HPMS VM

# Concept

Tutta la componente di gameplay viene gestita tramite script LUA.

# Funzioni

In seguito le funzioni messe a disposizione dalla Virtual Machine per interfacciarsi al motore.

## Gestione Assets

e = hpms.make\_entity('data/out/01.hdat') -- Creates new entity  
hpms.delete\_entity(e) -- Safe deletes and entity  
  
b = hpms.make\_background('data/resources/textures/B01\_B.png') -- Creates new background image  
hpms.delete\_background(b) -- Safe deletes a background image  
  
p = hpms.make\_picture('data/resources/textures/B01\_P.png') -- Creates new foreground image  
hpms.delete\_picture(p) -- Safe deletes a foreground image  
  
d = hpms.make\_depth\_mask('data/resources/textures/B01\_D.png') -- Creates new depth mask image  
hpms.delete\_depth\_mask(d) -- Safe deletes a depth mask image  
  
hpms.add\_entity\_to\_scene(e, scene) -- Adds an entity to world  
hpms.add\_picture\_to\_scene(b, scene) -- Adds a picture to world

## Gestione Entità

e.position = hpms.vec3(0, 0, 0) -- Get and set entity position  
e.rotation = hpms.from\_axis(hpms.to\_radians(90), 0, 0, 1) -- Get and set entity rotation  
e.scale = hpms.vec3(0, 0, 0) -- Get and set entity scale  
e.anim\_index = 0 -- Get and set current entity animation  
e.anim\_frame\_index = 10 -- Get and set frame for current entity animation  
e.visible = *true* -- Get and set entity visibility flag  
e.anim\_loop = *false* -- Get and set entity loop animation flag  
e.anim\_play = *true* -- Get and set entity play animation flag  
  
p.alpha = 1 -- Get and set picture alpha  
p.x = 100 -- Get and set picture x  
p.y = 50 -- Get and set picture y  
p.visible = *true* -- Get and set picture visibility flag

## Funzioni Matematiche

r = hpms.to\_radians(90) -- Convert degrees to radians  
d = hpms.to\_degrees(1) -- Convert radians to degrees  
  
q = hpms.quat() -- Creates empty quaternion  
q = hpms.quat(1, 0, 0, 0) -- Creates quaternion with w, x, y, z  
q = hpms.from\_axis(hpms.to\_radians(90), 0, 0, 1) -- Creates quaternion from euler angles  
q = hpms.quat\_mult(q\_a, q\_b) -- Creates quaternion as multiplication of two quaternions  
q.w = 1 -- Get and set w property  
q.x = 0 -- Get and set x property  
q.y = 0 -- Get and set y property  
q.z = 0 -- Get and set z property  
  
v2 = hpms.vec2() -- Creates 2 dimensions empty vector  
v2 = hpms.vec2(0, 0) -- Creates 2 dimensions vector with x, y  
v2 = hpms.vec2\_add(v2\_a, v2\_b) -- Creates 2 dimensions vector as sum of two 2 dimensions vectors  
v2 = hpms.vec2\_sub(v2\_a, v2\_b) -- Creates 2 dimensions vector as subtraction of two 2 dimensions vectors  
v2 = hpms.vec2\_mult(v2\_a, v2\_b) -- Creates 2 dimensions vector as multiplication of two 2 dimensions vectors  
v2 = hpms.vec2\_div(v2\_a, v2\_b) -- Creates 2 dimensions vector as division of two 2 dimensions vectors  
dist = hpms.vec2\_dist(v2\_a, v2\_b) -- Calculates distance between two 2 dimensions vectors  
dot = hpms.vec2\_dot(v2\_a, v2\_b) -- Calculates scalar product between two 2 dimensions vectors  
v2.x = 0 -- Get and set x property  
v2.y = 0 -- Get and set y property  
  
v3 = hpms.vec3() -- Creates 3 dimensions empty vector  
v3 = hpms.vec3(0, 0, 0) -- Creates 3 dimensions vector with x, y, z  
v3 = hpms.vec3\_add(v3\_a, v3\_b) -- Creates 3 dimensions vector as sum of two 3 dimensions vectors  
v3 = hpms.vec3\_sub(v3\_a, v3\_b) -- Creates 3 dimensions vector as subtraction of two 3 dimensions vectors  
v3 = hpms.vec3\_mult(v3\_a, v3\_b) -- Creates 3 dimensions vector as multiplication of two 3 dimensions vectors  
v3 = hpms.vec3\_div(v3\_a, v3\_b) -- Creates 3 dimensions vector as division of two 3 dimensions vectors  
dist = hpms.vec3\_dist(v3\_a, v3\_b) -- Calculates distance between two 3 dimensions vectors  
dot = hpms.vec3\_dot(v3\_a, v3\_b) -- Calculates scalar product between two 3 dimensions vectors  
cross = hpms.vec3\_cross(v3\_a, v3\_b) -- Calculates cross product between two 3 dimensions vectors  
v3.x = 0 -- Get and set x property  
v3.y = 0 -- Get and set y property  
v3.z = 0 -- Get and set z property  
  
v4 = hpms.vec4() -- Creates 4 dimensions empty vector  
v4 = hpms.vec4(0, 0, 0, 0) -- Creates 4 dimensions vector with x, y, z, w  
v4 = hpms.vec4\_add(v4\_a, v4\_b) -- Creates 4 dimensions vector as sum of two 4 dimensions vectors  
v4 = hpms.vec4\_sub(v4\_a, v4\_b) -- Creates 4 dimensions vector as subtraction of two 4 dimensions vectors  
v4 = hpms.vec4\_mult(v4\_a, v4\_b) -- Creates 4 dimensions vector as multiplication of two 4 dimensions vectors  
v4 = hpms.vec4\_div(v4\_a, v4\_b) -- Creates 4 dimensions vector as division of two 4 dimensions vectors  
dot = hpms.vec4\_dot(v4\_a, v4\_b) -- Calculates scalar product between two 4 dimensions vectors  
v4.x = 0 -- Get and set x property  
v4.y = 0 -- Get and set y property  
v4.z = 0 -- Get and set z property  
v4.w = 0 -- Get and set w property  
  
m4 = hpms.mat4() -- Creates 4x4 matrix  
m4 = hpms.mat4\_add(m4\_a, m4\_b) -- Creates 4x4 matrix as sum of two 4x4 matrices  
m4 = hpms.mat4\_sub(m4\_a, m4\_b) -- Creates 4x4 matrix as subtraction of two 4x4 matrices  
m4 = hpms.mat4\_mult(m4\_a, m4\_b) -- Creates 4x4 matrix as multiplication of two 4x4 matrices  
m4 = hpms.mat4\_div(m4\_a, m4\_b) -- Creates 4x4 matrix as division of two 4x4 matrices  
el = hpms.mat4\_elem\_at(m4, 0, 0) -- Returns matrix element at given index

## Gestione Input

k = key\_event.key -- Returns the current triggered key  
t = key\_event.input\_type -- Returns the type of input for triggered key

## Gestione Scena

scene.ambient\_light = hpms.vec3(1, 1, 1) -- Get and set ambient light for current scene  
  
camera.position = hpms.vec3(1, 2, 0) -- Get and set camera position for current scene  
camera.rotation = hpms.vec3(hpms.to\_radians(90), 0, 0) -- Get and set camera rotation for current scene