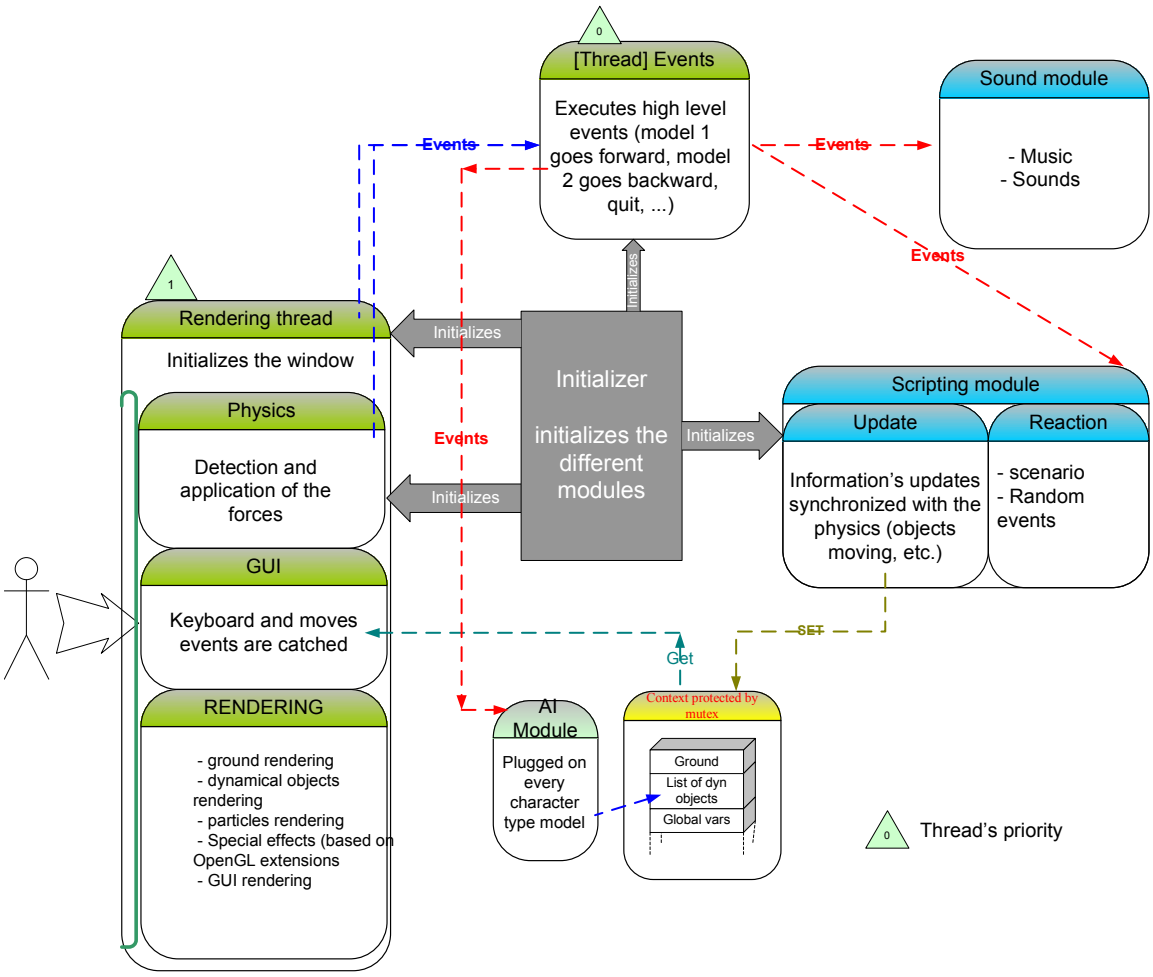
PAVILLON-NOIR : LE JEU

TECHNICAL DOCUMENTATION

INDEX

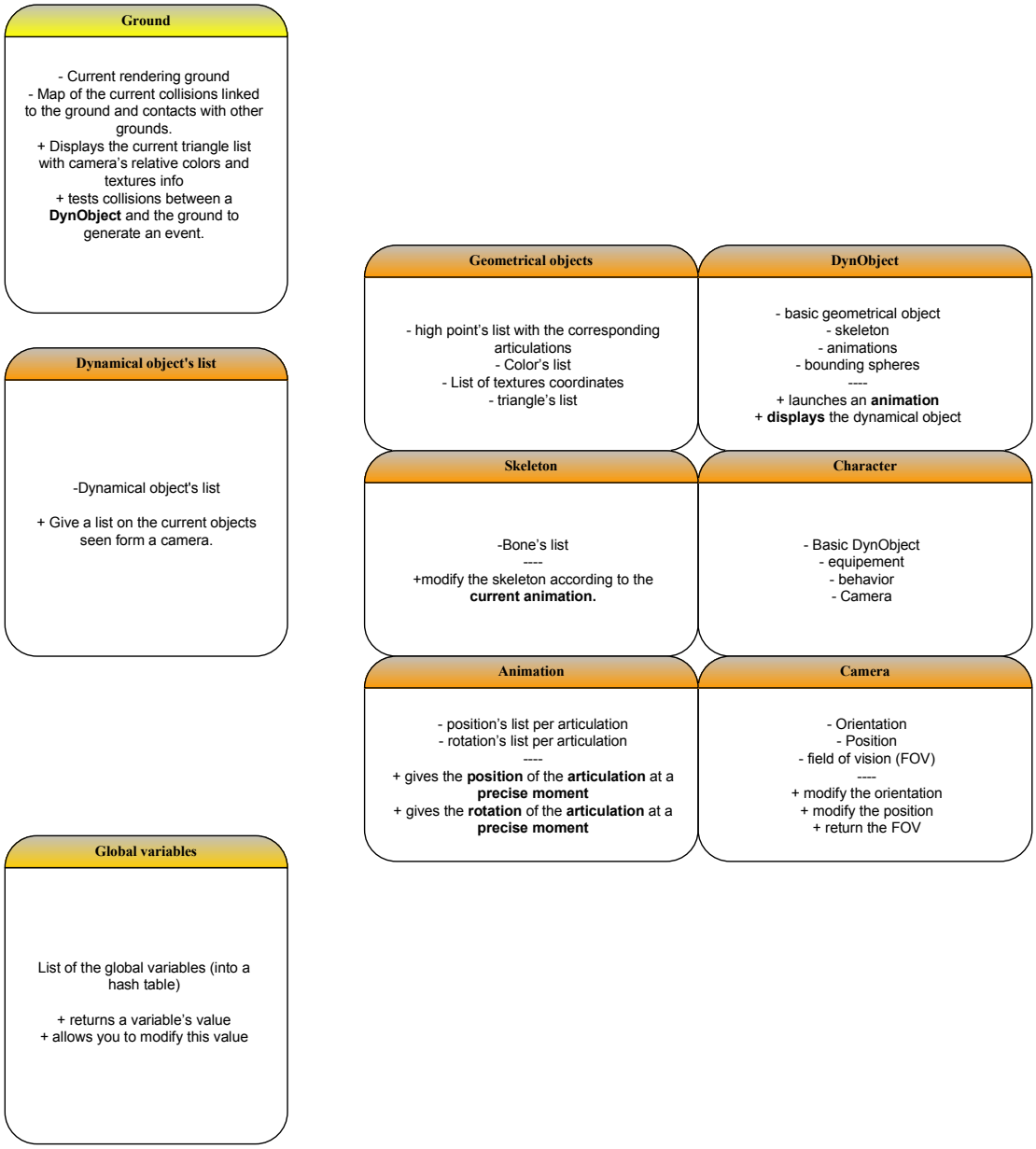
TECHNICAL STRUCTURE

SYSTEM RESSOURCES MANAGER:



The system resources manager initializes all the modules and gives it a simple interface with their environment.

