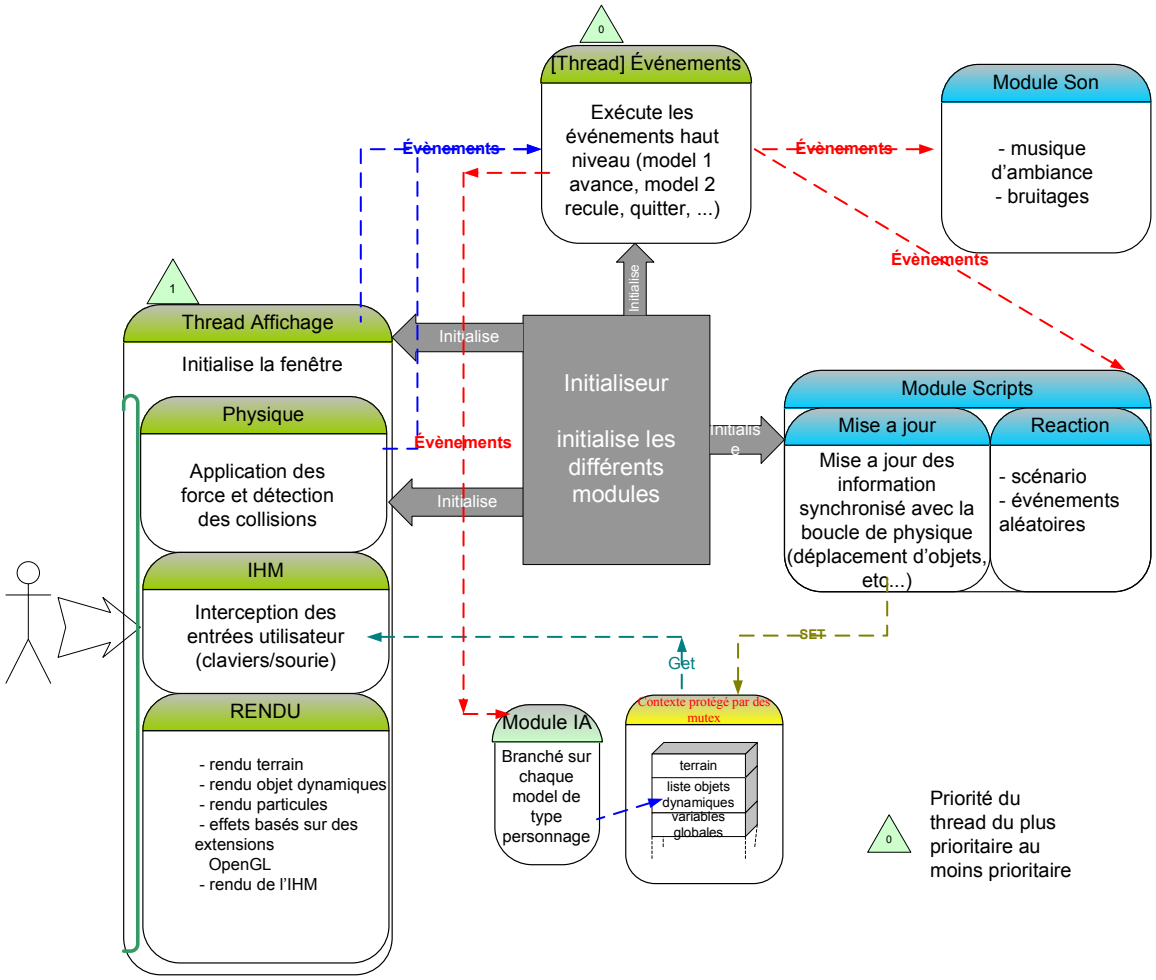
PAVILLON-NOIR : LE JEU

DOCUMENTATION TECHNIQUE

TABLE DES MATIERES

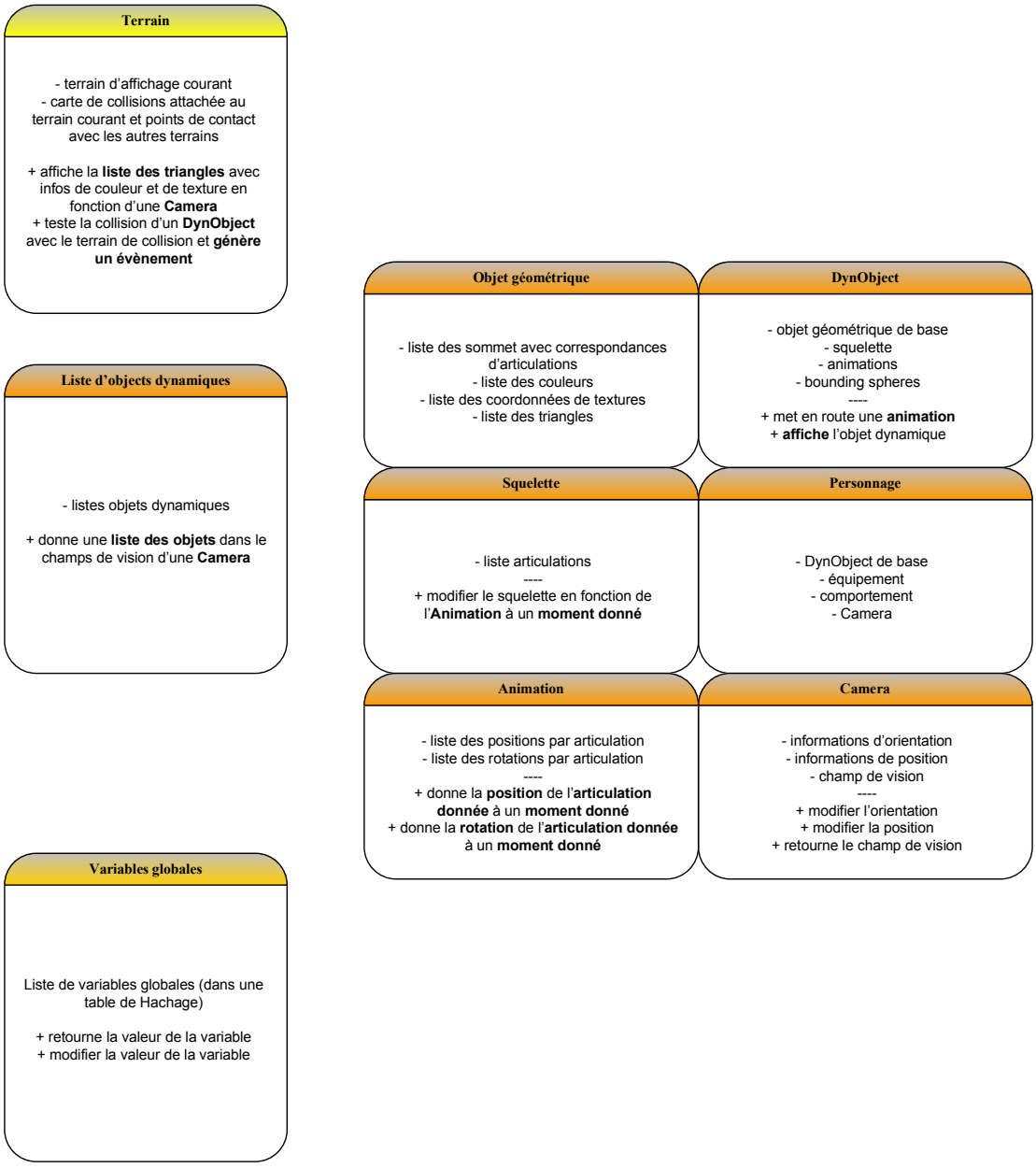
ARCHITECHTURE TECHNIQUE

GESTIONNAIRE DE RESSOURCES SYSTEME :



Le gestionnaire de ressources systèmes est chargé d'initialiser l'ensemble des modules et de leur fournir une interface simple avec leur environnement.

