Framework fr docs

Contributed by Administrator
Last Updated Thursday, 16 November 2006

IV. Les terrains

IV.1 Manipuler un terrain (un champs d'altitudes)

IV.2 Manipuler un terrain de collision (un champs d'altitude)

V. Divers

V.1 Gérer une caméra contenue dans un fichier modèle

V.2 Utiliser un noeud contenue dans un fichier modèle

V.3 Créer un ligne directrice interpolée pour gérer un mouvement scénarisé

V.4 Déclarer un groupe de paramètres xml

V.5 Ajouter un chemin de recherche pour une 'font' ou une image

V.6 Ajouter un membre de classe dans un plugin

I. Les éléments graphiques

Les éléments graphiques seront ajoutés à la scène courante lors du chargement. Cela va créer une instance de la classe concernée ('cooki3d::File', 'cooki3d::Texte', 'cooki3d::Charater'...). Cette instance peut être directement accessible et manipulable par son nom si un plugin est associé au level déclaré. On pourra ainsi faire des manipulations dynamiques lors des événements ou encore implémenter des manipulations qui ne sont pas possibles depuis le framework.

TIPS:

> pour voir directement la définition de la classe dans votre éditeur, il suffit de cliquer sur l'entête du bloc ('file', 'texte', 'characterSeqAnim'...).

NEXT VERSION:

> la manipulation des instances directement via un langage de script choisi (tcl, lua)

I.1. Lire un fichier du format ive, osg, 3ds...

ACTION dans le framework : sur le noeud <jeu> faire 'new Level'

sur le noeud <level> faire 'new File'

CLASSE : 'cooki3d::File'

Cette classe permet de manipuler un fichier 3d. On pourra modifier son placement dans l'espace, sa taille, la facon dont les chemins sont renommée de manière uniforme...

Techniquement la noeud chargé devient fils d'un noeud osg::PositionAttitudeTranform (pour le placement et la taille) nommé par le 'nom du level'->'nom de l'instance'

ATTRIBUTS xml/c++ :

> le nom de l'instance ('nom')

> le chemin du fichier à charger ('item')

> la position ('x', 'y', 'altitude')

> la taille ('scale')

> booléen déterminant si les chemins doivent être renommés de manière uniforme et unique ('pathUnicity')

NOTE :

Par defaut on est limité au format les plus courants pour reduire la taille des packages déployés (cf liste des plugins fournie) pour les autres formats, il faudra fournir les librairies 'dll' et 'so' avec la demo.

NEXT VERSION:

La prochaine version (>=0.22) supportera :

> le téléchargement de packages additionnels qui pourront être utilisés par plusieurs démos. (comme par exemple les formats adittionnels non fournis par défaut ou encore le support de QT4...)

I.2. Afficher du texte ACTION dans le framework : sur le noeud <jeu> faire 'new Level'

sur le noeud <level> faire 'new Texte'

CLASSE : 'cooki3d::Texte'

Cette classe permet de manipuler un objet osg::Text. On peut ainsi choisir la couleure, la taille, la "font" police de

caractère, la position et le texte à afficher.

ATTRIBUTS xml/c++ :

> le nom de l'instance ('nom')

> les composantes de la couleur ('colorR','colorG','colorB','colorA')

> la taille ('charaterSize')

> la font ('font')

> la position ('posX', 'posY', 'posZ')

> le texte à afficher. ('texte')

TIPS:

> il faut souvent ajouter un chemin pour que la font soit trouvée

Action dans le framework : sur le noeud <texte> ou <level> faire 'add filePath'

NOTE:

Seule les textes projetés en 2D sont supportés.

NEXT VERSION:

La prochaine version (>=0.22) supportera :

> une option pour autoriser le texte à ne pas être projecté en orthogonal

I.3. Créer un personnage (ou une entité) composé d'animations et de séquences

ACTION dans le framework : sur le noeud <jeu> faire 'new Level'

sur le noeud <level> faire 'new CharacterSeqAnim'

CLASSE : 'cooki3d::CharacterSeqAnim' (super classe 'cooki3d::Character')

Cette classe permet de gérer un noeud personnage (ou une entité ayant des comportements). Ensuite il faut associer des animations et/ou des séquences au personnage. Celles-ci appartiennent généralement à des fichiers, il faut donc en déclarer au moins un et y faire référence.

ATTRIBUTS xml/c++ :

> le nom de l'instance ('nom')

> un facteur d'attraction terrestre par rapport aux terrains envionnants ('attraction').

