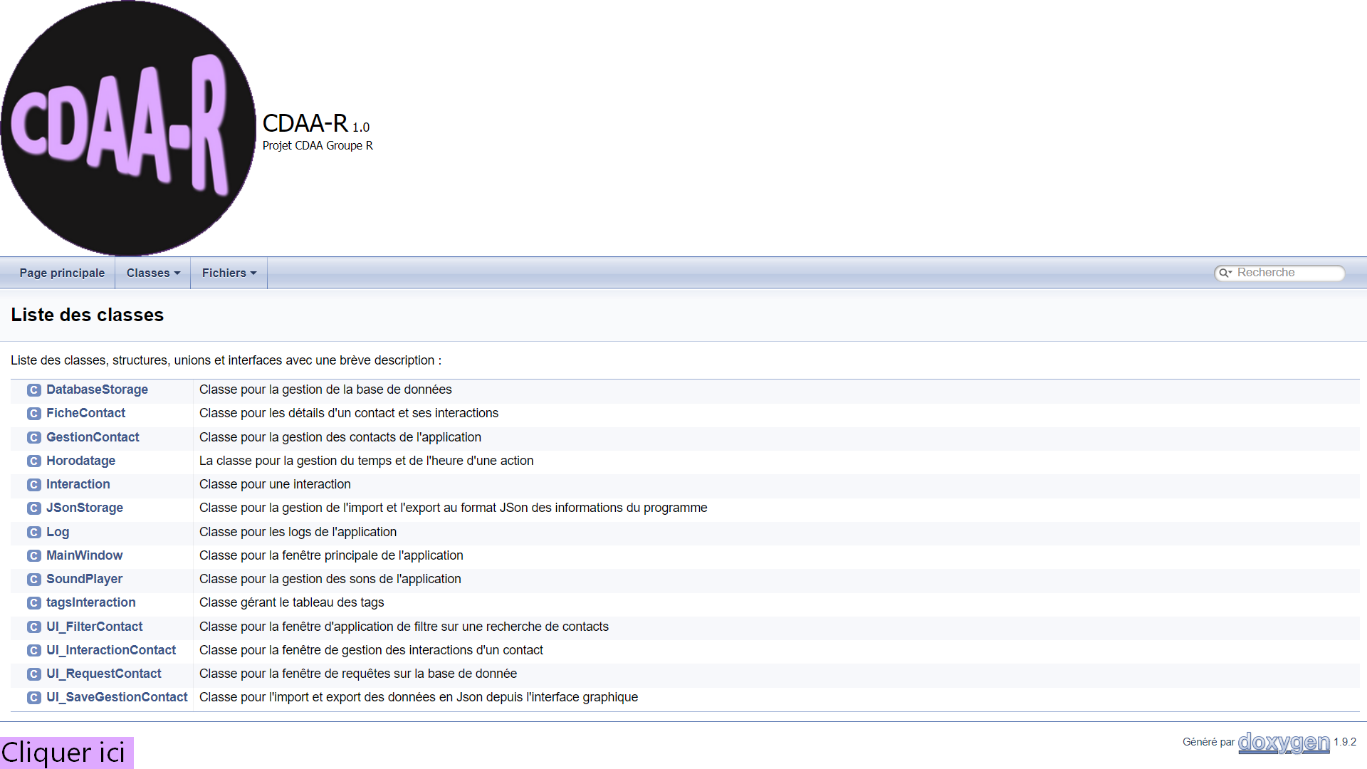


CDAA-R

LACHAUD Samuel / PAZOLA Loïs – Info S5 TP4



[](Doxygen/index.html)

2/10

SOMMAIRE

3/10

Notre projet étant en source libre sur **GitHub**, la documentation **Doxygen** est un outil indispensable pour permettre à toute personnes s’intéressant au projet de savoir rapidement les composants et le fonctionnement de celui-ci. On peut alors retrouver cette documentation **ci-dessous** ou à l’adresse suivante :

* <https://github.com/samlach2222/ProjetCDAA-R/tree/main/Doxygen/html>

Le projet CDAA-R est un projet développé en **QT** (partie graphique) et en **C++** (partie métier). Celui-ci consiste en la création d’une application de gestion de contacts avec toutes les fonctions qui lui sont liées. Les notions alors abordées sont :

* La conception graphique
* La programmation Objet
* La gestion de base de données **SqLite**
* La gestion de fichiers **JSON**
* Les différents contrôleurs de QT

Nous allons donc présenter dans ce rapport notre application finie, des coulisses de celle-ci, jusqu’à son installation.

