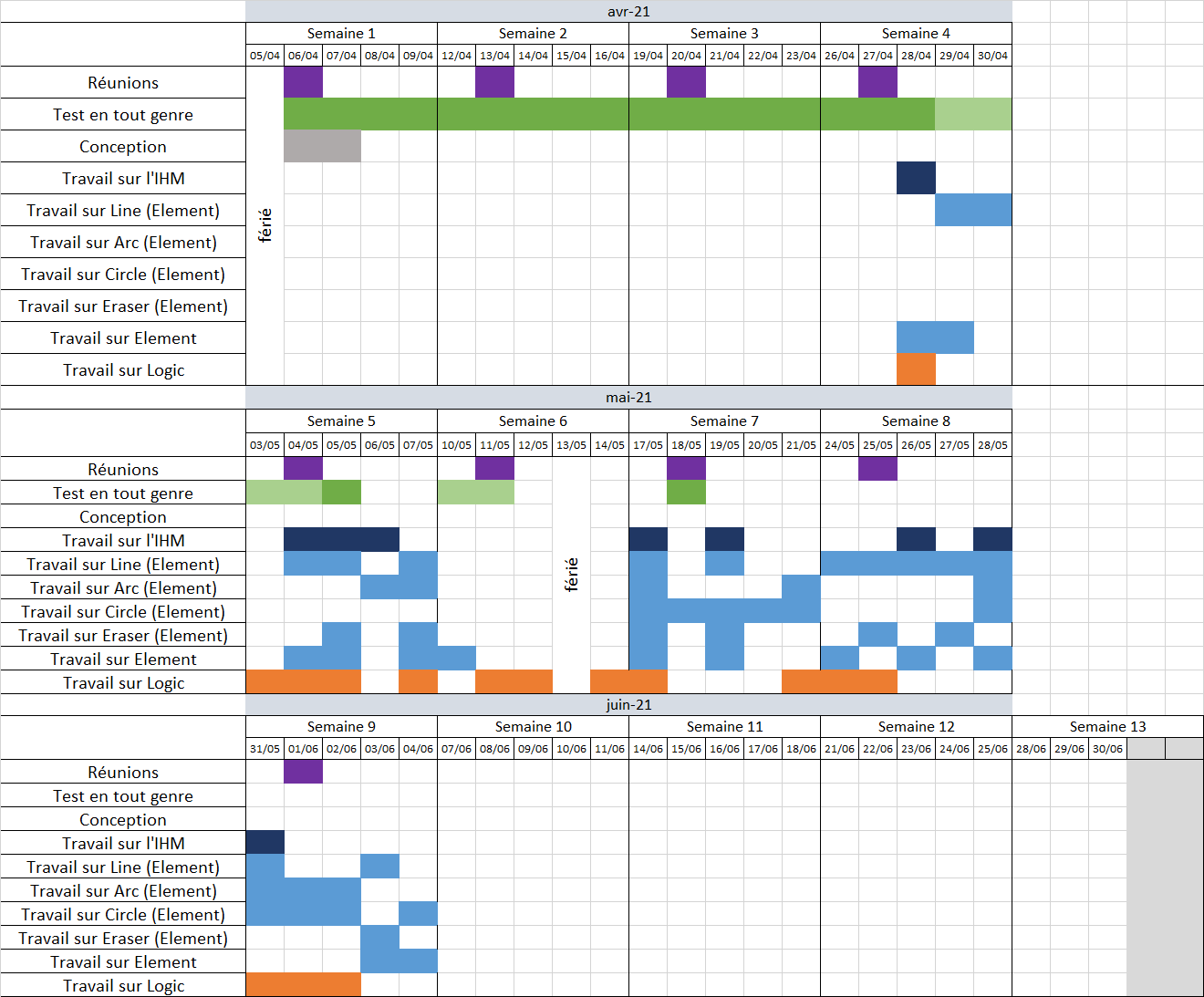


Image : Diagramme Gantt de mon stage

Comme nous pouvons le voir sur ce diagramme de Gantt (Image 12 : Diagramme Gantt de mon stageImage 12 : Diagramme Gantt de mon stage) : mon travail s’est décomposé en deux grandes phases :

1. Une première phase d’une durée d’environ un mois : une phase de documentation, d’essais
2. La seconde phase étant la programmation du projet s’étant de la fin du mois d’Avril à aujourd’hui.