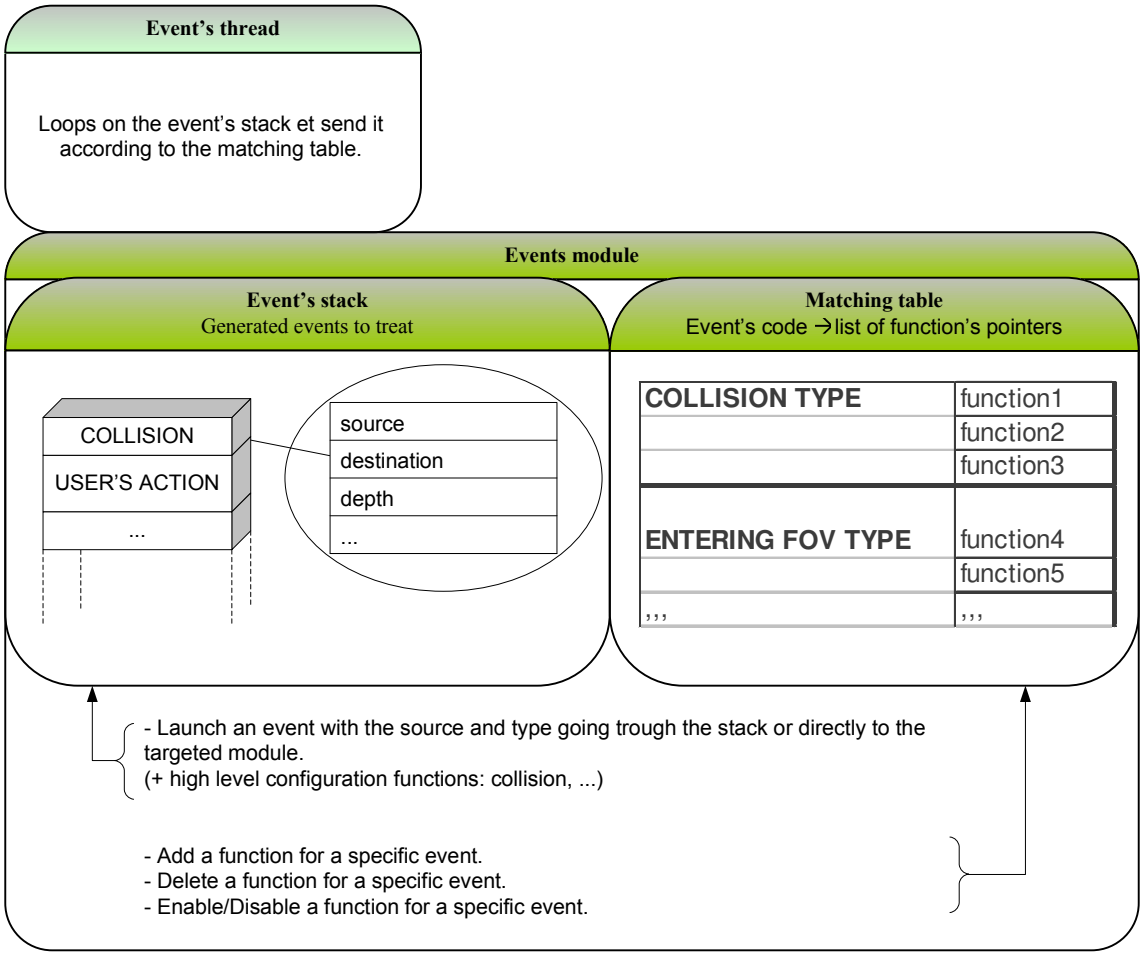
CONTEXT:



The context is a shared memory space where all data's relative to each module are stored (models, animations, textures, global variables, ground…). These objects are automatically protected for being thread safe.

