Cycle des Ingénieurs de l'ENSG 3ème année
Spécialité PPMD

Rapport d'analyse

Interface graphique d'aide à la création de réseaux de neurones

---

30 novembre 2023

Commanditaire : Arnaud LE-BRIS, IGN - LASTIG

Auteur : Malo DE LACOUR, ENSG

Sommaire

1. Contexte

Les méthodes par apprentissage profonds sont de plus en plus répandues pour réaliser diverses tâches (classification, régressions, ...). Différents outils tels que Pytorch, Tensorflow, Keras ... sont disponibles pour créer et entraîner ce type de modèle. L'une des étapes essentielles en apprentissage profond est la définition de l'architecture du réseau de neurones, c'est-à-dire définir l'enchaînement et la nature des différentes couches de neurones. Cette architecture est définie par du code. C'est l'un des facteurs sur lequel il est possible d'intervenir pour améliorer de manière significative les performances d'un modèle. De ce fait, cette architecture est amenée à être régulièrement modifiée avec parfois l'ajout d'étapes complexes. Cependant la diversité des enchaînements ainsi que des structures de réseaux rend parfois difficile la compréhension de l'architecture dans un environnement de développement classique.

De ce fait, une représentation schématique permettant de synthétiser et d'expliciter une architecture pour un humain s'avère pertinente. En effet, le concepteur pourra "dessiner" l'architecture du réseau de neurones et visualiser celle-ci de manière explicite pendant la phase de conception. Cet outil rendra la création de réseau de neurones intuitifs, ergonomiques et permettra de réaliser un gain de temps important.

2. Objectifs de l'étude

Le logiciel devra permettre à l'utilisateur de réaliser graphiquement son architecture et de le convertir sous forme de code python. L'essentiel du travail consistera à réaliser ou adapter une interface graphique visuellement approprié à la problématique des CNN.

2.1 Existant

2.1.1 Programmation par blocs :

Plusieurs domaines tels que le jeu vidéo (Blueprint dans Unreal Engine), la modélisation procédurale (Blender, Houdini, Speedtree) ainsi que les SIG (modelBuilder de ArcGIS) utilisent déjà la programmation par blocs. Ces interfaces de programmations visuelles ont surtout été développées pour rendre accessible le développement et la programmation (souvent complexe à appréhender) à des non-programmeurs. Cette solution ergonomique et intuitive permet entre autres à des designers et créateurs dans le milieu du jeu vidéo et des effets spéciaux de programmer des éléments. Cela représente aussi de nombreux avantages comme en SIG avec un gain de temps pour le développement et le déploiement de petits outils personnalisés.

2.1.2 Représentation visuelle des CNN

Il existe de nombreux outils tels que Visual Keras, neutron et conx qui permettent de visualiser la structure d'un réseau de neurones déjà implémenté. Ils permettent d'illustrer un réseau en montrant l'enchaînement des différentes couches ainsi que leur nature et dimension. Ces outils produisent un résultat très esthétique pour illustrer ou analyser mais ne permettent aucune interaction et ne sont donc pas adapter à la phase de conception. De

plus, certaines méthodes de visualisation ne sont pas adaptées à des architectures complexes.

Il existe cependant ENNUI et DeepLearning Studio qui sont des solutions très pertinentes pour répondre aux besoins de simplicité et d'ergonomie. Le problème de la première solution ENNUI réside dans le manque d'ergonomie pour manipuler des structures plus complexes. La seconde solution est propriétaire.

2.2 Contraintes

Aucune contrainte technique n'a été imposée par le commanditaire. Le choix de la librairie d'apprentissage privilégiée est Pytorch, cependant une librairie Tensorflow ou Keras convient très bien aussi. Il faut cependant intégrer un certain nombre de fonctionnalité décrite ci-dessous par ordre de priorité :