> un position initialse X,Y,Z ('positionX','positionY','elevationZ')

> un facteur en seconde de transition en ses animations ('blendTime')

> une vitesse de déplacement lors des translations ('vitesseDeplacement')

> une vitesse de rotation lors de rotations ('vitesseRotation')

> un facteur de taille homogène ('scale')

> le noeud de référence qui sera manipulé lors des déplacement... ('osgNodeName')

TIPS:

> partir d'un exemple donné et avancer pas à pas pour réussir à faire votre propre personnage

NOTE:

Pour utiliser la fonction allerA(...) qui permet de se déplacer sur un

terrain, il faut déclarer une animation ou une sequence s'appellant

{nom du perso}_walk

NEXT VERSION:

La prochaine version (>=0.22) permettrera de :

> choisir les suffixes des animations courantes (actuellement en dur '_walk' pour le déplacement et '_idle' pour faire du surplace)

I.4. Créer un personnage basé sur l'api cal3d

CLASSE : 'cooki3d::CharacterOsgCal' (super classe 'cooki3d::Character')

Il s'agit d'utiliser un personnage basé sur des bones et un squelette.

Cette fonctionnalité est désactivée. Elle nécessite le téléchargement

de packages additionnels (api cal3d...) qui pourront être utilisés

par plusieurs demos.

La manipulation de ce personnage est strictement similaire à

un CharacterSeqAnim.

TIPS:

NEXT VERSION:

La prochaine version (>=0.22) supportera :

> Activation de ce type/classe de personnage

> permettrera de choisir les suffixes des animations courantes (actuellement en dur '_walk' pour le déplacement et '_idle' pour le surplace)

II. Les événements

Un événement sera associé à un code qui recevra en paramètre une référence sur l'événement. Il nécéssite donc la création d'un bloc 'library' qui sera lié au level concerné. Le patron du code sera généré automatiquement si l'option 'autogen' reste cochée. Ainsi l'utilisateur a juste à ajouter le code spécifique à executer lors du déclenchement de l'événement.

SUPER CLASSE : 'cooki3d::Event'

NOTE:

> Il faut veillé à bien modifier uniquement les parties entre les balises prévus à cet effet, sinon les modifications seront perdues lors de la prochaine génération.

II.1. Déclarer un événement clavier ACTION dans le framework : sur le noeud <jeu> faire 'new Level'

sur le noeud <level> faire 'new EvtKey'

CLASSE: 'cooki3d::EvtKey'

ATTRIBUTS xml/c++ :

> le nom de l'instance ('nom')

> le type d'événement : touche poussée ou relevée ... ('evtType')

> le symbole clavier de la touche ('keySymbol')

> les touches modes complémentaires : alt, ctrl... ('modKeyMask')

> booléen déterminant si le code est auto-généré ('autogen')

> le code associé à l'événement ('code')

NOTE:

Il est possible de centraliser le code associé à plusieurs touches

en utilisant un bloc 'EvtKeyGroup' ayant alors des fils 'EvtKey'

TIPS:

II.2. Déclarer un événement souris

ACTION dans le framework : sur le noeud <jeu> faire 'new Level'

sur le noeud <level> faire 'new EvtMouse'

CLASSE: 'cooki3d::EvtMouse'

ATTRIBUTS xml/c++ :

> le nom de l'instance ('nom')

> le masque de sélection utilisé lors du lancé de rayon ('nodeMask')

> le type d'événement : click de souris poussé, relevé, mouvement, glissé ('type')

> booleen déterminant si le code est auto généré ('autogen')

> le code associé à l'événement ('code')

> les zones d'activation : fils de type 'osgNode'

Il faut définir des noms de noeuds graphiques fils à cet événement souris. En faite il s'agit de zones 3D d'activation de l'événement. Ces zones sont définies par l'utilisateur par des chemins (nodePath) dans la scène menants à des noeuds décrivant des géométries (osg::Drawable). Ces chemins peuvent utiliser des carctères spéciaux :

> Le caratère '*' représente n'importe quelle zone

> '/level0->ParkColoriage/Scene Root/palette*' signifie tout ce qui commence

par palette et qui a pour parent 'Scene Root' ayant lui même pour parent

'level0->ParkColoriage'

> '@' représente la zone vide

TIPS:

> pour grouper plusieurs événements souris sur un même code, il n'y a pas de classe 'EvtMouseGroup', néanmoins vous pouvez décocher l'option 'autogen' et utiliser un code déjà existant.

II.3. Déclarer un événement de collision

ACTION dans le framework :

sur le noeud <jeu> faire 'new Level'

sur le noeud <level> faire 'new EvtBoundingShapeCollision'

CLASSE : 'cooki3d::EvtBoundingShapeCollision'

Un événement de collision est relié à deux 'BoundingShape' qui en principe peuvent entrer en collision. Il est donc nécéssaire de déclarer 2 blocs 'BoundingShape' (qui peuvent être partagés avec d'autre événements de collision).

