Présentation technique

1. Généralités

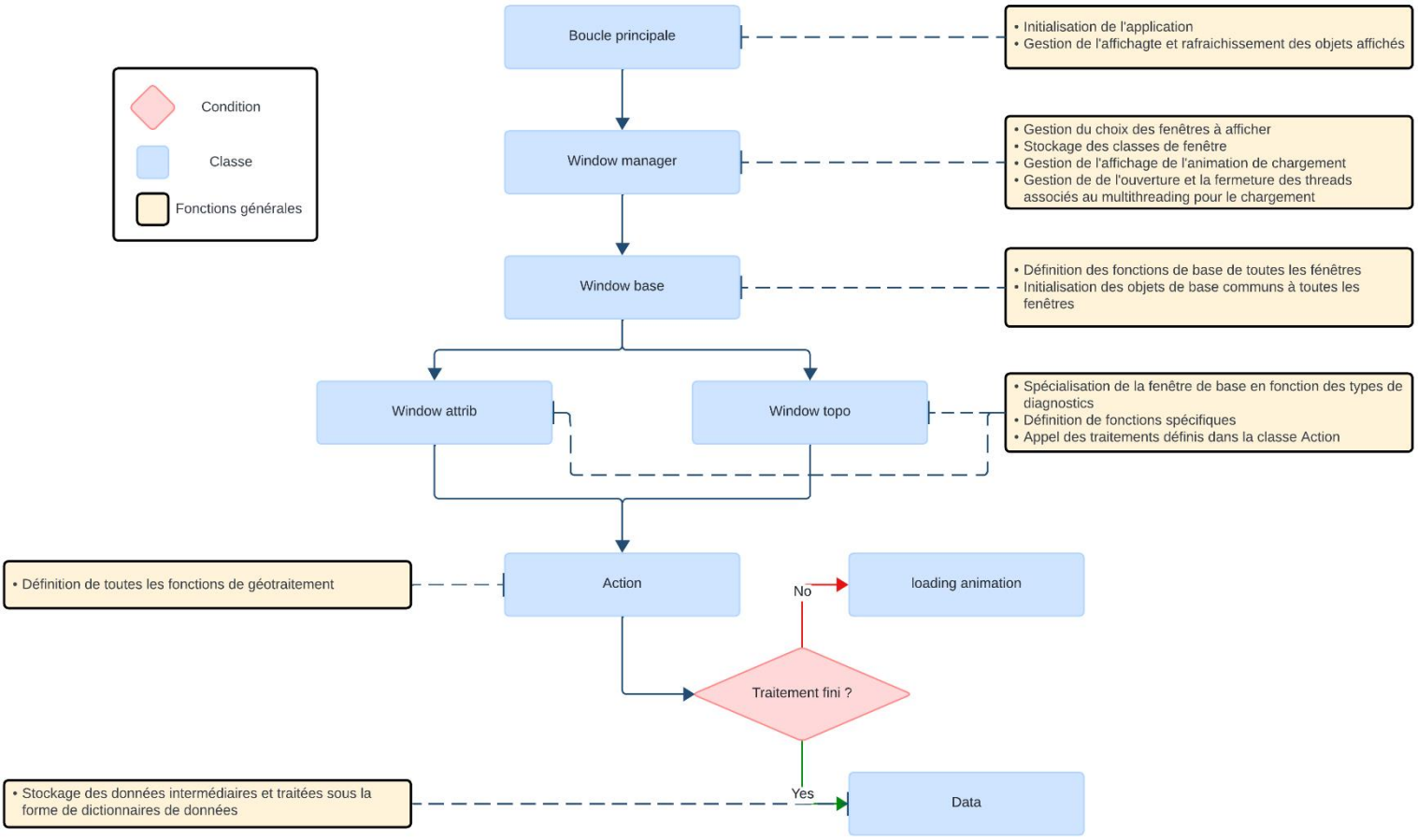
L'application fonctionne sur la base des modules python *pygame-ce* et *pygame_gui.* Ces derniers permettent de gérer efficacement l'affichage d'objets à l'écran et facilite la programmation des interactions entre les objets à l'écrans, les actions de l'utilisateurs, et les traitements en arrière-plan.

