Tech stack :

* Glfw : windowing
* IMGUI : User Interface
* GLAD : OpenGL abstraction layer
* PhysX : Physique
* SoLoud : audio
* Assimp : 3D model importer
* Stb\_image : Texture importer

Rendering API :

* OpenGL

Required Features :

* Buffers
* Shaders
* Camera
* Input
* Textures
* Models

# Documentation du Projet C++ - Survivant

## Introduction

Le but du projet Survivant est de créer un moteur de jeux. En équipe de 3, nous devrons rechercher, structurer, projeter et annoter notre plan pour s’attaquer à cette charge. Après sa complétion, nous utiliserons notre moteur pour développer un FPS puzzle-game plutôt simple.

## Configuration du Projet

### Fichiers Sources

TODO

### API de rendu

Pour ce projet nous utiliserons OpenGL pour sa simplicité relative de prise en main, son ancienneté la rendant incontournable dans l’industrie et sa compatibilité avec un large panel de configurations. De plus, la familiarité de l’équipe avec cette API et la quantité de ressources disponibles sur son utilisation permettront de produire des résultats plus rapidement et de fournir plus d’efforts sur d’autres aspects du moteur.

### Bibliothèques Externes

Cette sous-section contient la liste des bibliothèques utilisées dans ce projet, ainsi qu’une courte explication de leurs rôles et importances (Fonctionnement, Fichiers Sources, Type de Bibliothèque, Chargement).

* Explication de son fonctionnement
* Justification du choix de la librairie
* Processus de chargement de la bibliothèque
* Intégration des fichiers sources (.lib et/ou .dll précompilé(e)s, ou générés avec *FetchContent*)
* Bibliothèque statique(.lib) / dynamique(.dll)

#### GLFW (Windowing)

##### Fonctionnement

Cette librairie « Open Source » est cruciale pour la création et gestion de fenêtre compatible avec multiples API de rastérisation. GLFW fournit une abstraction multi-platform pour des applications graphiques.

##### Justification

GLFW est l’API de fenêtrage standard de OpenGL. De plus, l’équipe a de l’expérience avec la librairie.

##### Étapes de chargement

1. Initialisation avec [glfwInit](https://www.glfw.org/docs/3.3/group__init.html#ga317aac130a235ab08c6db0834907d85e)().
2. Définir la version de l’API graphique (dans notre cas 4.6) avec [glfwWindowHint](https://www.glfw.org/docs/3.3/group__window.html#ga7d9c8c62384b1e2821c4dc48952d2033)([GLFW\_CONTEXT\_VERSION\_MAJOR](https://www.glfw.org/docs/3.3/group__window.html#gafe5e4922de1f9932d7e9849bb053b0c0), 4) et [glfwWindowHint](https://www.glfw.org/docs/3.3/group__window.html#ga7d9c8c62384b1e2821c4dc48952d2033)([GLFW\_CONTEXT\_VERSION\_MINOR](https://www.glfw.org/docs/3.3/group__window.html#ga31aca791e4b538c4e4a771eb95cc2d07), 6).
3. Créer une fenêtre par défaut avec [glfwCreateWindow](https://www.glfw.org/docs/3.3/group__window.html#ga3555a418df92ad53f917597fe2f64aeb)(hauteur, largeur, titre).
4. Utiliser la librairie dans la boucle principale.
5. Sélectionner la fenêtre avec [glfwMakeContextCurrent](https://www.glfw.org/docs/3.3/group__context.html#ga1c04dc242268f827290fe40aa1c91157)(fenêtre).
6. Finir en détruisant la fenêtre avec [glfwDestroyWindow](https://www.glfw.org/docs/3.3/group__window.html#gacdf43e51376051d2c091662e9fe3d7b2)(fenêtre) et ensuite [glfwTerminate](https://www.glfw.org/docs/3.3/group__init.html#gaaae48c0a18607ea4a4ba951d939f0901)().

##### Intégration des fichiers sources

Ajoutés au projet avec *FetchContent*

##### Type de Bibliothèque

Statique

##### Sources

<https://www.glfw.org/docs/3.3/quick.html>

#### Glad (GL Loader-Generator)

##### Fonctionnement

Glad est une librairie permettant le chargement des fonctions de l’API graphique supportant OpenGL, Vulkan, WGL, EGL, GLX et OpenGL ES

##### Justification

Glad est un standard de l’industrie, simple d’utilisation et s’intégrant bien avec GLFW que nous utilisons pour le fenêtrage. De plus, l’équipe ayant utilisée cette librairie dans plusieurs projets par le passé, ce choix était pour nous une évidence.

##### Étapes de chargement

1. Initaliser l’API graphique avec **gladLoadGL** en passant une fonction de chargement (ex : **gladLoadGL(glfwGetProcAddress)** )
2. Utiliser les fonctions de l’API graphique

##### Intégration des fichiers sources

Code source versionné avec le projet

##### Type de bibliothèque

Statique

##### Sources

<https://github.com/Dav1dde/glad/wiki/C#quick-start>

<https://gen.glad.sh/>

#### ImGui (User Interface)

##### Fonctionnement

ImGui est une librairie d’interface graphique développée pour assister au développement d’engins de jeux.

##### Justification

ImGui est une librairie très légère, avec un nombre limité de dépendances et facile à prendre en main, qui priorise la vitesse d’itération et la simplicité de visualisation.

##### Étapes de chargement

1. Initialiser le contexte avec IMGUI\_CHECKVERSION() et ImGui::CreateContext().
2. Configurer les flags selon les types d’input (ex : io.ConfigFlags |= ImGuiConfigFlags\_NavEnableKeyboard).
3. Initialiser le « *backend* » (dans notre cas GLFW et OpenGL) avec ImGui\_ImplGlfw\_InitForOpenGL(GLFW\_WINDOW, true) et

ImGui\_ImplOpenGL3\_Init(GLSL\_VERSION\_STRING).