L'événement est mise à jour selon un nombre de frame fixé par l'utilisateur ('refreshPeriod') le minimum étant 1 (vérification à chaque frame).

Lorsque l'événement est détecté, le code associé est executé, et ensuite un délais d'attente fixé par l'utilisateur ('standyDelay') démarre avant la reprise de la détection.

ATTRIBUTS xml/c++ :

> le nom de l'instance ('nom')

> période de rafraichissement de l'événement en nombre de frames ('refreshPeriod')

> un délais d'attente avant reprise de la détection ('standbyDelay')

> première 'BoundingShape', boite englobant l'objet 1 ('collision1')

> deuxième 'BoundingShape' ('collision2')

> booleen déterminant si le code est auto généré ('autogen')

> le code associé à l'événement ('code')

TIPS:

> pour grouper plusieurs événements de collision sur un même code, il n'y a pas de classe 'EvtCollisionGroup', néanmoins vous pouvez décocher l'option 'autogen' et utiliser un code déjà existant.

III. Les médias

Un média peu s'apparenter à un lecteur mp3 avec un bouton pause, play, stop, avancer, reculer...

On a donc principalement les méthodes pause(), play(), stop(), setPostion(nbSecondes) pour avancer ou reculer et setDuration(nbSecondes) pour redimensionner le flux si cela est possible ('cooki3d::Animation')

SUPER CLASSE : 'cooki3d::MediaEvaluable'

INTERFACE : 'cooki3d::Media'

III.1 Manipuler une animation interpolée contenu dans un fichier modèle

ACTION dans le framework : sur le noeud <jeu> faire 'new Level'

sur le noeud <level> faire 'new Animation'

CLASSE: 'cooki3d::Animation'

Un objet de cette classe permet de manipuler comme un média une instance de 'osg::AnimationPath' contenu dans la scène. Cette classe 'osg' est un outil qui permet d'interpoler la position et la rotation instantannée d'un point en fonction d'une liste de points de contrôle et du temps.

ATTRIBUTS xml/c++ :

> le nom de l'instance ('nom')

> booléen déterminant si l'instance doit prendre les valeurs du modèle automatiquement ou si il faut découper l'animation du modèle selon les trois paramètres 'debut', 'fin' et 'fps' de manière manuelle.

> (en mode manuel) numéro de la première "image" (clef) de l'animation ('debut')

> (en mode manuel) numéro de la dernière "image" (clef) de l'animation ('fin')

> (en mode manuel) nombre d'images par secondes ('fps')

> type de mouvement souhaité de l'animation ('mouv')

> chemin du noeud osg contenant l'animation [osg::AnimationPath] ('nodeName')

TIPS:

> Il est possible de manipuler directement un groupe d'animations concernant plusieurs noeuds de facon centralisée et synchronisée. Il faut pour cela utiliser le bloc 'animationGroup'. Une animation groupée hérite des mêmes attibuts xml qu'une animation classique. A ceci près qu'elle peut avoir plusieurs noeuds d'animation. On pourra aussi utiliser la syntaxe 'debut_chemin_scene*' pour pouvoir décrire un ensemble de noeuds d'animation commencant par 'debut_chemin_scene'.

III.2 Manipuler une séquence d'images

ACTION dans le framework : sur le noeud <jeu> faire 'new Level'

sur le noeud <level> faire 'new Sequence'

CLASSE: 'cooki3d::Sequence'

Un objet de cette classe permet de manipuler comme un média une instance de 'osg::Sequence' contenu dans la scène. Cette classe 'osg' est un outil qui permet dérouler une liste de frame précalculée en fonction du temps.

ATTRIBUTS xml/c++ :

> le nom de l'instance ('nom')

> booléen déterminant si l'instance doit prendre les valeurs du modèle automatiquement ou si il faut découper l'animation du modèle selon les trois paramètres 'debut', 'fin' et 'fps' de manière manuelle.

> (en mode manuel) numéro de la première "image" de la séquence ('debut')

> (en mode manuel) numéro de la dernière "image" de la séquence ('fin')

> (en mode manuel) nombre d'images par secondes ('fps')

> type de mouvement souhaité de la séquence ('loopMode')

> état de la séquence au démarrage de l'application ('sequenceMode')

> chemin du noeud osg contenant la séquence [osg::Sequence] ('nodeName')

III.3 Manipuler un son

ACTION dans le framework : sur le noeud <jeu> faire 'new Level'

sur le noeud <level> faire 'new Sound'

CLASSE : 'cooki3d::Sound'

Cette classe utilse la SDL pour lire et mixer les fichiers '.wav'. Elle peut être utlisée pour les petits son et bruitages.