Documentation Doxygen

Qu’est-ce que CDAA-R

1. Introduction



Diagramme de Classe

1. Conception

4/10



5/10

1. Package Métier/Logic

Dans ce premier package, la classe principale est la classe **GestionContact**, Elle est donc composée de plusieurs contacts, elle gère donc ceux-ci. Pour toutes modifications, on a une agrégation de la classe **Log** sur **GestionContact**, afin de notifier tout changement. A un contact on peut ajouter des interactions, **FicheContact** est donc composée d’**Interactions**. Une interaction est elle-même composée d’un ou plusieurs **tagsInteraction** (tags date et todo). Afin de toujours savoir quand un ajout a été fait, les classes **FicheContact**, **Interaction** et **Log** sont horodatées avec la classe **horodatage** dans des liaisons d’agrégation

Avec ces compositions nous avons donc des liaisons qui sont grammaticalement logiques, en effet, s’il n’y a plus de gestions de contacts, les contacts n’existent plus, s’il n’y a plus de contact, il n’y a plus d’interactions liées et de même pour les tags. Nous avons donc ainsi un respect des principes SOLID

Enfin, la dernière classe est SoundPlayer, c’est une classe statique, elle est appelée sans être instanciée, elle n’a donc pas de liaisons UML.

1. Package Interface/IHM :

Au sein de ce package, les liaisons sont plus faciles. Toutes les classes UI\_nomDeLaClasse sont des composants de **MainWindow**. En effet si celle-ci est détruite, il n’y a plus de programme, donc toutes les autres fenêtres formées par les classes UI sont fermées, il y a donc compositions.

Si les liaisons intra-package sont faciles, les liaisons extra-packages sont plus complexes. **MainWindow** est composée de **GestionContact** du package précédent. Par la même occasion, **UI\_SaveGestionContact** récupère **GestionContact** pour l’import et l’export. Enfin **UI\_InteractionContact** récupère **FicheContact** pour récupérer directement les interactions de ce contact.

1. Package Stockage/Storage :

Pour finir ce package est composé uniquement de classes statiques, il ne comprend donc aucunes liaisons UML. 

Explication des liaisons interclasses

Nous trouvons sur la feuille précédente le diagramme UML de Classes présentant notre projet. Il est composé de la partie **Métier/Logic** qui forme l’ensemble des classes gérant les structures et les choses internes, la partie **Interface/IHM** gérant toutes les classes visuelles, avec des fenêtres montrées au joueur, et enfin la partie **Stockage/Storage** gérant les entrées et sorties du projet que ce sois par fichier ou par base de données. Les classes qui composent ces trois packages et leurs liaisons seront décrites dans la partie suivantes.



1. Installation

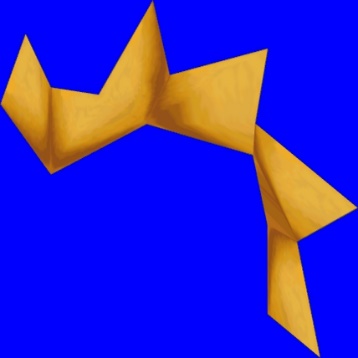


6/10

6 / 9



Application des textures

Une image contenant nature, ciel nocturne

Description générée automatiquementUne image contenant texte

Description générée automatiquement

Une fois les couleurs de Spyro appliquées, nous passons aux textures, la première texture portera sur les yeux, la seconde sera une texture enveloppante sur une sphère.

Notre première texture s’applique sur les deux faces du devant sur la sphère qui forme la tête de Spyro, elles sont appliquées face par face. C’est l’ajout qui manquait le plus au réalisme. Notre deuxième texture supprime le triangle de crinière et le remplace par sa texture avec transparence. Pour cela, nous avons mis une couleur bleue partout où fallait être transparent sur l'image, puis après avoir chargé la texture en RGB, nous les convertissons en RGBA en vérifiant l'intensité de bleu de chaque pixel et si l'intensité est suffisamment haute on met l'alpha à 0 (totalement transparent) et sinon à 255 (totalement opaque). A noter que le triangle de crinière est maintenant un rectangle afin de mettre plus facilement une texture.

