HPMS Engine

# Concept

Partendo dal prototipo sviluppato tramite LWJGL (JRD3) verrà sviluppato un motore analogo più leggero e ottimizzato in codice nativo (C++)

# Filesystem

Per avere compressione, trasparenza e velocità di caricamento delle risorse, verranno utilizzati dei formati binari e un filesystem virtuale.

Utilizzando la libreria *PhysFS*, tutte le risorse presenti in un archivio (formato *7z*, in questo caso rinominato *hpak*) vengono caricate in fase di startup e montate su una directory virtuale strutturata nel modo seguente:

data  
|  
+--- Data.cfg  
|  
+--- scripts  
|  
+--- rooms  
|  
+--- shaders  
|  
+--- models  
|  
+--- textures  
|  
+--- screens  
|  
+--- sounds

Gli scripts, shaders, immagini e audio rimarranno in formato standard, mentre i modelli 3D e i descrittori delle ambientazioni verranno compressi in un formato binario.

# Engine

# Runtime

# Editor

L’editor verrà sviluppato in Java + JavaFX per velocizzare i tempi di realizzazione della GUI. L’applicativo permette di:

* Caricare la walkmap e le camere da un file esportato da blender;
* Caricare le immagini degli sfondi e dei livelli di parallasse;
* Poter visualizzare un’anteprima delle varie inquadrature con il proprio sfondo e livelli intermedi;
* Poter assegnare ai vari settori la view che setteranno quando verranno calpestati del giocatore;
* Poter assegnare ai vari settori la profondità che assegneranno ai vari oggetti 3D quando questi li calpesteranno (questo andrà definito per ogni vista presente, in quanto la profondità dipende da essa);
* Poter aggiuingere gli oggetti 3D della scena;
* Poter aggiungere comportamenti ai vari oggetti o alla scena stessa, tramite script LUA, usando un sistema di condizione/azione;
* Poter importare i vari modelli 3D e visualizzarne l’anteprima;
* Poter esportare dal progetto un HPAK contenente:
  + Un XML/JSON che descrive la scena (incluse camere, settori, scripts ecc…);
  + Le immagini ed eventuale audio;
  + I modelli 3D (che verranno prima convertiti nel formato binario usando una routine esterne scritta in C++, che importerà i modelli nelle strutture usate dal runtime e le serializzerà in relativi file binari).
* Poter salvare tutto il lavoro in formato *.hpmsproj*