ATTRIBUTS xml/c++ :

> le nom de l'instance ('nom')

> le nom du fichier à chargé ('fic')

> le volume du son ('pcVolume')

> boolean déterminant si il faut lancer la lecture au démarrage ('playOnStart')

III.4 Manipuler un son mp3

ACTION dans le framework : sur le noeud <jeu> faire 'new Level'

sur le noeud <level> faire 'new SoundMpeg'

CLASSE : 'cooki3d::SoundMpeg' (SUPER CLASSE : 'cooki3d::Sound')

Cette classe utilse la SDL et libsmpeg pour lire et mixer les fichiers '.mp3'. Elle peut être utlisé pour les gros fichiers.

ATTRIBUTS xml/c++ :

Les attributs sont hérités et similaires à la classe 'cooki3d::Sound'

III.5 Manipuler une barre de temps "threadée" et ses commandes clefs

ACTION dans le framework : sur le noeud <jeu> faire 'new Level'

sur le noeud <level> faire 'new TimeBar'

sur le noeud <timeBar> faire 'new Command'...

CLASSE : 'cooki3d::TimeBar' et 'cooki3d::Command'

La barre de temps évolue parallèlement (Thread) au proccesus de l'application et execute les commandes paramétrés au fûr et à mesure du déroulement du temps. Une commande peut être une fonction ou une méthode C++.

Ainsi une commande est associée à un code. Il nécéssite donc la création d'un bloc 'library' qui sera lié au level concerné. Le patron du code sera généré automatiquement si l'option 'autogen' reste cochée. Ainsi l'utilisateur a juste à ajouter le code spécifique qui doit être executer lors du déclenchement de la commande.

ATTRIBUTS xml/c++ de 'cooki3d::TimeBar':

> le nom de l'instance ('nom')

> la taille de la barre de temps en secondes ('timeSize')

> booléen indiquant s'il faut lire en boucle ('loop')

> booléen déterminant si il faut lancer la lecture au démarrage ('playOnStart')

ATTRIBUTS xml/c++ de 'cooki3d::Command':

> le nom de l'instance ('nom')

> l'instant ou la commande sera executée ('time')

> booléen déterminant si le code est auto généré ('autogen')

> le code associé à la commande ('code')

TIPS:

> pour grouper plusieurs commandes sur un même code, vous pouvez décocher l'option 'autogen' et utiliser un code déjà existant d'une autre commande.

NEXT VERSION:

La prochaine version (>=0.22) supportera :

> la possibilité d'utiliser des commandes sous forme de script (tcl, lua...). Cela a déjà été réalisé dans le projet Zdm en tcl par un simple héritage sur la classe 'cooki3d::Command'.

IV. Les terrains

IV.1 Manipuler un terrain (un champs d'altitudes)

ACTION dans le framework : sur le noeud <jeu> faire 'new Level'

sur le noeud <level> faire 'new SolGravitation'

CLASSE : 'cooki3d::SolGravitation'

Cette classe permet de charger (éventuellement de générer) et d'utiliser un champs d'altitude. Celle-ci est principalement utilisée par les classes héritant 'cooki3d::Character'. Il peut y avoir plusieurs champs d'altitudes chargés (instance de 'SolGravitation'), c'est la première altitude trouvée pour la position donnée qui sera prise.

ATTRIBUTS xml/c++ de 'cooki3d::SolGravitation':

> le nom de l'instance ('nom')

> le fichier à partir duquel les altitudes sont calculées ('heightMapGeo')

> booléen déterminant si le fichier terrain doit apparaitre dans la scène

> mode de détection choisi. Il y a deux modes possibles 'MODE_HEIGHTMAP' où une 'HeightMap' (une champs d'altitudes) précalculée est utilisé ou bien le 'MODE_DYNAMIC' qui effectue un lancé de rayon vertical à chaque consultation ('altitudeDetection').

> booléen déterminant si le champs d'altitudes doit apparaitre ('heightMapVisible')

> position et taille de la 'heightmap' ('x', 'y', 'altitude', 'scale')

NOTE:

> Il faut bien veiller à ce que la position et taille de la 'heightmap' corresponde à celles du fichier modèle qui sera visible, sinon les altitudes renvoyées ne correspondront pas à la position demandée et risque même sortir de la map et provoquer une erreur.