- **Représentation graphique** simple, interactive et ergonomique de l'architecture CNN
- Implémentation des couches et fonctions couramment utilisées en CNN
    - **Couches** : Pooling (max-min-avg) – Convolution – DenseLayer – Dropout – Batch-Normalisation – Flatten – Relu – Linear
    - **Opérations spécifiques** : Concaténation – Addition – Multiplication
      Permettre de convertir l'architecture représentée dans le logiciel en code python.
- **Permettre de sauvegarder** le travail réalisé sur l'interface dans un fichier spécifique pour reprendre ultérieurement.
- **Intégrer un système de contrôle** des flux (taille des données cohérentes d'un bloc à l'autre)
- Permettre de créer de nouveaux blocs personnalisés réutilisables à partir d'un ensemble de bloc de base

Une contrainte pourrait provenir de l'utilisation d'un code déjà existant. En effet, la création d'une interface graphique est très chronophage et parfois technique. C'est pourquoi il est très probable que je réutilise un code de programmation visuelle ce qui implique de s'approprier celui-ci. De plus, la structure du projet doit permettre d'ajouter de nouveaux blocs pour garantir l'évolutivité et la pérennité de celui-ci.

2.3 Objectifs

L'utilisateur doit pouvoir "dessiner" en toute simplicité son architecture. Il doit pouvoir réaliser des actions simples pour y parvenir. Ces actions sont représentées et synthétisées dans le diagramme de cas d'utilisation ***figure 1***.

- Ajouter et paramétrer un bloc
- Créer ou supprimer des connexions entre les blocs
- Convertir et enregistrer le schéma en code python
- Enregistrer le schéma pour conserver une version éditable

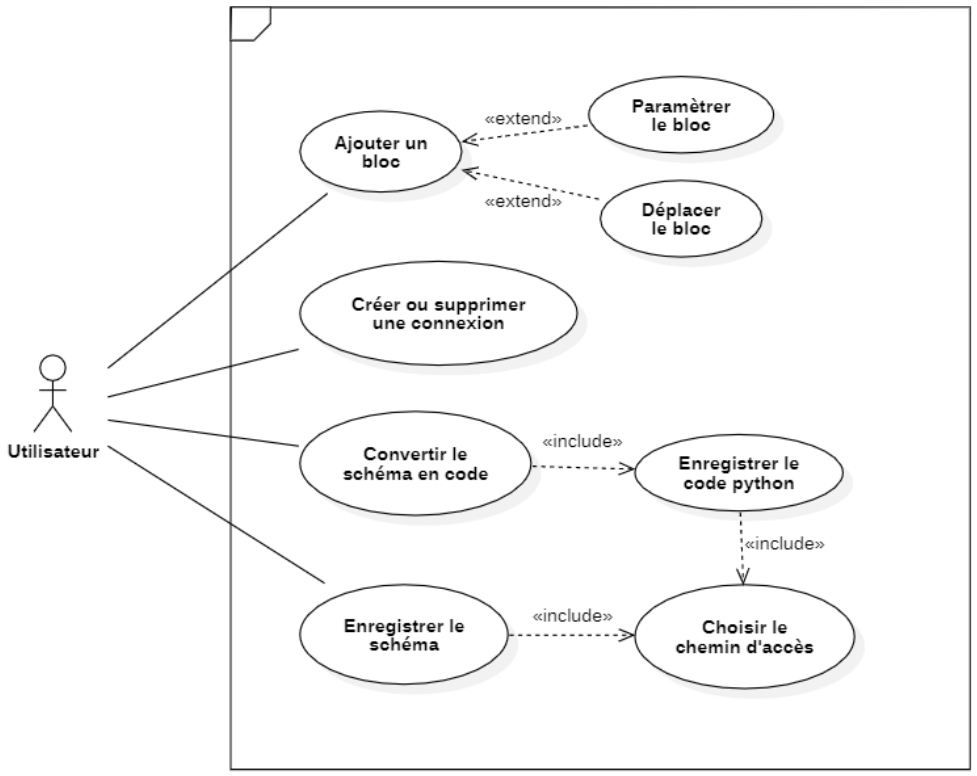

Figure 1 : Diagramme de cas d'utilisation

3. Analyse fonctionnelle

3.1 Architecture globale

Cette interface est un outil indépendant d'aide à la conception qui doit pouvoir fonctionner indépendamment de d'autres outils. Son architecture et son fonctionnement sont à réaliser de zéro. La solution de visualisation retenue est un système par bloc avec des connexions en entrée et sortie. L'utilisateur peut déplacer et connecter ces blocs avec un système de glisser/déposer. Une surbrillance des blocs et connexions permet à l'utilisateur de visualiser les éventuelles erreurs dans son architecture. Pour des questions d'ergonomie, la taille des données n'est pas directement visible dans le schéma de bloc.