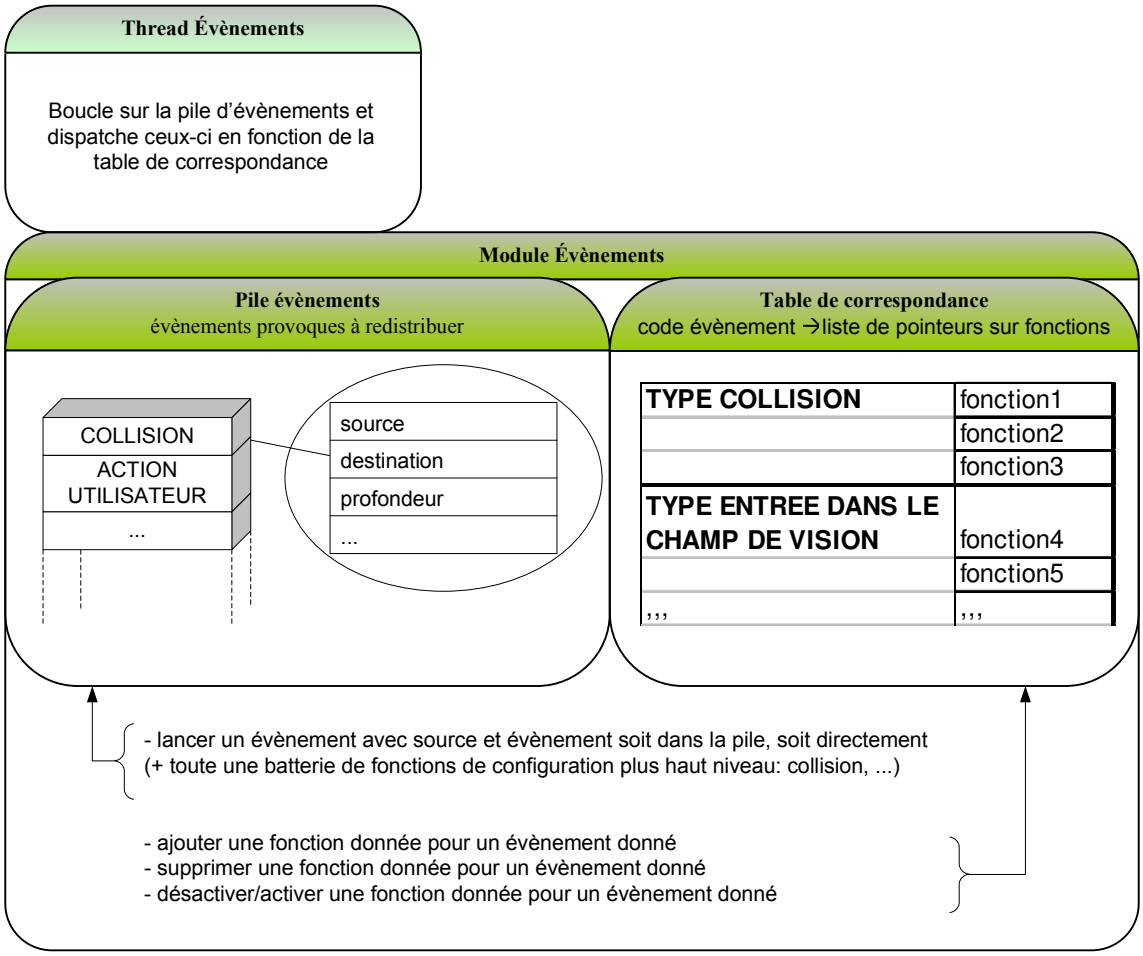
CONTEXTE:



Le contexte est un espace de mémoire partagée dans laquelle se trouvent toutes les données relatives à chaque module (modèles, animations, textures, variable globales, terrain…). Les objets sont automatiquement protégés pour l'accès via plusieurs threads.