Voici donc Spyro avec ses yeux, sa crinière et les textures associées :

La deuxième texture est une texture enveloppante, sur une sphère. Spyro se situe dans la sphère, on appelle cette texture donnant une zone de ciel une Skybox. Spyro se trouve alors maintenant dans un environnement.

Voici donc Spyro avec son environnement et la texture associée :

8/10



Animation du modèle

Une image contenant texte, tableau blanc

Description générée automatiquementUne image contenant texte, graphiques vectoriels

Description générée automatiquement

La dernière amélioration aurait du porter sur les lumières, elles sont bien présente dans le code mais non visibles. En effet nous avons une lumière statique et une seconde lumière qui tourne sur l’axe Y autour de Spyro. Elles sont bien activées, mais la lumière qu’elles produisent n’est pas visible.

Il nous manque les normales pour le reflet de la lumière. Par manque de temps celles-ci n’ont pas pu être implémentées.

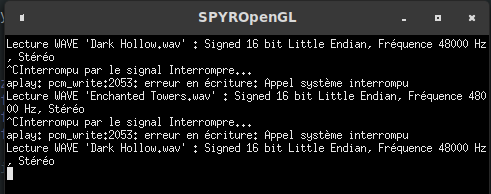
Mise en lumière

Notre Spyro manque maintenant de vie, nous lui appliquons donc une animation automatique en temps réel et une animation occasionnelle (lancée avec une touche du clavier détaillé en partie 4).

La première animation est le mouvement des ailes. L’animation s’effectue sur trois points différents, la première partie de l’aile effectue une rotation sur l’axe Z, cependant elle effectue également une rotation sur l’axe X, mais deux fois moins grande. En même temps s’effectue une rotation de la deuxième partie de l’aile de la même amplitude que la rotation en X.

La deuxième animation s’effectue lors de l’appui sur la touche de clavier, Spyro bouge alors sa partie supérieure de sa bouche de haut en bas jusqu’à la fin de l’animation sonore.

9/10



10/10

En bonus, dans SPYROpenGL, de la musique est jouée en arrière-plan, celle-ci est jouée dans un thread appelant des commandes Linux. Les trois musiques sont les suivantes :

Dark Hollow, Enchanted Towers, Sgt. Byrd's Theme

Elles sont donc jouées en arrière-plan. Il y’a quelques subtilités, quand une musique s’arrête, une autre prend le relais, mais jamais la même d’affilé, évitant donc la redondance pour l’utilisateur. On peut également passer à la musique suivante en appuyant sur ‘CTRL+C’ sur la deuxième fenêtre ouverte lors de l’exécution nous avons donc l’affichage suivant permettant de savoir les informations de la musique jouée actuellement :

Changement de musique de fond

De nombreuses touches du clavier permettent d’utiliser facilement SPYROpenGL, voici la liste des commandes permettant la navigation ou l’utilisation de fonctionnalités :

* **'z’** 🡪 Permet de zoomer sur Spyro
* **‘Z’** 🡪 Permet de dézoomer sur Spyro
* **‘i’** et **‘k’** 🡪 Permettent de déplacer Spyro sur l’axe Y
* **‘j’** et **‘l’** 🡪 Permettent de déplacer Spyro sur l’axe X
* **‘J’** et **‘L’** 🡪 Permettent de déplacer Spyro sur l’axe Z
* **‘q’** 🡪 Permet de quitter l’application
* ‘**ESPACE’** 🡪 Permet de faire bouger la bouche de Spyro avec le son de sa voix
* **Touches fléchées** 🡪 Permettent de rotationner la caméra autour de Spyro

Utilisation du clavier

Maintenant que nous avons vu la réalisation du projet et sa finition, nous allons voir les fonctionnalités associées, plus particulièrement les touches claviers utilisées et la gestion de la musique, ceci nous permettant une utilisation plus personnalisée du projet SPYROpenGL.

1. Utilisation du projet