#### Première phase : documentation et essais

La première phase est une phase de recherche. Une observation du travail fait précédemment. Une découverte des bibliothèques : NumPy, Matplotlib, SciPy sont des bibliothèques très utiles pour les calculs scientifiques mais que je n’ai jamais utilisé auparavant, mes connaissances en Python étaient assez pauvres au début du stage. J’ai donc commencé par des tutoriels, des vidéos, des démonstrations de ce qu’il était possible de faire, afin de prendre en mains ces nouveaux outils.

Dès nos premières réunions, nous avons parlé du principal problème de ce projet : la gestion d’une densité : le nombre d’élément locaux qui seront utilisés afin de donner forme à l’élément global voulu (Image 13 : Démonstration densité).

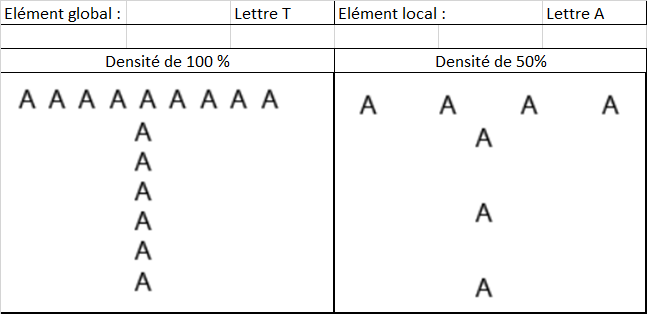


Image : Démonstration densité

La première idée dont nous avons discuté était d’avoir une grille sur laquelle on pouvait cliquer afin de décider des emplacements où faire apparaître des éléments locaux lors de « l’impression » de la figure de Navon (Image 14 : Exemple grille).

Cette idée avait déjà étudié et programmé auparavant, en C++, l’interface n’était pas simple d’utilisation : Il fallait dessiner sur la grille la figure voulue, enregistrer un fichier au format .txt puis téléverser ce nouveau document sur le programme afin qu’il créer une figure de Navon à partir de celui-ci.

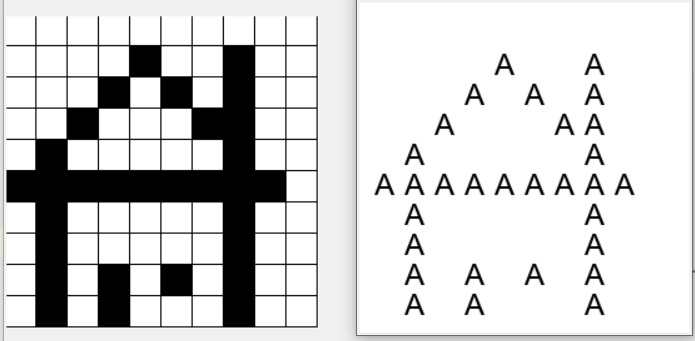


Image : Exemple grille

Nous avons recherché plusieurs solutions possibles concernant la gestion de la densité avec cette méthode de grille de dessin et des cases cliquables, plusieurs techniques ont été testé :

1. La première consistait à créer un « masque de densité » : ce masque serait une grille de la même taille que la grille de dessin, et elle serait composée de 1 et de 0 en fonction du pourcentage de densité donné. Et avant de créer la figure de Navon, on « superpose » les deux grilles et on ne garde de la grille de dessin que les cases noircies (Image 15 : Exemple densité (50%) par masque).

Comme on peut le voir sur cet exemple, le masque peut finir par faire perdre tout son sens à la figure de Navon voulue.