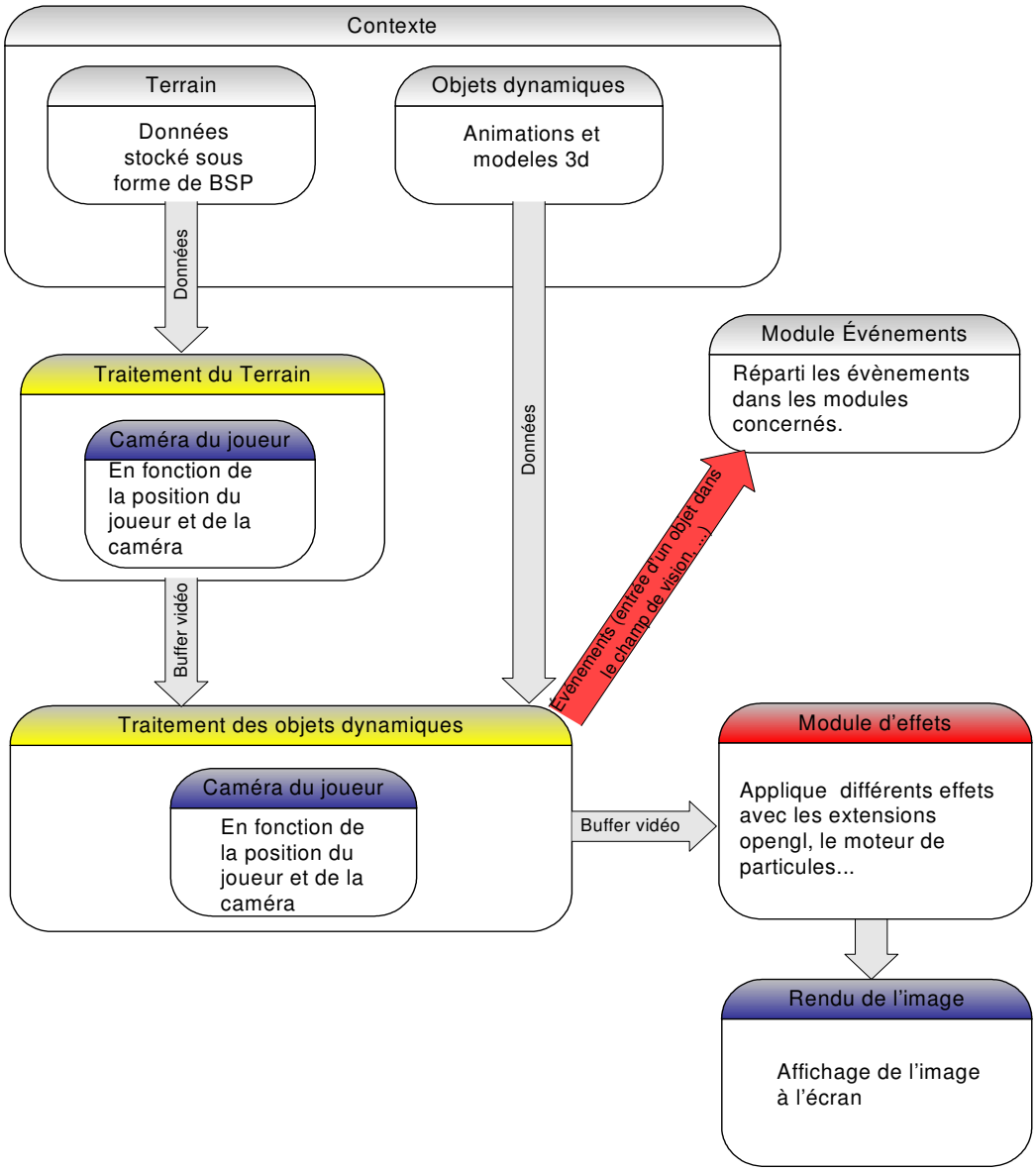
Tous les objets sont exportables dans un format qu'il est possible de recharger pour permettre de faire des sauvegardes à tout moment.

MODULE D'EVENEMENT:



La gestion des événements se fait sous forme de pile. Chaque événement de cette pile a un type donné et est envoyé à tous les modules qui l'ont demandé. Une table de correspondance permet de faire le lien entre le code d'un événement et un pointeur sur la ou les fonctions à appeler qui lui sont associées. Il est possible d'ajouter ou de supprimer des éléments dans cette pile et d'activer ou de désactiver une fonction donnée pour un événement donné.

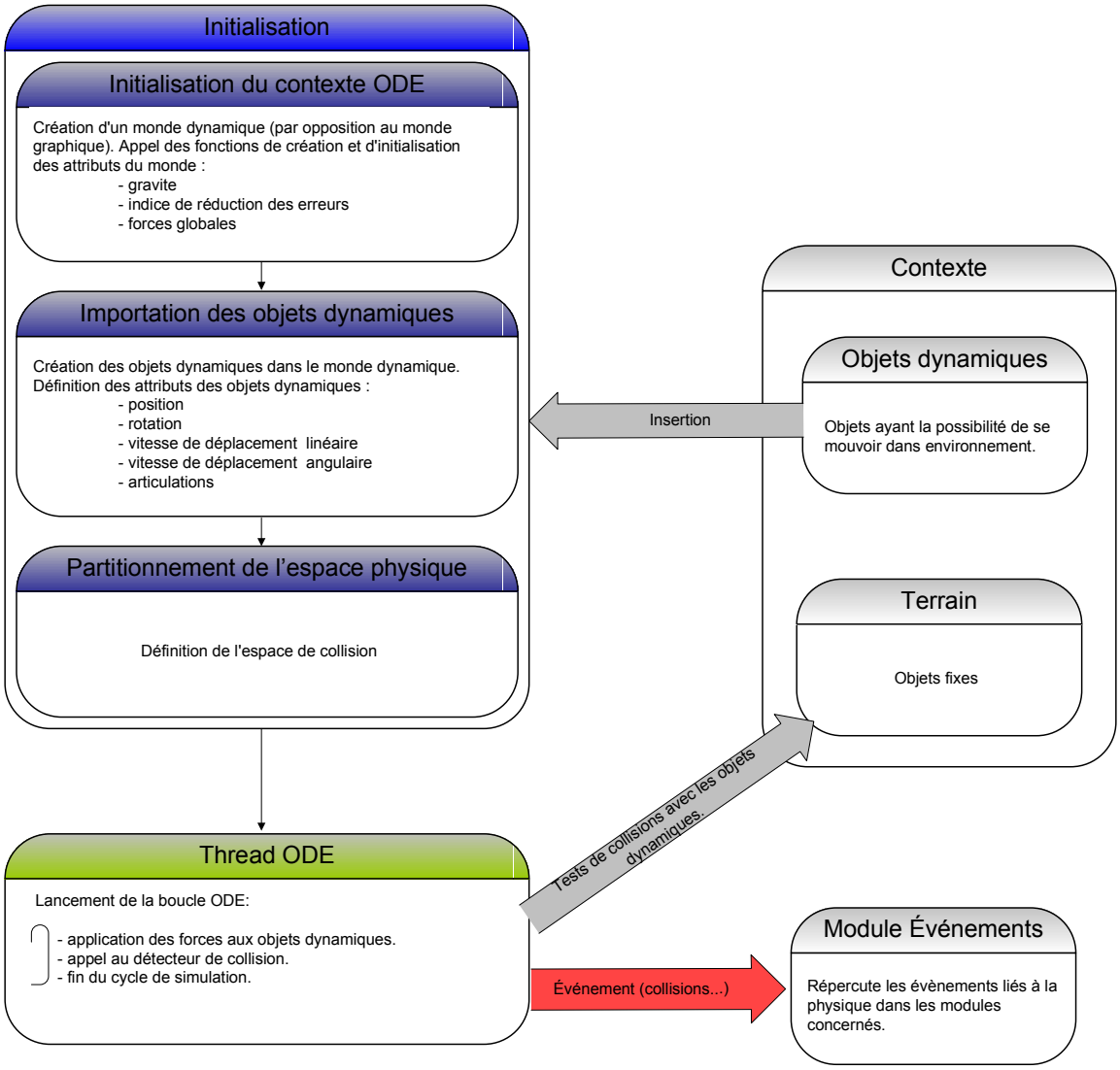
MODULE DE RENDU VISUEL :