Cette méthode de programmation nécessite de manipuler différents objets de base. Cela rend une structure orientée objet particulièrement pertinente pour la réalisation et l'évolutivité de ce projet. L'utilisateur est amené à manipuler et paramétrer ces objets :

- **un bloc (ou node)** : Cet objet correspond à une fonction avec ces paramètres ou à une variable. Il est constitué d'entrées et sorties matérialisé par des pins.
- **des paramètres** : Ce sont différents types d'objets qui peuvent composer un bloc tel que des champs numériques, boutons et curseurs. Ils permettent de saisir des paramètres comme des valeurs ou des méthodes que le bloc utilise.

- **une accroche (ou pin)** : C'est un morceau de bloc sur lequel vient s'accrocher une connexion. Chaque pin correspond à un paramètre et matérialise une entrée ou sortie de bloc
- **une connexion** : Elle permet de relier les blocs entre eux en s'accrochant aux pin. Elle est définie par un début et une fin. La connexion est représentée par un "cable" pour matérialiser le lien entre les blocs.
- **un schéma** : Le schéma est l'ensemble de blocs et connexion avec les paramètres qui y ont été renseigné. Il représente l'architecture du réseau en entier.

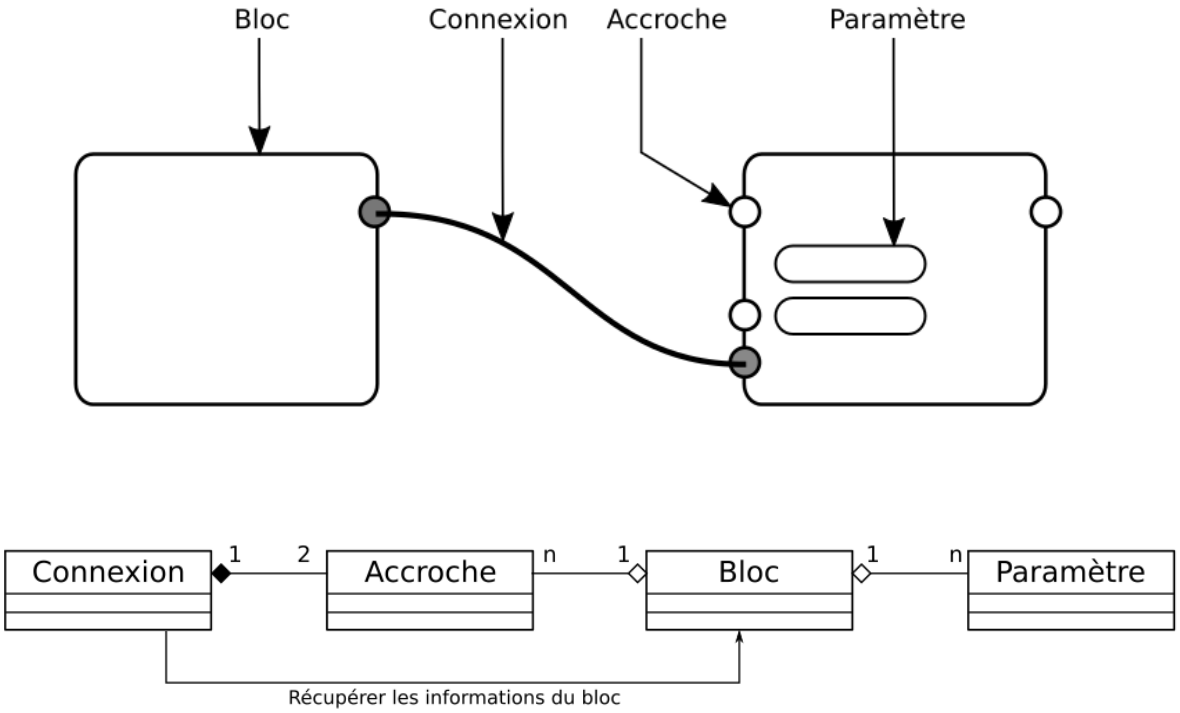

Figure 2 : Diagramme de classe simplifié des objets et leur représentations graphique

La **figure 2** représente les liens entre les différents objets décrits précédemment à l'aide d'un diagramme de classe.