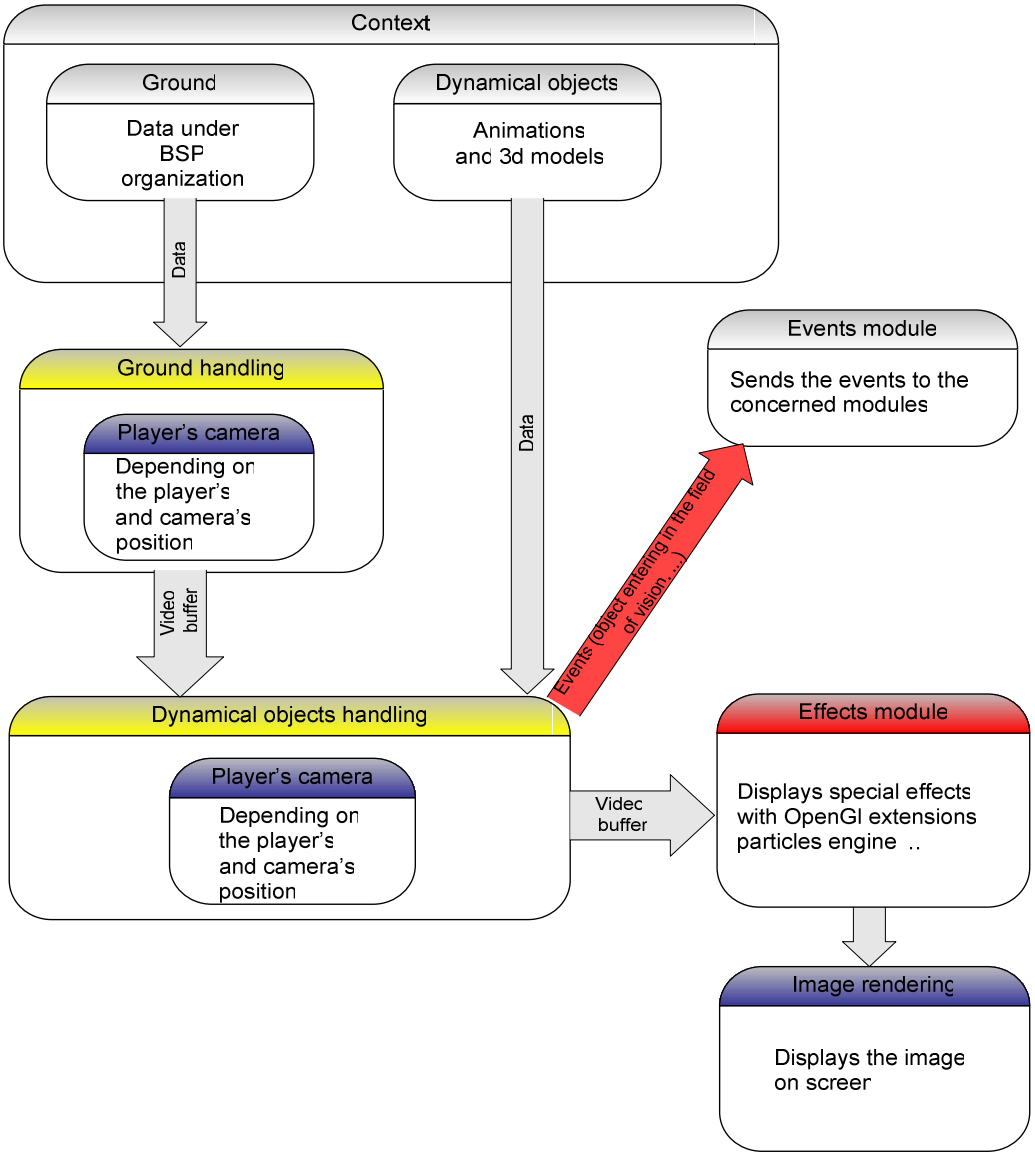
All objects can be exported in a specific format to allow reloading and saving of the game at any time.

EVENTS MODULE:



Event's handling is based on a stack. Every event has a type and is sent to the modules that are linked to it. A matching table helps us making the link between an event's code and the associated functions. You can add or delete some elements of this stack or just enable or disable a specific function for one event in particular.

VISUAL RENDERING MODULE:

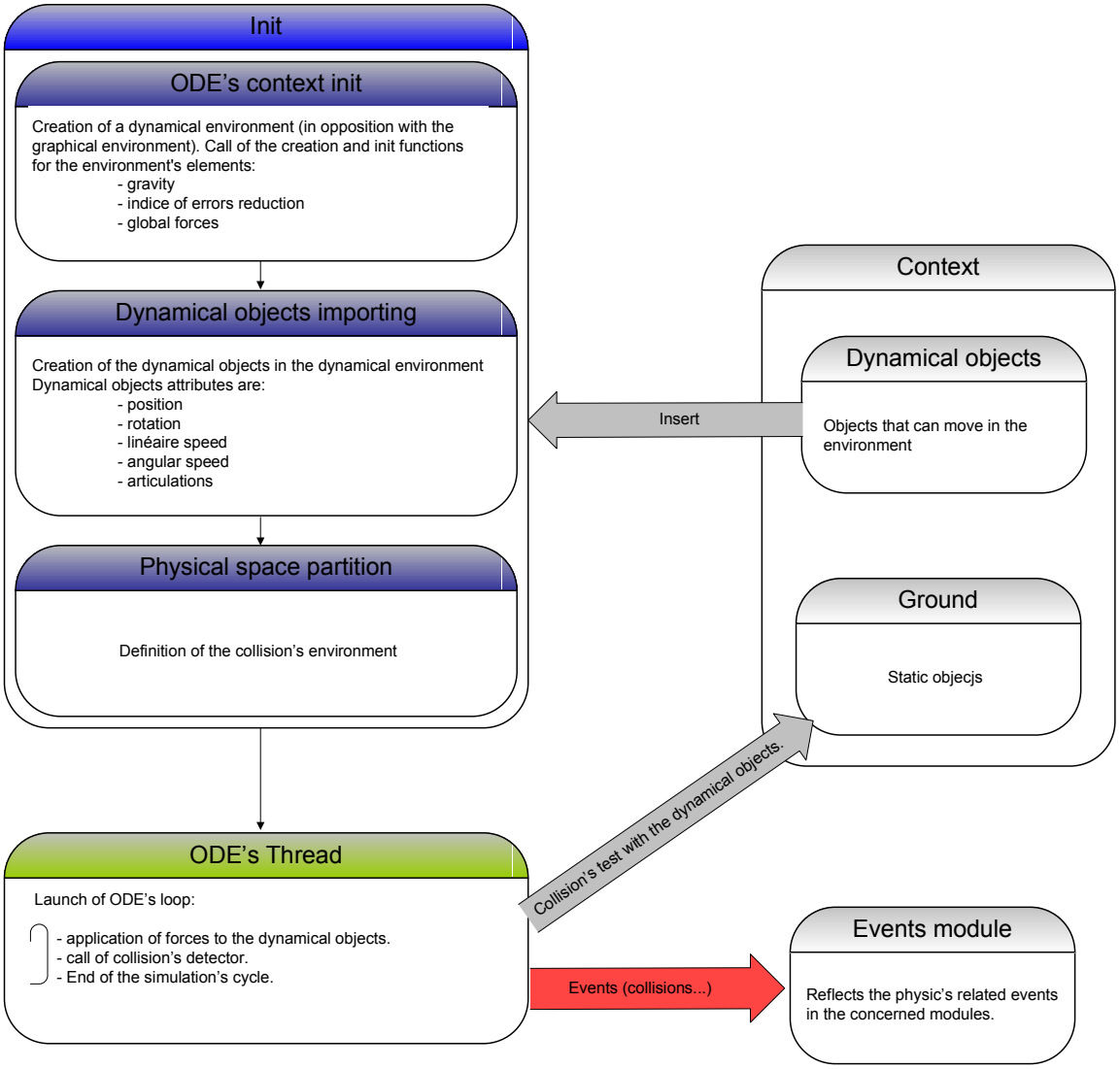


When the rendering module is called, we determine according to the camera's position and orientation witch part of the environment has to be displayed. Then we will determine the characters and dynamical objects that need to be displayed in that area.

At last, the special effects module will add more modification to increase the rendering's realism (with effects on textures for example)

Once all these action are completed, we can display the image in the game et start handling the next image if no other module needs to control the application.

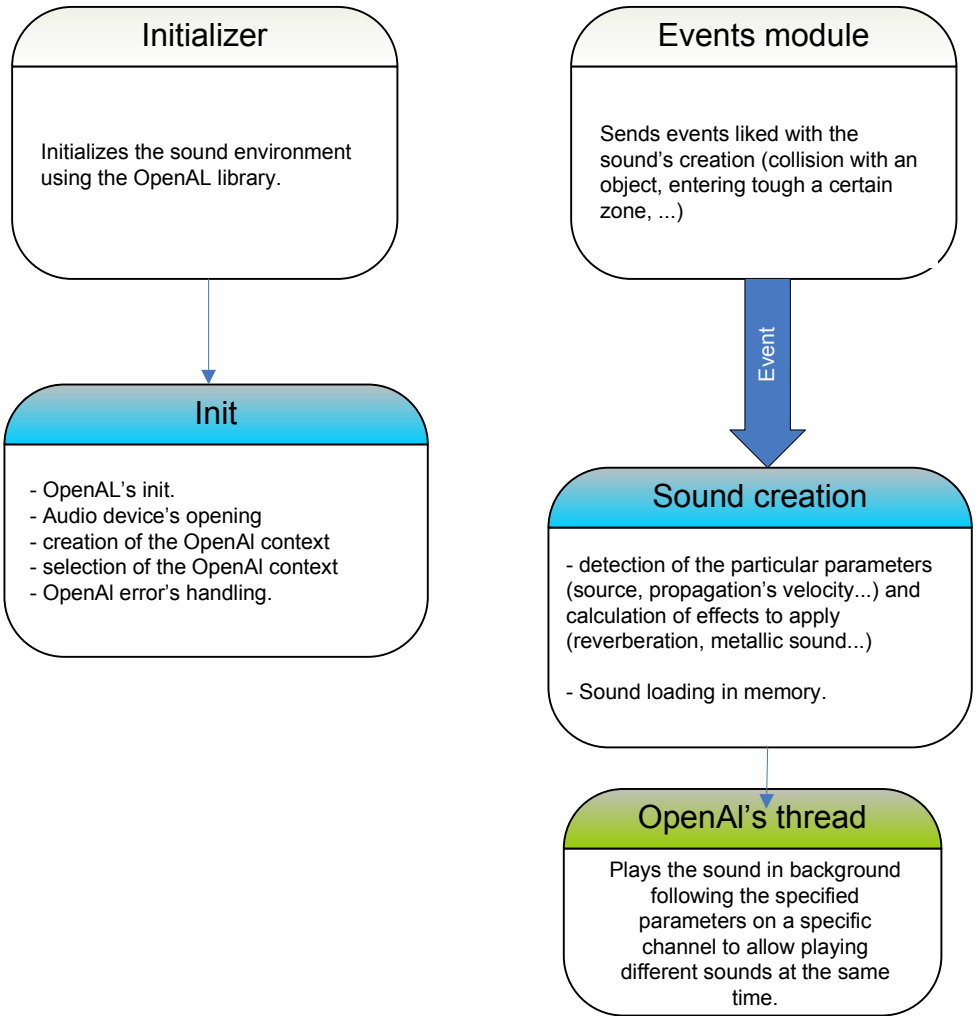
PHYSICS:



The physics engine is based on the OPAL, an interface to ODE (for Open Dynamics Engine), the only engine distributed on the GPL license.

It allows a dynamic representation of virtual environments interfacing with the space partition and rendering modules.

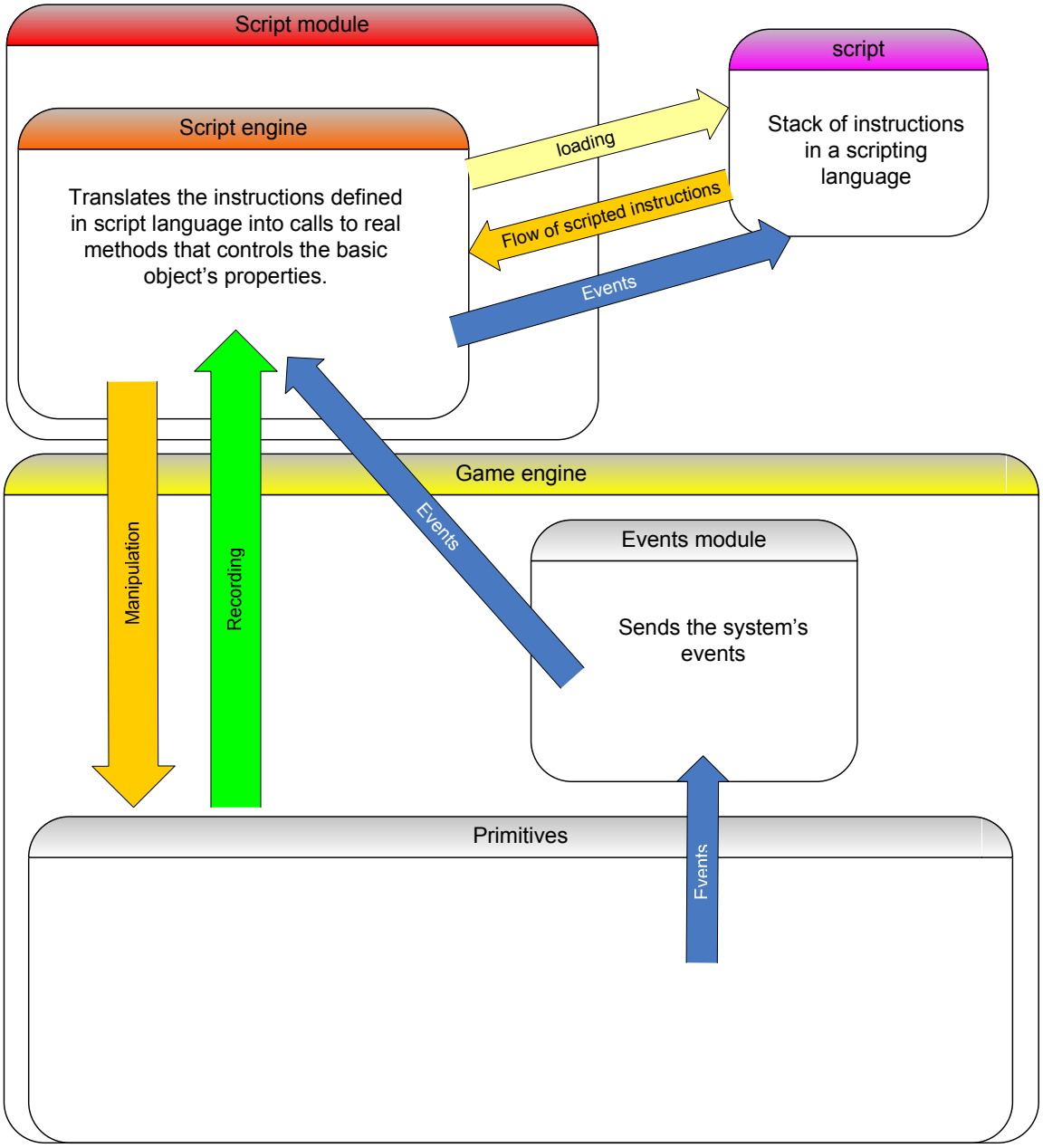
SOUND MODULE:



The sound engine is launched by reception of a sound event. The action is then started via the OpenAl library that adapts the sound to the source and to every environment parameters (outdistances, propagation's velocity…).

The sound is then played on a channel and parameters are then reinitialized.

SCRIPTING MODULE:



The scripting engine interprets scripts in order to manipulate the game's environment. In this way, it is possible to define some planed sequences using the game's engine (A battle for example).