Quand le module de rendu est appelé, on détermine, à partir de la position de la caméra dans l'espace, la partie du monde qu'il doit rendre. Ensuite, on va déterminer les personnages et les objets dynamiques présents dans cet espace et les afficher. Enfin, le module d'effets graphiques va se charger d'appliquer des transformations pour donner plus de réalisme au jeu.

Une fois toutes ces opérations effectuées, on peut afficher l'image du jeu et recommencer le traitement d'une nouvelle image jusqu'à ce qu'un autre module reprenne le contrôle du programme.

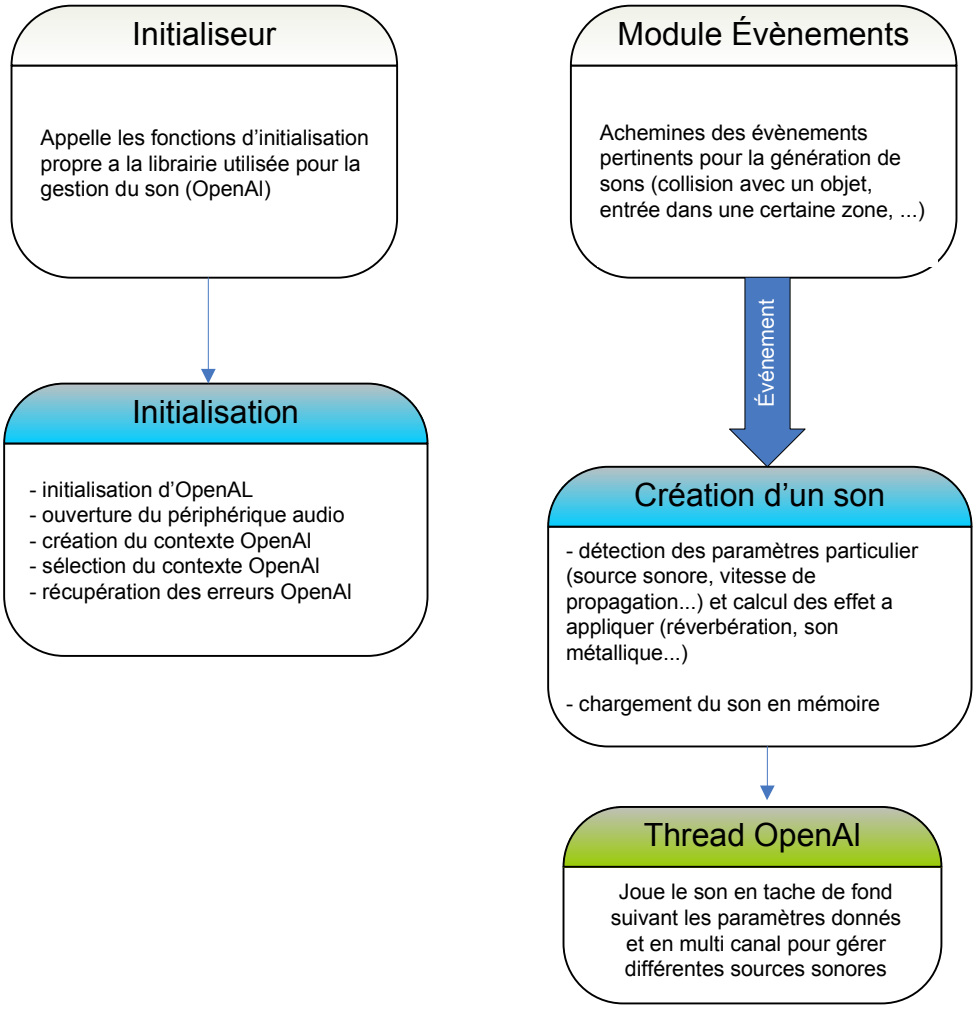
GESTION DE LA PHYSIQUE :



Le simulateur physique est basé sur OPAL, surcouche d'ODE (pour Open Dynamics Engine), seul simulateur physique distribué sous licence GPL.

Il permet la modélisation de la dynamique du monde virtuel. Le simulateur physique s'interface avec les module de partitionnement de l'espace et de rendu.