3.2 Utilisation du logiciel

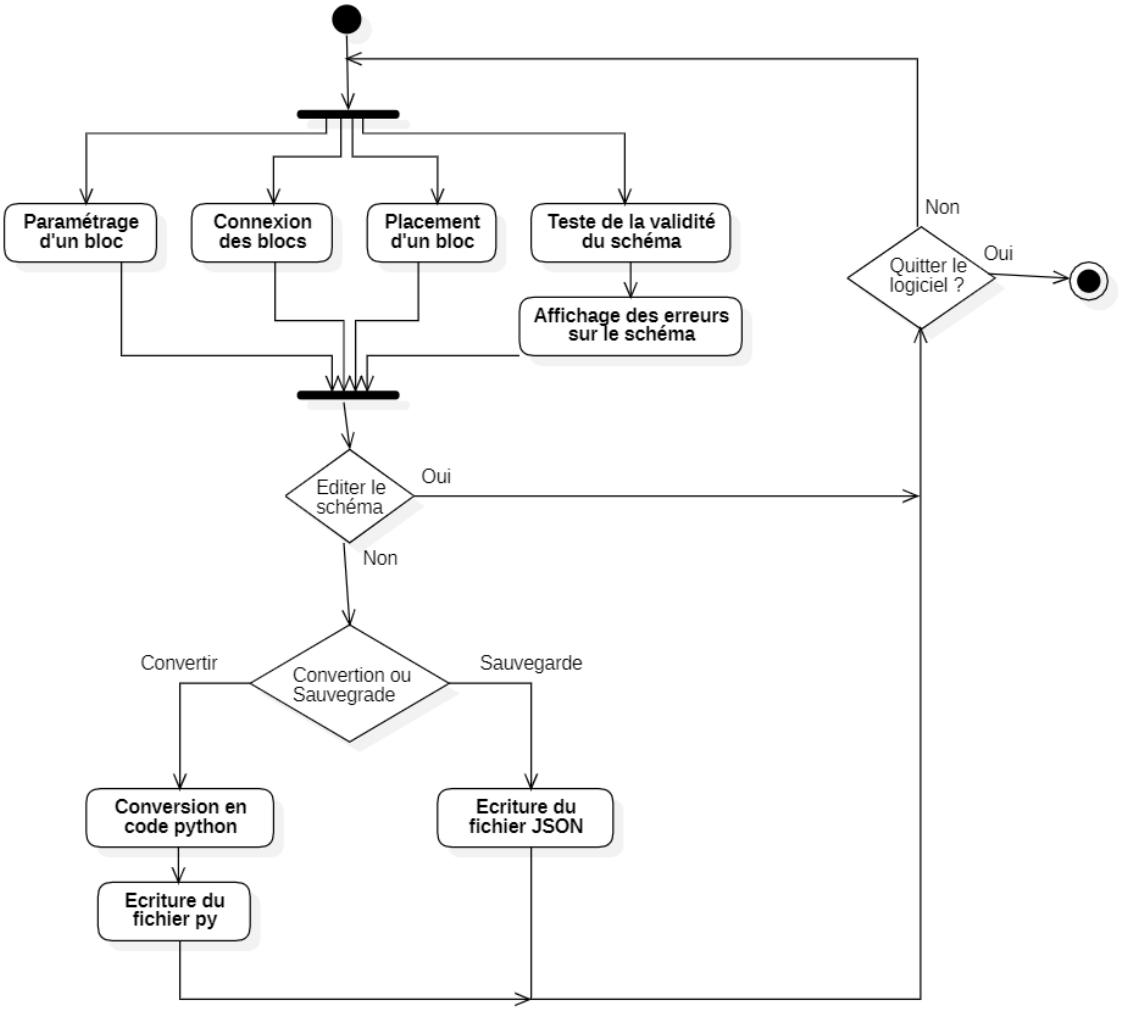

Figure 3 : Diagramme d'activité de l'interface

La **figure 3** montre la suite d'action nécessaire pour réaliser un code python et une sauvegarde de schéma. La première phase consiste à éditer le schéma à l'aide de différentes actions (connexion, création de bloc, etc...). Un contrôle de validité est réalisé en parallèle de l'édition pour indiquer à l'utilisateur d'éventuelles erreurs à corriger.