Image : Exemple densité (50%) par masque

1. La seconde technique consistait à créer des espaces entre chaque lignes et colonnes de la grille de dessin. On reprend la grille de dessin et entre chaque ligne, on ajoute entre chaque ligne et sous chaque ligne un nombre d’espace choisit (Image 16 : Densité par 2 espaces**Erreur ! Source du renvoi introuvable.**).

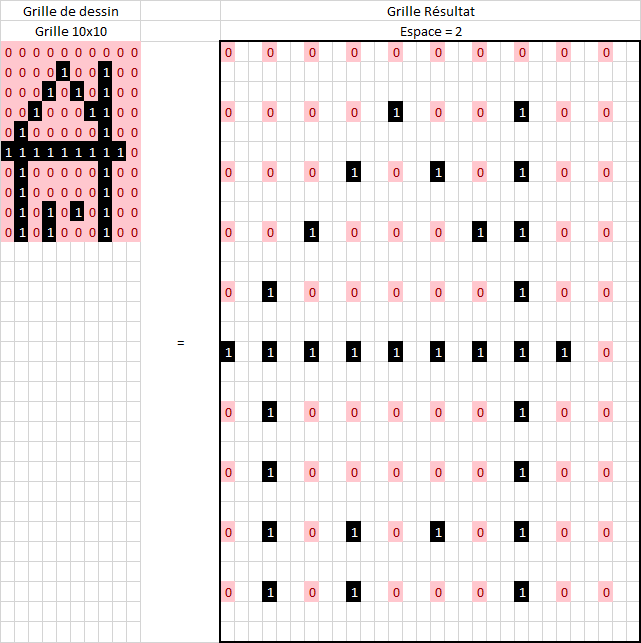


Image 16 : Densité par 2 espaces

Une figure de Navon créée avec 2 espaces comme ici augmente considérablement la taille de la figure de Navon et le temps d’exécution, de création de la figure de Navon augmente considérablement lui aussi.

1. La troisième technique consistait à ajouter, au lieu d’espace comme la version précédente, d’autres éléments locaux du même type : on multipliait l’état de la case un nombre de fois choisit par l’utilisateur (Image 17 : Densité par multiplication des cases) :

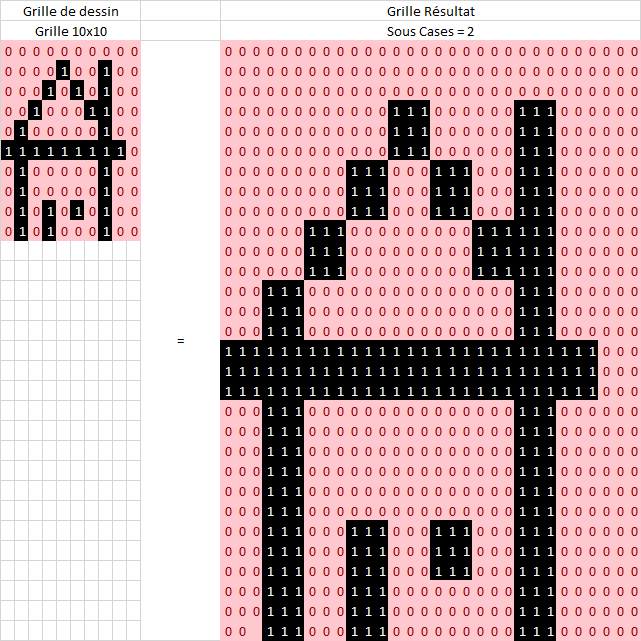


Image 17 : Densité par multiplication des cases

Comme avec les espaces, le temps d’exécution et la taille de la figure augmentent considérablement.

1. Il y eu d’autres expérimentations tels que le mélange de la technique « d’espacement » et de « multiplication des cases » mais sans réels résultats convenables.