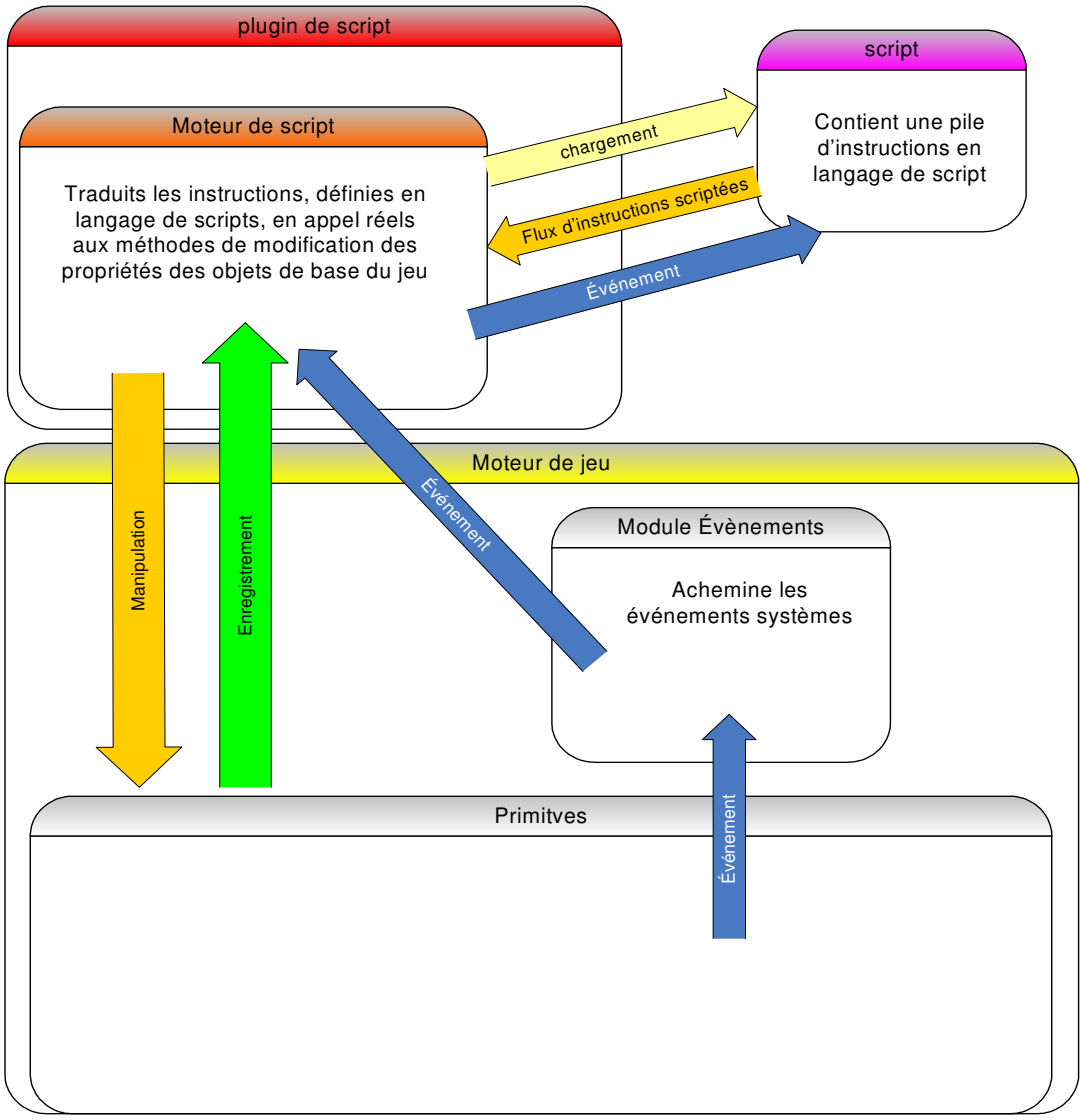
MODULE DE RENDU SONORE :



Le moteur de son est déclenché par la réception d'un événement correspondant. L'action créée est alors répercutée via la librairie OpenAL qui permet d'adapter le son à la source et à tous les paramètres d'environnement (distances, vitesse de propagation…).

Le son est alors joué sur un certain canal et tous les paramètres sont ensuite réinitialisés.

MODULE DE SCRIPTING :



Le moteur de script a pour fonction d'interpréter des scripts de manipulation de l'environnement du jeu. Le but étant de pouvoir décrire des séquences scénaristiques animées utilisant le moteur du jeu, par exemple la mise en scène d'une bataille navale.

- Les scripts doivent permettre les manipulations suivantes sur les entités du jeu :
    - Création
    - Suppression
    - Déplacement
    - Configuration
    - Lancement d'animations spécifiques
    - Lancement d'actions spécifiques.

- Chaque instruction est lancée avec au moins les paramètres suivant:
    - Un identifiant d'entité
    - Une donnée spécifique à l'action à exécuter

- Chaque script est exécuté de façon itérative dans un thread virtuel dépendant du moteur de jeu. Il est donc possible de lancer plusieurs scripts en même temps dans différents thread virtuels.

- Le temps d'exécution de chaque instruction est dépendant des objets du jeu ciblé qui doivent réaliser les actions demandées. Ce temps varie en fonction des objets ciblés et de la façon dont leurs paramètres ont été configurés, par exemple un homme se déplacera moins vite d'un point A à point B qu'un cheval.

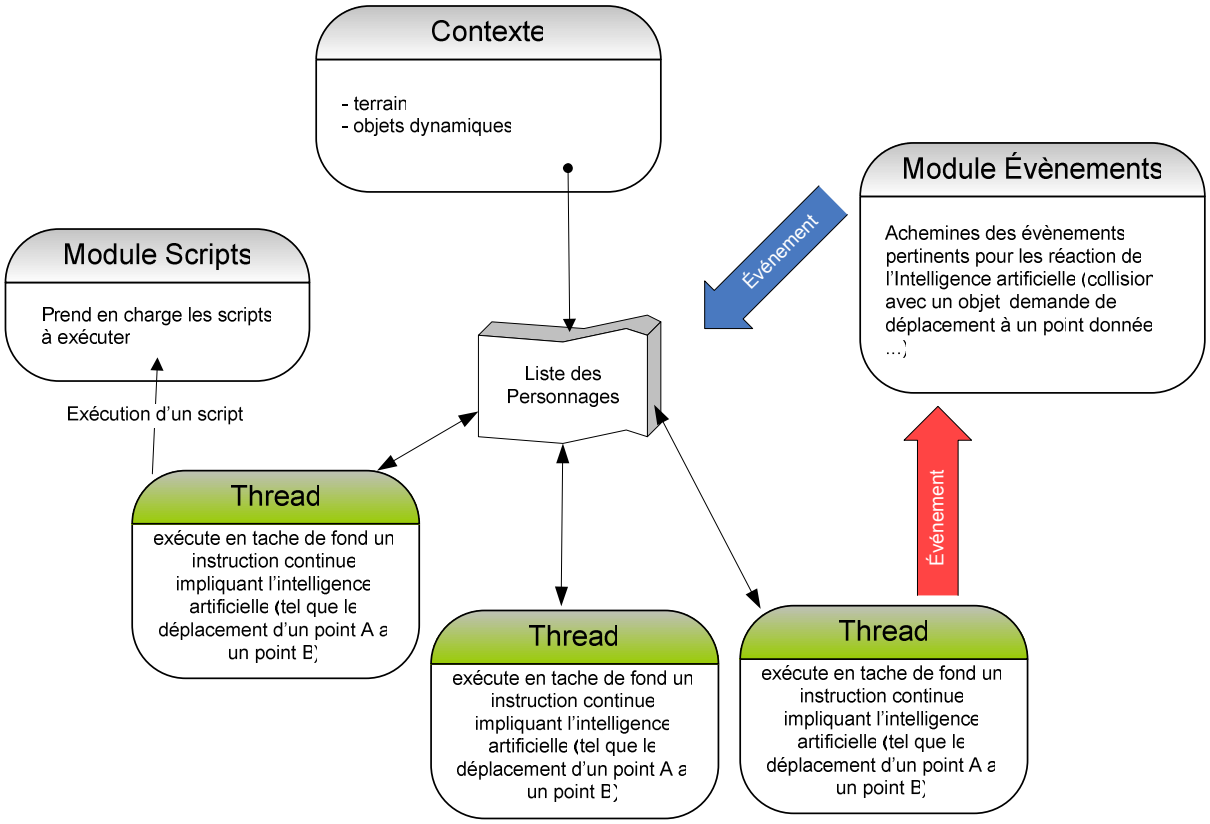
- Quand un objet a terminé l'exécution d'une action il doit envoyer un évènement au moteur de script pour qu'il puisse passer à l'instruction suivante.