Dans un deuxième temps, il y a une phase de conversion/sauvegarde pour permettre à l'utilisateur de sauvegarder une version éditable ou convertir en code python son schéma. Cette étape constitue une sortie temporaire de la phase d'édition, mais l'utilisateur revient en édition juste après.

Il est possible de quitter le logiciel à tout moment et donc de mettre fin au processus sans nécessairement réaliser d'action.

4. Réalisation technique

4.1 Librairies utilisées

4.1.1 Librairie Node Editor

Pour éviter un travail long et fastidieux, il à été suggéré de travailler sur cette interface graphique à partir d'un code existant. Cela permet de ne pas réimplémenter les éléments de base tels que l'espace d'édition ou la sélection d'objets. Après quelques recherches, il à été décidé d'utiliser la librairie de *Node Editor* https://github.com/bhowiebkr/python-node-editor . Celui-ci est libre et gratuit, comporte déjà la plupart des fonctions élémentaires de notre cahier des charges tout en restant simple à prendre en main. Le code est relativement simple et bien documenté.

4.1.2 Librairie d'apprentissage

Il est précisé dans le sujet que cette interface devra permettre de générer un code compatible Pytorch. Après discussion avec le commanditaire et au vu de mes compétences personnelles en intelligence artificielle, l'utilisation d'une autre librairie n'est pas problématique. C'est pourquoi mon choix de librairie d'apprentissage se porte sur Keras.