- Scripts need to manipulate the game's entities for: :
    - Creation
    - Suppression
    - Moving
    - Configuration
    - Execution of specific animations.
    - Execution of defined actions.

- Each action needs at least these parameters:
    - An entity's ID.
    - A specific data for the action to execute.

- Each script is executed into a virtual thread depending on the game's engine. In this way, it is possible to execute many scripts at the same time.

- The execution time depends on the objects and on the actions they have to perform. It can change depending on the objects and on the parameters (A man will go slower from point A to B than a horse).

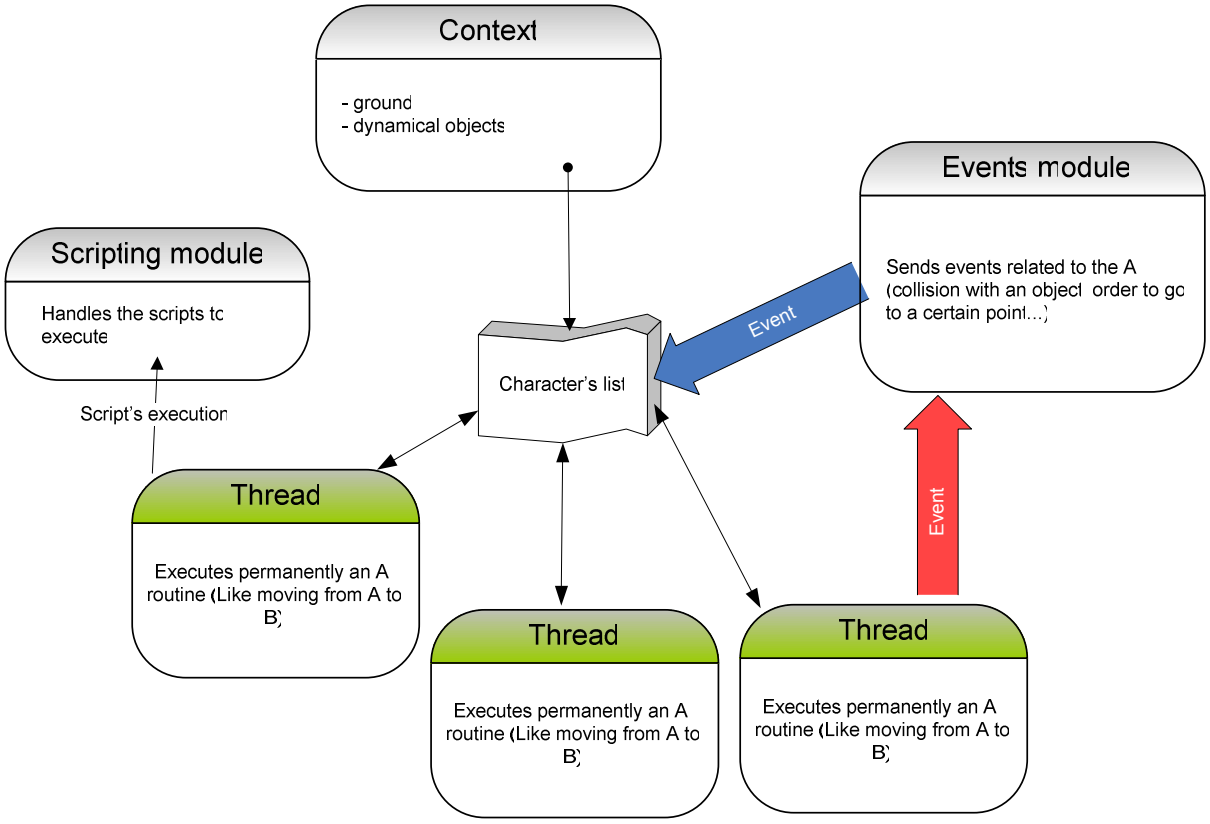
- When an object has ended an action, it has to send an event back to the script engine in order to proceed for the next instruction.

- Every object has it own way of achieving an action (A man will move by walking while a bird will fly).

- A script can be launched, paused or stopped by the scripting engine, by another object or by another script.

- It is also possible to define a time for an action. In this way the next action will be executed after a certain amount of time.