- Chaque objet a sa propre façon d'exécuter les actions qui lui sont demandées, un homme se déplacera en marchant alors qu'un oiseau le fera en volant.

- Un script peut être lancé, mis en pause, ou arrêté par le moteur de script soit par un objet du jeu (exemple : une boite invisible présente dans le monde virtuel qui déclenche un script quand le joueur la traverse), soit par un autre script.

- Il est aussi possible de temporiser l'exécution d'une instruction. C'est-à-dire de finir un certain temps avant l'exécution de la prochaine instruction.

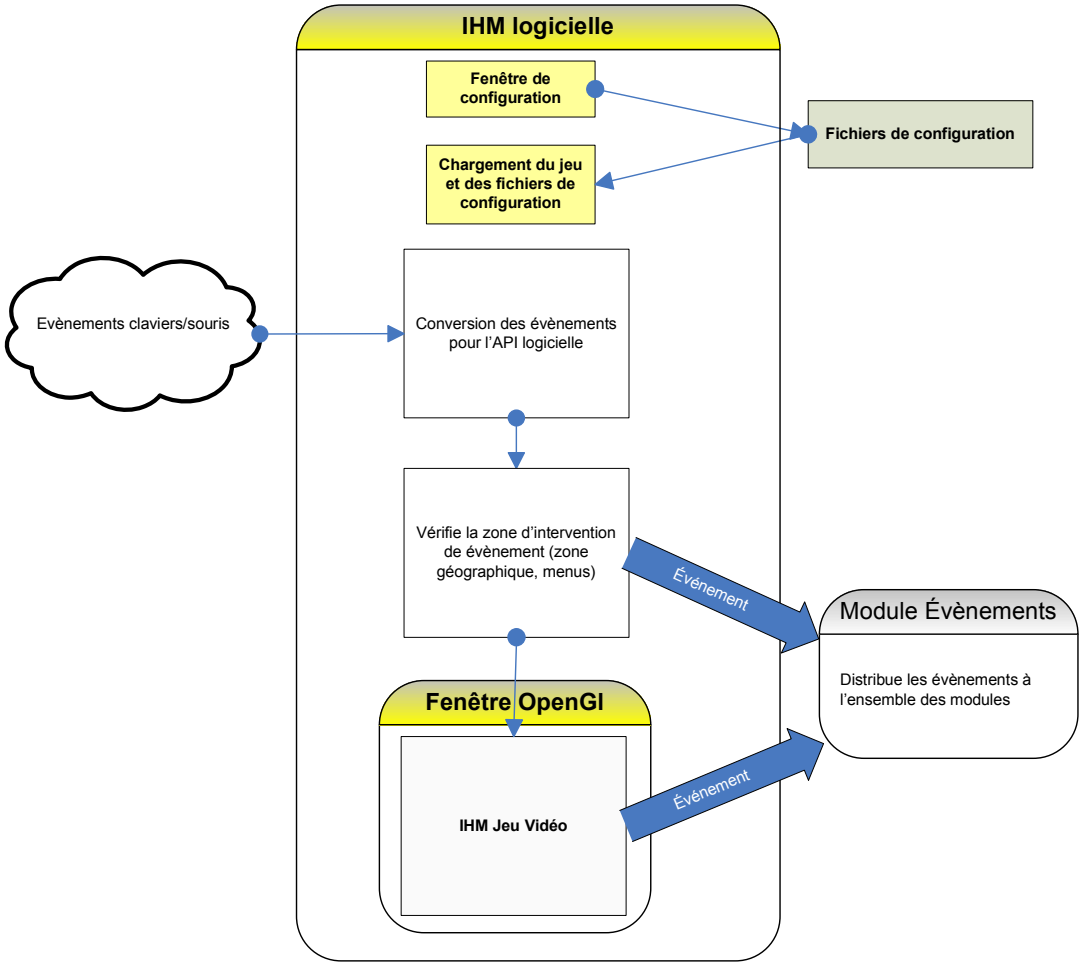
- Un script peut prendre la main sur le joueur et les cameras.



Un thread est créé pour chaque personnage et va servir à calculer les instructions continues telles que le déplacement d'un point A à un point B.

Le personnage réagit à un événement selon les différents cas rencontrés et le comportement qui lui est dicté par sa classe d'intelligence artificielle. Pour cela, il peut ajouter des événements via la pile. Il pourra aussi exécuter un script qui prend le relais des actions à réaliser.

IHM :



L'interface utilisateur constitue le lien entre le jeu et le joueur, c'est ce qui va lui permettre d'intervenir sur l'environnement.

Une interface de type logicielle permet dans un premier temps de lancer une fenêtre servant à configurer et modifier certains paramètres avant que le jeu ne se lance. Elle va ensuite lancer une fenêtre OpenGL en plein écran intégrant le jeu.

C'est elle qui va transmettre tous les événements claviers et souris générés par le joueur pour qu'elles soient traduites en actions haut niveau (avance, tourne, saute, etc.), les actions qui affecteront l'interface du jeu seront envoyées à celle-ci, les autres seront redirigées directement vers le module événementiel.