Ces derniers tests n’étant pas concluant, il fallait trouver une meilleure solution… Les grilles sont simples à utiliser, mais le long temps de création de la figure de Navon n’est pas envisageable pour le programme final et les grilles présentent un autre problème : Nous ne pouvons pas créer de figures complexes composées de cercles ou de demi-cercles. Ces tests nous ont prouvé que passer par une grille de dessin n’est pas une solution adaptée pour ce que l’on cherche à faire.

Nous partons donc sur une variante de la seconde idée programmé auparavant, qui était faites de coordonnées et de géométriques analytique :

#### Seconde phase : programmation du projet

La programmation de ce projet ne s’est pas passé tel un simple projet fait durant les cours à l’IUT : Chaque avancée ne fut que de petites avancées et il m’a fallu souvent revenir sur mon précédent code afin de le développer, de le modifier voire de le supprimer.

Sachant que ce rapport vous a été donné avant la fin de mon stage, il se peu que beaucoup de chose change à nouveau. Mais l’idée générale sera expliquée :

Comme on peut le retrouver sur le diagramme de Gantt (Image 12Image 12) : J’ai séparé la programmation en trois grandes parties :

1. IHM : (en bleu foncé) comprend les différentes parties visibles du projet : la fenêtre principale est composée d’une partie permettant à l’utilisateur de créer une figure de Navon de A à Z et seconde partie permettant d’obtenir facilement une figure à partir de plusieurs paramètres.

Une troisième partie, ayant débuté juste avant le rendu des rapports, comprend un aperçu direct de la figure de Navon en cours de création.

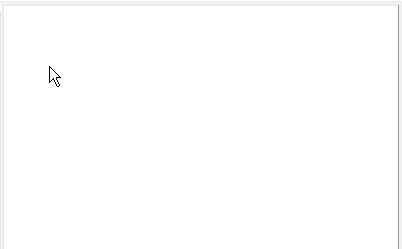
1. Les éléments : (en bleu clair) correspond à tous les éléments dessinables que comprend le programme : l’utilisateur peut dessiner un trait, un demi-cercle et un cercle et le programme « traduit » ces éléments en une suite de lettres ou d’images locales pour la figure de Navon. L’utilisateur a, à sa disposition, une gomme (Eraser) afin de supprimer un élément.
2. Logic : (en orange) comprend toutes les fonctions de créations de la figure de Navon. Logic comprend aussi une partie avec des paramètres nécessaires à la création de l’application.

A travers ces trois parties nous pouvons nous poser une multitude de questions : Comment dessiner ses différents éléments ? Comment est calculé la façon dont sont placés les lettres ou les images locales pour la figure de Navon finale ? Comment la densité a-t-elle finalement été gérée ?

##### Comment dessiner ses différents éléments ?

L’utilisateur, comme toute autre personne dépend de sa souris afin d’utiliser l’application, j’ai donc utilisé ses « dépendances » afin de créer ses éléments sur la surface de dessin, par exemple : la création d’un trait débute par un **Clic** sur la surface de dessin, puis l’utilisateur **Déplace** la souris jusqu’à obtenir l’élément sous sa forme voulue et fini par **Relâcher** le bouton de la souris afin de confirmer la création de cet élément.

Ses trois dépendances sont les mêmes pour chaque élément que l’utilisateur peut créer. A l’exception du Eraser servant lui, à l’inverse, de supprimer des éléments. Le **Clic** indique la création d’un trait, d’un cercle. Chaque **déplacement** de la souris avec le clic enfoncé change la forme de l’élément, puis le **relâchement** du clic de la souris permet son enregistrement : Ses informations, telles que ses coordonnées, son type etc… sont enregistrées afin d’être réutilisées ultérieurement (: Création d'une ligne).



Gif 1 : Création d'une ligne

### Les difficultés rencontrées

## Conclusion

### Les suggestions pour l’entreprise

### Les leçons tirées de ce travail

1. Laboratoire d’Etude de l’Apprentissage et du Développement [↑](#footnote-ref-1)
2. Unified Modeling Language, Langage de modélisation unifié : langage de modélisation graphique à base de pictogrammes [↑](#footnote-ref-2)