Le bon fonctionnement de l'application nécessite l'installation de différents modules dans l'environnement de travail :

- pygame-ce
- pygame_gui
- pandas
- geopandas
- sys, os, time
- numpy
- threading
- tkinter
- shapely
- gemgis
- fiona
- copy
- folium
- matplotlib
- jenkspy
- webbrowser
- Levenshtien
- collections
- json

2. Fonctionnement

La figure ci-dessous représente le fonctionnement général de l'application.



Lorsque l'application est lancée, une boucle *while* est initialisée. Une horloge est lancée et détectera à chaque *tick* des actions spécifiques que nous avons choisies. Par exemple, nous avons défini dans cette classe, que la seule manière de stopper la boucle sera de cliquer sur la croix rouge en haut à droite de la fenêtre de l'application. Cela stoppera tous les processus, et fermera l'application. Cette boucle se situe dans le fichier *main.py*.

La boucle initialisera le manager de fenêtre. Dans ce dernier, toutes les classes représentant les différentes fenêtres que nous pouvons afficher dans l'application sont stockées. Le manager de fenêtre permet ainsi de définir le type de fenêtre que nous affichons à l'écran. Lorsqu'un bouton dont la fonction est de changer de fenêtre est cliqué, la boucle principale demande au manager de fenêtre d'effectuer l'action associée. Le code du manager de fenêtre se situe dans le fichier *window_manager.py*.

Pour créer nos fenêtres, nous avons développé la classe window base. Cette dernière rassemble toutes les fonctionnalités et tous les objets que nous affichons sur l'ensemble des fenêtres. C'est par exemple dans cette classe que nous avons programmé les fonctions permettant d'afficher des boutons, des menus déroulants, des panneaux d'affichage, etc… Le code de l'organisation de base des fenêtres se situe dans le fichier *window_base.py*.

Chaque fenêtre est une spécialisation de la classe window base dans laquelle nous avons programmé des comportements spécifiques aux fonctions que nous voulions pour cette

dernière. Ainsi, chaque fenêtre utilisée dans le diagnostic topologique se situe dans le fichier *window_topo.py* sous la forme d'une classe. De la même manière, chaque fenêtre utilisée dans le diagnostic attributaire se situe dans le fichier *window_attrib.py* sous la forme d'une classe.

C'est dans ces fenêtres que nous avons programmé les logiques de remplissage des dictionnaires de données en fonction des choix de l'utilisateurs pour les associations faites à partir des différents menus déroulants. De plus, nous y trouvons les boutons qui permettront de lancer les différents diagnostics, appelant les fonctions de la classe action. Cette dernière regroupe toutes les fonctions associées aux géotraitements, import et export de données. Le code définissant la classe action se situe dans le fichier *action.py*.

Au lancement de chaque traitement lourd, nous avons mis en place un système de multithreading. Ce type de système permet à l'application de ne pas se figer dans le temps lorsqu'un traitement est lancé et de le faire fonctionner en arrière-plan. Cela nous permet d'afficher une animation de chargement en bas à droite de l'écran, pour indiquer à l'utilisateur que son traitement est en cours. Les logiques de lancement de multithreading se situent dans les fenêtres elles-mêmes pour le lancement, et dans le manager pour la détection de la fin du traitement et de l'arrêt de l'animation. Le code associé à l'animation de chargement se trouve dans le fichier *loading_animation.py* sous la forme d'une classe.

Une fois un traitement terminé, si ce dernier a produit de la nouvelle information, cette dernière sera stockée dans la classe data. Cette dernière fonctionne comme un historique des différentes modifications que nous avons pu effectuer sur les données d'origines, pour avoir accès à tous ses états et que toute l'information du code soit centralisée.

Finalement, toute l'esthétique de l'application est contenue dans le fichier *themes.json*.