API DE SCRIPT

SON ROLE

L'API de script ""PnScript"" a été conçue dans le but d'offrir, aux concepteurs du jeu Pavillon noir, la possibilité de manipuler les différentes entités disponibles dans un niveau (personnages, éléments du décor dynamiques, cameras …). Elle permet aussi de lancer le niveau suivant. Les manipulations sur les entités ne se limitent qu'aux actions que chacune d'entre elle est capable d'exécuter. C'est-à-dire que toutes les actions qu'une entité peut réaliser, l'API peut la déclencher.

LE LANGAGE DE L'API

Cette API est développée pour le langage de script LUA. Pour plus d'informations sur LUA, je vous invite à consulter le site officiel http://www.lua.org.

SPECIFICATIONS

FONCTIONS

- **scriptID getScriptId()**

Retourne l'identifiant du script dans lequel on se trouve. Cet identifiant est utile pour les fonctions de manipulation de script.

- **entityID spawnEntity(entityType, coordX, coordY, coordZ)**

Permet de faire apparaître une nouvelle entité à une coordonnée spécifiée sur du niveau. Elle revoie un identifiant unique de l'objet.

- **propertyValue getProperty(entityID, propertyName)**

Retourne l'état de la propriété *propertyName* de l'entité *entityID*.

- **void setProperty(entityID, propertyName, value)**

Assigne la valeur *value* pour la propriété *propertyName* de l'entité *entityID*.

- **succeedState doAction(entityID, actionName, …)**

Déclenche l'action *actionName* de l'entité *entityID*. Cette fonction peut prendre un certain nombre de paramètres supplémentaires qui varient en fonction de l'action demandée. Une fois l'action terminée, la fonction retourne un booléen *TRUE* en cas de succès ou *FALSE* en cas d'échec.

- **boolean destroy(entityID)**

Fait disparaitre un objet présent dans la partie. Retourne un booléen *TRUE* en cas de succès ou *FALSE* en cas d'échec.

- **entityList getEntitiesByType(entityType)**

Retourne la liste des entités de type *entityType* présents sur la carte.

- **boolean pauseScript(scriptID, time)**

Met en pause le script *scriptID* pendant *time* millisecondes" ou indéfiniment si *time* prends la valeur 0. Retourne un booléen *TRUE* en cas de succès ou *FALSE* en cas d'échec.

- **boolean resumeScript(scriptID)**

Relance le script *scriptID* où ce dernier s'est arrêté. Retourne un booléen *TRUE* en cas de succès ou *FALSE* en cas d'échec.

- **boolean stopScript(scriptID)**

Stop le script *scriptID*. Retourne un booléen *TRUE* en cas de succès ou *FALSE* en cas d'échec.

- **boolean reloadScript(scriptID)**

Relance le script *scriptID* depuis le début. Retourne un booléen *TRUE* en cas de succès ou *FALSE* en cas d'échec.

- **boolean loadScript(scriptID, independant)**

Lance le script *scriptID*. Si *indépendant* est *TRUE* lance le script *scriptID* en parallèle du script courant. Si *indépendant* est *FALSE* attend la fin du script *scriptID* avant de continuer le script courant. Retourne un booléen *TRUE* en cas de succès ou *FALSE* en cas d'échec.

NORME DE CODE

La norme de code du projet se base en grande partie sur la norme utilisée à EPITECH.

LIMITES

→ la longueur d'une ligne de code ne doit pas être supérieure à 80 colonnes
→ pas de limite en ce qui concerne la longueur des fonctions, on appréciera un découpage clair et fonctionnel du code

COMMENTAIRES

- tous les commentaires doivent être énoncés clairement en anglais.
- au début de chaque fonction, un paragraphe de commentaire présente son rôle, ses paramètres et sa valeur de retour.
- une zone de commentaires documentée (doxygen) commence par /*! et fini par */ (http://www.doxygen.org/)
- les commentaires dans le corps d'une fonction doivent être succincts et commencer par //
- à l'extérieur d'une fonction/classe on utilisera /* en début et */ en fin de commentaires

NOMMAGE DANS LE CODE

→ Si le vocabulaire est complexe on appréciera un lexique en début de fichier (nom anglais = nom français)

- les noms sont clairs, fonctionnels et en anglais
- les types de variable/enum sont en minuscules
- les noms de variables/fonctions commencent par une minuscule et comportent une majuscule en début de chaque mot: variablePublic
- les noms de variables/fonctions privées commencent par un "_"
- les noms de classes commencent par une majuscule et comportent une majuscule en début de chaque mot
- les macros/enum sont nommées en majuscule, les mots sont séparés par des "_"
- les macros de protection contre la double inclusion sont composées du nom du fichier en majuscule précédé et suivi d'un "_", les "." sont remplaces par un "_" (par défaut CDT nomme la macro attachée au fichier file.h _FILE_H_)

- → les chaines de caractères sont définies en début de fichier

NOMMAGE DES FICHIERS

- les noms des fichiers comportant une classe sont définis selon le nom de la classe qu'ils contiennent (pour .cpp et .hpp)
- les noms des fichiers ne comportant pas de classe sont en minuscule, les mots sont séparés par des "_" (pour .cpp et .h)

LES BLOCS

- dans tous les cas les accolades ouvrantes et fermantes se trouvent toujours alignées à gauche
- → on pourra se passer de blocs superflus dans des cas simple (par exemple : if (true) return true;)

UTILISATION DES MOTS CLEFS

- chaque fonction se termine par un return (même void)
- les mots clefs sont toujours suivis par un espace
- on évitera l'utilisation du mot clef "this"

INDENTATION

- l'indentation est basée sur 2 espaces
- les accolades se situent au même niveau que leur père
- chaque nouveau bloc est indente
- dans le cas de nombreux define successifs on appréciera une utilisation des indentations dans le but de clarifier le code

DECOUPAGE DU CODE

- une seule déclaration de variable par ligne
- les operateurs sont séparés des termes par un espace
- les noms des variables/fonctions/classes sont alignes dans un même fichier
- le modificateur de type * est accole au type
- → le modificateur de type [] est accole au type
- chaque bloc est précédé d'une ligne vide