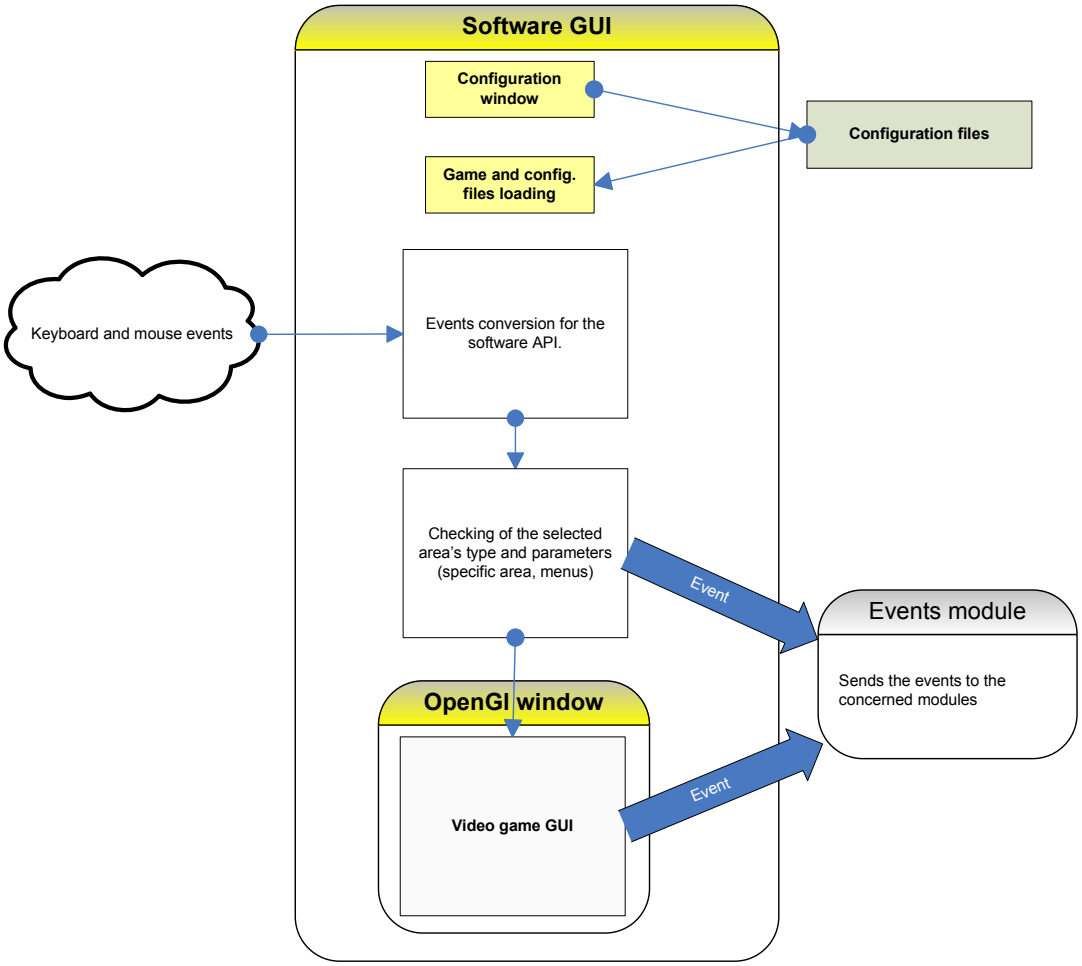
- A script can take control over the player and over the cameras.



A thread is created for each character and will handle instructions like going from point A to point B.

The character reacts to some events depending on the behaviors defined by its AI class. It might add events to the stack or even execute some scripts to execute specific actions.

GRAFICAL USER'S INTERFACE (GUI):



The user's interface is the link between the game and the player allowing him to interact with the environment.

First a software interface will be executed in order to configure the game before it starts (resolution, level of detail, etc.). Then it will launch the game applying these parameters in an OpenGL window.

This window will send all the mouse and keyboard events generated by the player and will translate it into high level actions (going forward, turn, jump. Etc…). The actions that will affect the game's interface are sent to it while the others are directly sent to the events module.

SCRIPTING INTERFACE

IT'S GOAL

The "PnScript" API was developed in order to provide to the Pavillon-noir developers the ability to easily manipulate the game's entities (characters, dynamic elements, cameras …). It also launches to next level at the end of one.

Manipulations on the entities are limited to the action that each of them is able to execute. In other words you can only execute with the script the action that a certain kinf of object can perform.

THE API'S LANGUAGE

This API is developed for the LUA language. For more information on LUA please visit http://www.lua.org.

SPECIFICATIONS

FUNCTIONS

- **scriptID getScriptId()**

Returns the Id of the current script. This Id is used in the script manipulation's functions.

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

Creates a new entity at the specified coordinates in the level. The function returns a unique Id for the created object.

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

Returns the states of the *propertyName* property for the *entityID* entity.

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

Gives value *value* to a *propertyName* proprety for the *entityID entity*.

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

Launches the action *actionName* for the entity *entityID*. This function can take many parameters depending on the action to perform. One the action is done, the function returns Boolean TRUE for success and FALSE for failure.

- **boolean destroy(entityID)**

Destroys an object in the game. The function returns Boolean TRUE for success and FALSE for failure.

entityList getEntitiesByType(entityType)

Returns the list of entities of type *entityType* presents on the current map.

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

Pauses the *scriptID* script for *time* milliseconds or for ever if *time* has a 0 value. The function returns Boolean TRUE for success and FALSE for failure.

boolean resumeScript(scriptID)

Restarts the script *scriptID* where it stopped. The function returns Boolean TRUE for success and FALSE for failure.

boolean stopScript(scriptID)

Stops the script *scriptID*. The function returns Boolean TRUE for success and FALSE for failure.

boolean reloadScript(scriptID)

Reloads the script *scriptID* from the beginning. The function returns Boolean TRUE for success and FALSE for failure.

boolean loadScript(scriptID, independant)

Launches the script *scriptID*. If *independant* is *TRUE* the script *scriptID* will be executed in parallel of the current script. If *independant* is *FALSE* the current script will wait for the script *scriptID* to finish before continuing. The function returns Boolean TRUE for success and FALSE for failure.

CODING STANDARD

The coding standard of the project is mainly based on the EPTECH standard.

LIMITS

→ A line cannot be longer than 80 colons
→ There is no limit for functions length but a clean organization is preferred.

COMMENTS

- All comments must be in English.
- At the beginning of every function, comments present its goal, parameter and return value.
- A documented comment zone (doxygen) starts with /*! and ends with */ (http://www.doxygen.org/)
- Comments inside a functions body must be short and has to start with //
- Outside a function or classes /* and */ will be used for comments

NAMING

→ If the vocabulary is complex, a lexical will be appreciated at the beginning of the file.