1. Au début de la boucle principale, démarrer une « *frame* » avec ImGui\_ImplOpenGL3\_NewFrame(), ImGui\_ImplGlfw\_NewFrame() et ImGui::NewFrame()
2. À la fin de la boucle principale, dessiner la « *frame* » avec ImGui::Render() et ImGui\_ImplOpenGL3\_RenderDrawData(ImGui::GetDrawData());
3. Arrêter le « *backend* » avec ImGui\_ImplOpenGL3\_Shutdown() et ImGui\_ImplGlfw\_Shutdown() puis détruire le contexte avec ImGui::DestroyContext();

##### Intégration des fichiers sources

Ajoutés au projet avec *FetchContent*

##### Type de Bibliothèque

Statique

##### Sources

<https://github.com/ocornut/imgui#dear-imgui>

<https://github.com/ocornut/imgui/wiki/Getting-Started>

#### PhysX (Physique)

##### Fonctionnement

PhysX permet de définir des acteurs et les déplacer en suivant les lois de mécanique classiques dans un environnement tri-dimensionnel. Elle supporte aussi les collisions, jointures, actionneurs et des tests physiques de rayons, de balayage et de chevauchement.

##### Justification

PhysX est une librairie de bas niveau très puissante, bien que complexe, qui nous permettra de créer un environnement réaliste. Elle peut également être configurée pour utiliser le GPU pour certaines tâches demandantes (CUDA).

##### Étapes de chargement

1. Initialiser avec PxCreateFoundation(PX\_PHYSICS\_VERSION, callback, error\_callback)
2. Connecter la foundation à un socket avec mPvd = PxCreatePvd(\*gFoundation), PxPvdTransport\* transport = PxDefaultPvdSocketTransportCreate(PVD\_HOST, 5425, 10), mPvd->connect(\*transport,PxPvdInstrumentationFlag::eALL). (Cooking?)
3. Créer un objet physique avec mPhysics = PxCreatePhysics(PX\_PHYSICS\_VERSION, \*mFoundation,

PxTolerancesScale(), recordMemoryAllocations, mPvd).

1. Utiliser des ‘’callback‘’ dans la boucle principale.
2. Finir avec mPhysics->release() et mFoundation->release().

##### Intégration des fichiers sources

TODO

##### Type de Bibliothèque

TODO

##### Sources

<https://gameworksdocs.nvidia.com/PhysX/4.0/documentation/PhysXGuide/Manual/Introduction.html#a-brief-overview-of-physx>

<https://docs.nvidia.com/gameworks/content/gameworkslibrary/physx/apireference/files/group__foundation.html>

#### SoLoud (Audio)

##### Fonctionnement

SoLoud est un engin créé pour simplifier la gestion audio.

##### Justification

Malgré la simplicité d’utilisation de cette librairie, SoLoud contient une multitude de fonctionnalités tel que l’ajustabilité du volume et de la vitesse de lecture, modifier l’audio avec des filtres, ajouter des sons à une queue, du support pour différents formats de fichiers et du bouclage audio transparent.

##### Étapes de chargement

1. Initialiser un ‘’core’’ avec SoLoud::Soloud gSoloud et gSoloud.init().
2. Importer un son SoLoud::Wav gWave avec gWave.load("pew\_pew.wav").
3. Modifier le son si désiré, pour ensuite le jouer avec gSoloud.play(gWave).
4. Finir avec gSoloud.deinit()

##### Intégration des fichiers sources

Ajoutés au projet avec *FetchContent*

##### Type de Bibliothèque

Statique

##### Sources

<https://solhsa.com/soloud/index.html>

#### Assimp (3D Model Importer)

##### Fonctionnement

La bibliothèque Open Asset Import Library (Assimp) permet le chargement et la manipulation de modèles 3D provenant de divers formats de fichiers.

##### Justification

Assimp offre une solution puissante et flexible pour traiter des données de modèles 3D dans une application. De plus, Assimp prend en charge une variété de formats, facilitant l'importation de modèles provenant de différentes sources.

##### Étapes de chargement

1. Créer une instance de la classe **Importer** (ex : **Assimp::Importer importer**)
2. Charger le fichier avec **importer.ReadFile** en spécifiant le chemin du fichier et les « *flags* » d'importation nécessaires (ex : **const aiScene\* scene = importer.ReadFile("model.obj", aiProcess\_Triangulate | aiProcess\_JoinIdenticalVertices | aiProcess\_SortByType)** )
3. Traiter les données de la scène Assimp chargée

##### Intégration des fichiers sources

Ajoutés au projet avec *FetchContent*

##### Type de Bibliothèque

Statique

##### Sources

<https://www.assimp.org/>

<https://github.com/assimp/assimp>

<https://assimp-docs.readthedocs.io/en/latest/usage/use_the_lib.html>

#### STB image (Texture Importer)

##### Fonctionnement

La bibliothèque STB image est une bibliothèque simple et légère permettant le chargement d’images dans divers formats.

##### Justification

STB image offre une solution rapide et facile à utiliser pour incorporer des images dans des applications graphiques.

##### Étapes de chargement

1. Charger le fichier avec **stbi\_load** (ex : **stbi\_load("texture.png", &width, &height, &channels, 0)** )
2. Une fois que les données chargées ne sont plus nécessaires, libérer la mémoire avec **stbi\_image\_free**

##### Intégration des fichiers sources

Fichier stb\_image.h versionné avec le projet

##### Type de bibliothèque

Statique – Header Only

##### Sources

<https://github.com/nothings/stb/blob/master/stb_image.h>

## Conclusion

Résumez brièvement les points clés de votre documentation.

TODO