- → lors de la définition d'une fonction on préférera voir le type de retour sur la même ligne que le nom de la fonction, néanmoins dans le cas de types trop longs et dans le but de respecter la limite des 80 colonnes on acceptera que le type soit décalé sur la ligne supérieure

RECOMMANDATIONS

- chaque classe, même virtuelle, contient un constructeur et un destructeur
- le constructeur d'une classe doit appeler les constructeurs des variables dans l'ordre ou elles ont été déclarées
- chaque fichier se termine par '\n' (gnu compliance)
- l'utilisation du "!" est prohibée pour tout ce qui n'est pas booléen
- pas d'include dans les .h dans la mesure du possible
- l'utilisation du terme "using" est hautement déconseillée, on préféra l'utilisation des typedef
- les .h/.hpp ne contiennent que des déclarations, aucune définition
- les .c/.cpp ne contiennent pas de déclaration de fonctions, ils devront inclure le .h/.hpp contenant cette déclaration
- pour tous les cas que cette norme ne couvrirait pas merci de garder une certaine homogénéité dans le code

FICHES D'ANOMALIES

Nous avons au cours du développement du projet rencontrés un certain nombre d'anomalies de gravitée inégales mais ayant entrainé des changements parfois importants par rapport au plan de développement initial (cf. Cahier des charges)

Voici donc les anomalies ayant entrainées des changements structurels sur Pavillon-noir, justifiant de l'utilisation de certains outils non présents dans le Cahier des charges.

Date : Le 21 Janvier 2005

Description de l'anomalie :

Apres une étude préliminaire, il était prévu d'utiliser la librairie WxWidgets pour lancer une fenêtre de configuration de type logicielle puis toujours avec cette librairie lancer un canevas OpenGL en plein écran.

Lors de tests plus approfondis sur les capacités de cette librairie à gérer un canevas OpenGL il s'est avéré impossible de lancer une fenêtre en plein écran dans la résolution voulue. La fenêtre reste dans la résolution du bureau et s'étire pour occuper tout l'écran.

Qualification de l'anomalie : ~~mineure~~ / ~~majeure~~ / critique

Notre projet est un jeu vidéo il est donc nécessaire que l'on puisse le lancer en plein écran dans la résolution souhaitée.

- ↳ *Implications :*

Il faut donc abandonner l'idée d'utiliser WxWidgets pour faire cette action et chercher une nouvelle façon de la réaliser.

Actions correctives à entreprendre : oui / ~~non~~

- Responsable : Florent Charles
- Date de début : 19/01/05
- Echéance : 21/01/05
- Date de clôture : 21/01/05

Résultats :

La librairie WxWidgets a été définitivement écartée. A sa place la librairie FOX (librairie de Widgets logicielle) beaucoup plus légère sera utilisée pour le lancement du panneau de configuration logiciel puis ce sera la librairie SDL (Simple DirectMedia Layer) qui lancera la fenêtre OpenGL et s'occupera d'intercepter tous les événements d'entre (clavier/souris).

De plus la librairie FOX convient mieux aux besoins liés à la réalisation de l'éditeur.

Date : Le 25 Janvier 2005

Description de l'anomalie :

Dans le gestionnaire d'évènements, la fonction PNEventManager::run() plus précisément, si j'écris sur std::cout à l'intérieur du bloc « if » servant a bloquer le thread lorsque la pile est vide, le programme n'entre plus dans les callbacks.

Si je commente la ligne de code servant à l'écriture, tout est correct.

Qualification de l'anomalie : mineure

Cette anomalie est mineure du fait qu'elle concerne un print de debug inutile mais peut impliquer un bug plus sérieux. Toujours est-il que le gestionnaire fonctionne bien sans ce print de debug.

↳ *Implications :* Il faut repérer la nature exacte du bug, de manière à définir si cela peut porter préjudice au fonctionnement du programme.

Actions correctives à entreprendre : Oui si ce bug s'avère être de nature plus sérieuse.

- Responsable : Pierre Martinez
- Date de début : 25/02/2005
- Echéance : 04/03/2005
- Date de clôture : -

Résultats :

Pour l'instant le bug n'entrave pas le fonctionnement du gestionnaire d'évènements si le print n'est pas effectué.

Date : mercredi 9 mars 2005

Description de l'anomalie :

Le plugin compile mais plante à l'exécution du code au moment d'enregistrer les classes de bind*. L'erreur se produit dans la classe _tree de stlport, et ce même en surchargeant à la place les classes d'exemple de luabind.

Qualification de l'anomalie : **majeur**

Cette anomalie est majeure car ce bug empêche tout simplement l'exécution du projet. D'autant plus que luabind n'est à la base pas compatible avec la version de boost utilisée pour le projet (1.32), Une modification du code source de la librairie a du être effectuée pour pouvoir compiler.

↳ *Implications :*
Rends le plugin inutilisable.

Actions correctives à entreprendre :

Trouver un autre outil de binding lua/c++, comme tolua++

- Responsable : Dupertuys Stéphane
- Date de début : 09/03/2005
- Echéance : 19/03/2005
- Date de clôture :

Résultats :

Il faut recommencer l'étude du fonctionnement d'un nouvel outil de binding depuis le début. Par conséquent cela ralenti le développement de la partie script de 1-2 semaines.

*bind : liaison entre deux langages de programmation différents.

Date : Le 11 Mars 2005

Description de l'anomalie :

Lors de notre étude préliminaire sur les librairies d'interface graphique fonctionnant en OpenGL nous avons choisi Crazy Eddie's GUI (CEGUI). Cette librairie intégré de nombreuses widgets destine au jeu et nous pensions qu'elle possédait un moteur de rendu qui lui était propre ce qui n'est pas le cas.

Qualification de l'anomalie : ~~mineure~~ / ~~majeure~~ / critique

Il est prioritaire que notre projet utilise une librairie de ce type et CEGUI est la seule convenant parfaitement à nos besoins.

- ↳ *Implications :*

Il faut soit trouver un moteur de rendu OpenGL pour CEGUI soit créer le notre pour la librairie.

Actions correctives à entreprendre : oui / ~~non~~

- Responsable : Florent Charles
- Date de début : 01/03/05
- Echéance : 11/03/05
- Date de clôture : 11/03/05

Résultats :

Un moteur de rendu OpenGL a été développé par un membre de la communauté du projet CEGUI. Il a ensuite été intégré au projet dans le plugin de rendu 3D.