- Names must be clear, functional and in English
- Variables and enums are in lower case
- Names of variables and functions start in lower case and has a capital letter at the beginning of each word (like publicVariable)
- Private functions and variables names start with "_"
- Classes names starts with a capital letter and have one at the beginning of each new word.
- Macros and enums are named in capital letters, words a separated by "_"
- Macros protecting from multi-inclusion are composed of the file name in capital letters separated with "_". The name starts with a "_" and other character like "." Are replaced with "_" (like _FILE_H_ for file.h)
- → Strings are defined at the beginning of the file.

FILES NAMING

- Files containing a class are defined by the name of this class (for .cpp and .hpp)
- Files containing no classes are in lower case, words are separated with "_" (for .cpp and .h)

BLOCKS

- "{" are always aligned to the left.
- You can create no blocks for simple applications (like if (true) return true;)

KEYWORKS USAGE

- Every function ends with a return (even void)
- Keywords are always followed by " "
- Usage of keyword "this" is depreciated.

INDENTATION

- Indentation is based on 2 spaces
- "{" and "}" has to be on the same indentation level.
- Each new block is indentured.
- In case of multi define usage, indentation can be used to make to code more easy to read.

CODE'S ORGANISATION

- One variable declaration per line
- Operators must be separated with one space
- Variable, functions and classes names are aligned in the same file.
- The * character is attached to the type
- The [] characters are attached to the type
- There must be an empty line before each block
- For a function's definition the return type has to be one the same line that the function's name. This rules is not valid if the function's definition is larger than 80 colons

RECOMMANDATIONS

- Each class (even virtual classes) has a constructor and a destructor.
- The classes constructor must call the variables constructors in the same order as they are declared
- Every file ends with a '\n' (gnu compliance)
- Usage of "!" is forbidden except for the Booleans
- No includes in .h if possible
- Keyword "using" is depreciated, prefer using typedefs
- .h and .hpp only contains declarations, no definitions
- .c and .cpp do not contain declarations, it must include the corresponding .h or .hpp
- For all case not defined in this standard please try to keep a certain homogeneity for the code.

ANOMALIES DATA SHEETS

During the development of the project we went trough different kind of anomalies with more or less impact.

Some of these anomalies had an impact on the initial development plan and made us change part of the conception.

Here are the anomalies that created structural changes on Pavillon-noir, changing in particular some tools used in the project.

Date: January 21 2005

Description of the anomaly:

At first, the library chosen to launch the configuration window was WxWidget.

After deeper tests, this library seems to have troubles handling an OpenGL canvas and it was impossible to launch this window with the desired resolution. The window stays at the same resolution that the desktop and is stretched to appear fullscreen.

Qualification of the anomaly: ~~minor~~ / ~~major~~ / critic

We need to launch the game's windows in the desired resolution.

- ↳ *Implications:*

We need to find another library to replace WxWidgets.

Actions to undertake: yes / ~~no~~

- Responsible : Florent Charles
- Beginning : 19/01/05
- Programmed end : 21/01/05
- Final end : 21/01/05

Results:

WxWidgets has been replaced by FOX (another software widget library). It is lighter and will be used for launching the configuration menu, the SDL library (Simple DirectMedia Layer) will launch our OpenGL window and will catch all keyboards and mouse events.

FOX will be used for the level editor as well.

Date : January 25 2005

Description of the anomaly:

In the event manager, in the function PNEventManager::run(), when I write a line with std::cout inside the « if » block that blocks the thread when the stack is empty, the program does not respond to the callbacks.

Commenting this line makes everything ok.

Qualification of the anomaly: minor/ ~~major~~ / ~~critic~~

This anomaly is minor but can imply a deeper bug. Without this debug print, the engine works fine.

↳ ***Implications:*** we have to find the origin of the bug to have a more precise idea of the problem.

Actions to undertake: Yes if the bug occurs to be more serious.

- Responsible : Pierre Martinez
- Beginning : 25/02/2005
- Programmed end : 04/03/2005
- Final end : -

Results :

For the moment the bug does not affect the module if the print is disabled.

Date : mars 9 2005

Description of the anomaly:

The program compiles but crashes when the code is executed. The error is located in the class _tree of stlport, even overloading the function without the basic luabind classes.

Qualification of the anomaly:: **major**

This anomaly is major because this bug blocks the entire execution of the program. This might be cause by the fact that luabind is not basically compatible with our version of BOOST. (1.32). In fact this library had to be modified in order to work with the project.

↳ *Implications:*
The plugging is not functional

Actions to undertake:

We have to find another tool for binding lua and c++, like tolua++

- Responsible : Dupertuys Stéphane
- Beginning : 09/03/2005
- Programmed end : 19/03/2005
- Final end : -

Results :

We have to study a new tool for binding. It will slow the development of the script engine by 1 or 2 weeks.

*binding : link between two different programming languages.

Date : Mars 11 2005

Description of the anomaly:

After studying the different GUI libraries working with OpenGL we chose Crazy Eddie's GUI (CEGUI). This library contains many widgets for video games and we thought that it contained its own rendering interface. We were wrong.

Qualification of the anomaly: ~~minor~~ / ~~major~~ / critic

We need this kind of library and CEGUI is the only one that fits on our needs.

↳ *Implications:*

We need to find an OpenGL rendering engine for CEGUI or to build our own engine.

Actions to undertake: yes / ~~no~~

- Responsible : Florent Charles
- Beginning : 01/03/05
- Programmed end : 11/03/05
- Final end : 11/03/05

Results :

An OpenGl rendering engine was developed by a member of the community around the CEGUI project. It has been integrated in our 3D rendering module.