IV.2 Manipuler un terrain de collision (un champs d'altitude)

ACTION dans le framework : sur le noeud <jeu> faire 'new Level'

sur le noeud <level> faire 'new SolCollision'

CLASSE : 'cooki3d::SolCollisions'

Cette classe permet de gérer des détections de collision via un champs d'altitudes. Elle hérite de tous les attributs de 'cooki3d::SolGravitation'. Dans le fonctionnement une instance de 'SolCollision' sera couplée avec l'instance courante de 'SolGravitation'. Pour qu'une collision soit détectée il faut que la hauteur prise sur l'instance de 'SolCollision' soit différente de celle prise sur l'instance de 'SolGravitation'. La map de 'SolCollision' représente en faite le terrain avec en plus les objets bloquants dispercé sur celui-ci.

ATTRIBUTS xml/c++ de 'cooki3d::SolCollision':

Strictement les mêmes que pour 'cooki3d::SolGravitation'

NOTE:

> Ce sont les instance de 'cooki3d::Character' qui vont utiliser et coupler les 'SolCollision' avec les 'SolGravitation'. Il faut donc bien veiller à l'homogénéité du décors qui sera visible (position et taille) avec la map de 'SolCollision' et de 'SolGraviation'.

V. Divers

V.1 Gérer une caméra contenue dans un fichier modèle

ACTION dans le framework : sur le noeud <jeu> faire 'new Level'

sur le noeud <level> faire 'new Camera'

CLASSE : 'cooki3d::Camera'

Il s'agit de gérer une caméra contenue dans un modèle (par exemple une vue définie par un infographiste dans 3dsmax). Le plugin d'exportation osgExp exporte/traduit une caméra en un objet osg::MatrixTransform. La classe 'Camera' va donc récupérer ce noeud et permettre à l'utilisateur de 'switcher' sur celle-ci.

ATTRIBUTS xml/c++ de 'cooki3d::Camera':

> le nom de l'instance ('nom')

> le chemin du noeud dans le modèle ('node')

V.2 Utiliser un noeud contenue dans un fichier modèle

ACTION dans le framework : sur le noeud <jeu> faire 'new Level'

sur le noeud <level> faire 'new Camera'

CLASSE : 'cooki3d::OsgNodeReference'

Un OsgNodeReference peu faire référence à n'importe quel type de noeud contenu dans le modèle. Cette référence est récupéré au moment du chargement pour l'utilisateur puisse ensuite accèder directement au noeud qu'il a paramétré (même méthode que pour 'cooki3d::Camera').

ATTRIBUTS xml/c++ de 'cooki3d::OsgNodeReference':

> le nom de l'instance ('nom')

> le chemin du noeud dans le modèle ('nodeName')

> booléen déterminant si l'utilisateur veut insérer un noeud de transformation intermédiare au dessus du noeud qu'il a choisi pour pouvoir le déplacer facilement ('insertTransform')

> type de la transformation à insérer ('transformType')

V.3 Créer un ligne directrice interpolée pour gérer un mouvement scénarisé

CLASSE 'cooki3d::GuideLine'

Cette classe n'existe pas encore. Elle devra servir à pouvoir donner à un personnage un mouvement déjà préparé/précalculé dans un modèle pour qu'il execute une trajectoire automatiquement, trajectoire qui sera généralement associé à une action précise.

TIPS:

> On peut faire cela manuellement en créant des osg::AnimationPath puis, en les affectant à l'animation path courant du personnage.

NEXT VERSION:

La prochaine version (>=0.22) supportera :

> Création et disponibilité de cette classe

V.4 Déclarer un groupe de paramètres xml

ACTION dans le framework : sur le noeud <jeu> faire 'new Level'

sur le noeud <level> faire 'new GroupParam'

CLASSE 'cooki3d::GroupParam'

Cette classe permet de passer des paramètres à l'application sous forme xml. Elle est principalement utilisée pour réaliser des plugins pour le framework cooki3d.

V.5 Ajouter un chemin de recherche pour une 'font' ou une image

ACTION dans le framework : sur le noeud <jeu> faire 'new Level'

sur le noeud <level> faire 'add filePath'

CLASSE Aucune

V.6 Ajouter un membre de classe dans un plugin

ACTION dans le framework : sur le noeud <jeu> faire 'new Level'

sur le noeud <level> faire 'add userMember'

CLASSE: Aucune

Un membre/attribut de classe sera associé à la classe attachée au level (exemple pour level0 : CLevel0). Il nécéssite donc la création d'un bloc 'library' qui sera lié au level concerné. Le patron du code sera généré automatiquement. Ainsi l'utilisateur pourra utiliser un attibut de classe spécifique qu'il faudra qu'il initialise et utilise dans les zones prévues à cet effet.