4.2 Architecture détaillée

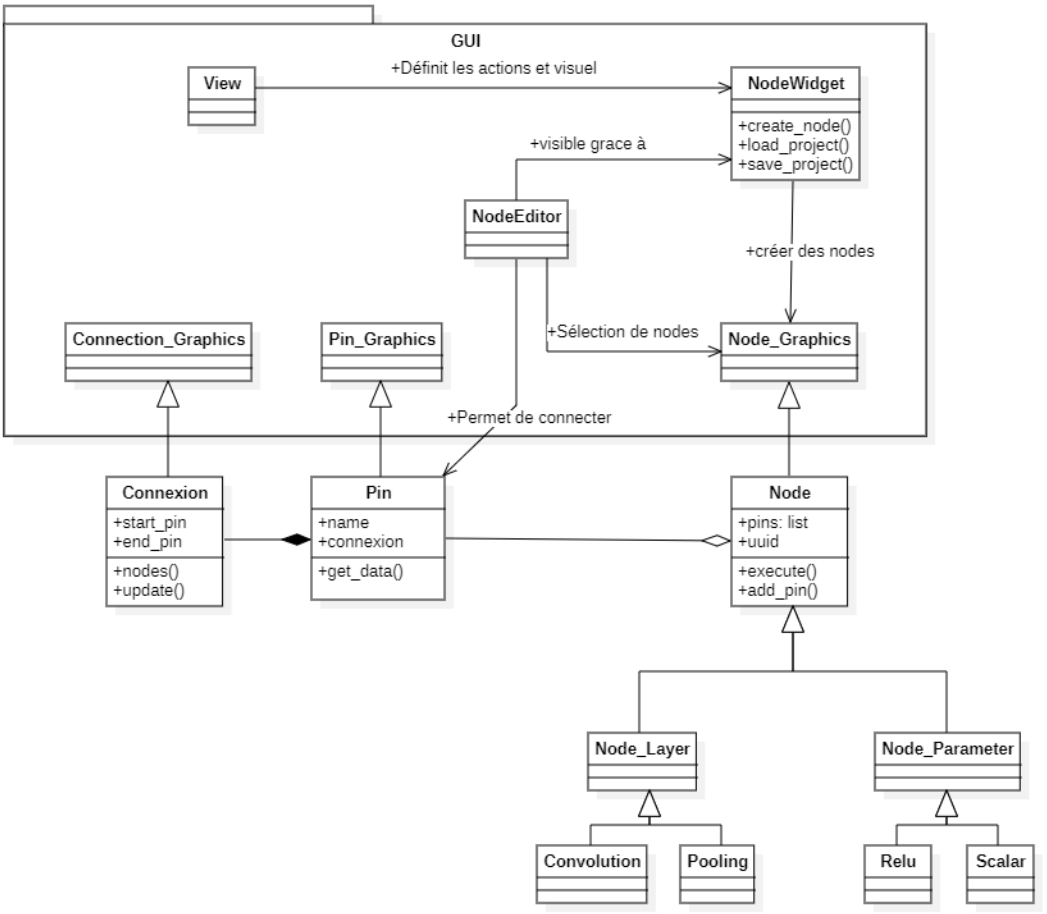

Figure 4 : Diagramme de classe détaillé avec les classes déjà implémenté par la librairie

La **figure 4** montre une architecture détaillée de l'interface en se basant sur l'architecture élaborée par la librairie. Ainsi, le bloc GUI constitue les éléments de la fenêtre, l'espace d'édition et les fonctions d'interaction avec celle-ci (Sélection, drag and drop, création de nouveaux objets). Ce bloc sera peu modifié par la suite (ajout d'un menu, affichage de paramètres). Les classes *Connexion*, *Pin* et *Node* sont à compléter avec des méthodes pour les faire communiquer et exécuter des actions. Toutes les fonctions et variables matérialisé par des blocs sont des classes qui hérite de la classe *Node,* cela permet une implémentation simple de toutes les couches utiles dans une architecture.

4.3 Méthode de sauvegarde

4.3.1 Sauvegarde du schéma éditable

La librairie de Node Editor décrite précédemment inclut déjà un système d'ouverture de fichier JSON pour charger un schéma enregistré. Celui-ci permet de charger les différents types de bloc, leur positions dans l'espace du schéma ainsi que les connexions qui les relient. Cependant, cela reste incomplet pour notre utilisation. Nous avons besoin d'enregistrer les valeurs et les méthodes indiquées en paramètre. Pour cela, nous reprenons la structure JSON déjà utilisée pour ajouter un item paramètre qui contient la liste des paramètres renseignés par l'utilisateur.

```
{
    "nodes": [
        {
            "type": "Nom_du_Node",
            "x": 4733,
            "y": 5122,
            "uuid": "fb254b28-ef69-4fc1-8619-ed4fd0581689"
            "parameters":[ { "value":
                             "method":
                             ......
                         ]
        },
        ......
    ],
    "connections": [
        {
            "start_id": "0a3b1313-e73c-4f3a-b878-7a7a8a76518b",
            "end_id": "4d4c8ab7-4047-4d3c-a00d-afec8d2d2ab4",
            "start_pin": "output",
            "end_pin": "input"
        },
        ......
    ]
}
```

Figure 5 : Exemple de format de sauvegarde d'un schéma

La structure du JSON envisagé est décrite **figure 5**. Il sera possible lors du développement d'ajouter des items si cela est nécessaire.

4.3.2 Sauvegarde du code convertie

La conversion en code python et par la même occasion la sauvegarde de celui-ci est réalisée lors de l'exécution du schéma, c'est-à-dire l'exécution des blocs qui le compose. Cela va écrire dans un fichier py le code associé au schéma pour pouvoir le réutiliser telle quelle dans un script python fonctionnelle.

5. Gestion de projet

5.1 Les outils

Dans le cadre de ce projet, plusieurs outils seront utilisés.Un dépôt Github est mis en place pour partager et archiver les versions du logiciel en développement. Les scripts python sont compilés et exécutés dans Spyder. La représentation graphique sera réalisée avec PyQt.

5.2 Les risques

Intitulé du risque	Type	Impact	Probabilité	Solutions
Prise en main d'une librairie d'apprentissage	technique	majeur	moyenne	- Bonne documentation
Manque de temps	humain	majeur	moyenne	- Bonne organisation, - réduction des fonctionnalités
Prise en main du code interface	technique	majeur	faible	- Bonne documentation
Modification ou ajout de fonctionnalités	humain et technique	mineur	faible	- Echanges réguliers avec le commanditaire

Figure 6 : Tableau des risques

La **figure 6** représente un certain nombre de risques auxquels nous pourrions faire face pendant ce projet. Le principale risque lors de la réalisation d'une interface est le manque de temps, car cela peut être très chronophage. Cependant, le choix d'utiliser une librairie déjà existante permet de réduire considérablement ce risque. Il y a encore des risques à apparaître, notamment liés à la prise en main de cette librairie.

5.3 Planning prévisionnel

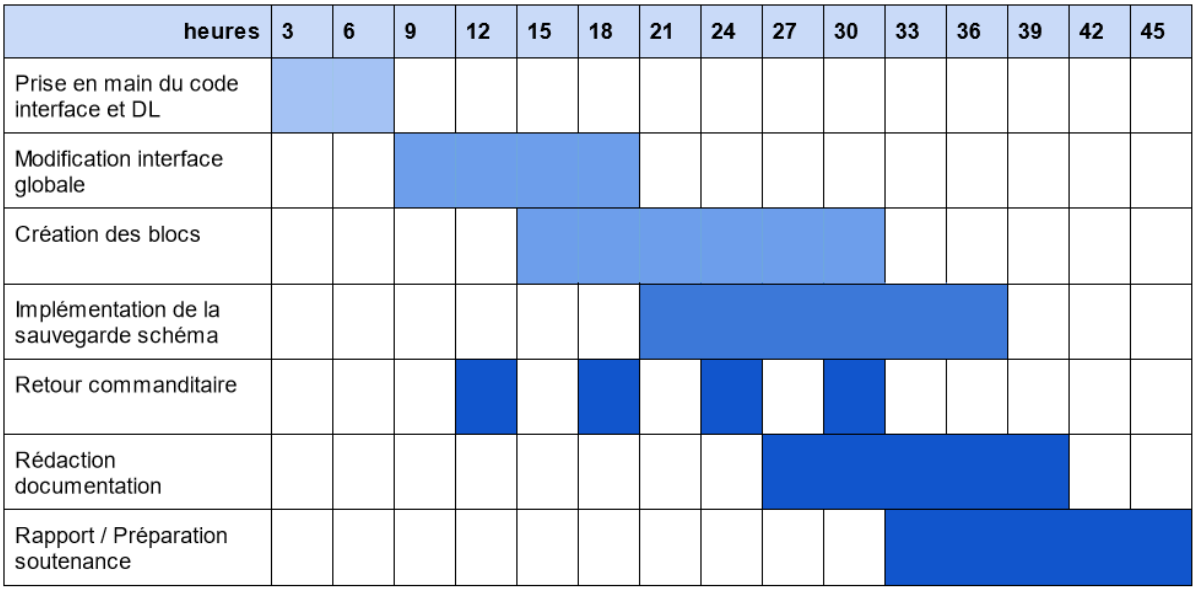

Figure 7 : Diagramme de Gantt de la phase de développement

La **figure 7** représente l'organisation et la répartition des principales tâches dans le temps. Nous pouvons remarquer qu'il y a 4 grandes étapes pour ce projet. Une première étape consiste à prendre en main le code existant pour une interface de programmation visuelle. Cette étape inclut de se renseigner sur la solution retenue PyQt. Ensuite, nous devons développer le logiciel avec plusieurs étapes réalisables en parallèle. Seule la fonctionnalité de sauvegarde du schéma est mise à part car cela nécessite un logiciel relativement abouti.

Pendant toutes ces étapes de développement, des points réguliers seront fait avec le commanditaire pour assurer un développement en accord avec les attentes formulées initialement. Enfin, les dernières étapes consisteront à réaliser une documentation et le rapport de fin de projet.

Il est à noter que les heures renseignées représentent les séances prévues pour ce projet. Un travail supplémentaire sera réalisé en dehors de ces séances notamment pendant les vacances de fin d'année.

6. Synthèse

Ce rapport donne un aperçu du travail à réaliser et de la manière dont je souhaite m'y prendre. Ce projet informatique nécessite la mise en place d'une conception orientée objets car celle-ci est très pertinente au vu du résultat attendu par le commanditaire. La structure du code utilisé est très intéressante et sera très bénéfique lors de la phase de développement.