



Elementary

Documentation technique

Elementary

Elementary est le nom de notre équipe de projet, composée de **Adrian Palumbo**, **Youssef Oulehri**, **Mathieu Vandeginste**, et **François Demiguel**.

Développement

Plus de 100 heures de développement ont été nécessaires pour développer cet éditeur de monde en 3D qui s'inspire du fameux "Minecraft", avec plus de choix en terme de personnalisation.

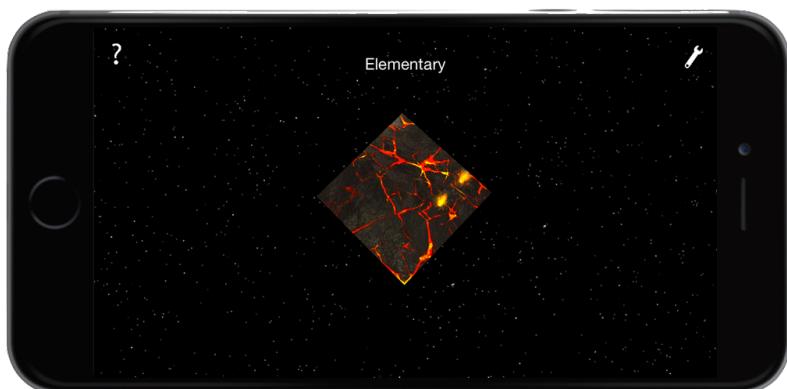
SceneKit

SceneKit est le framework d'Apple pour le **Swift** qui permet la manipulation d'objets 3D, avec une gestion poussée des caméras, lumières, particules, collisions et grandeurs physiques telles que la gravité.

Introduction

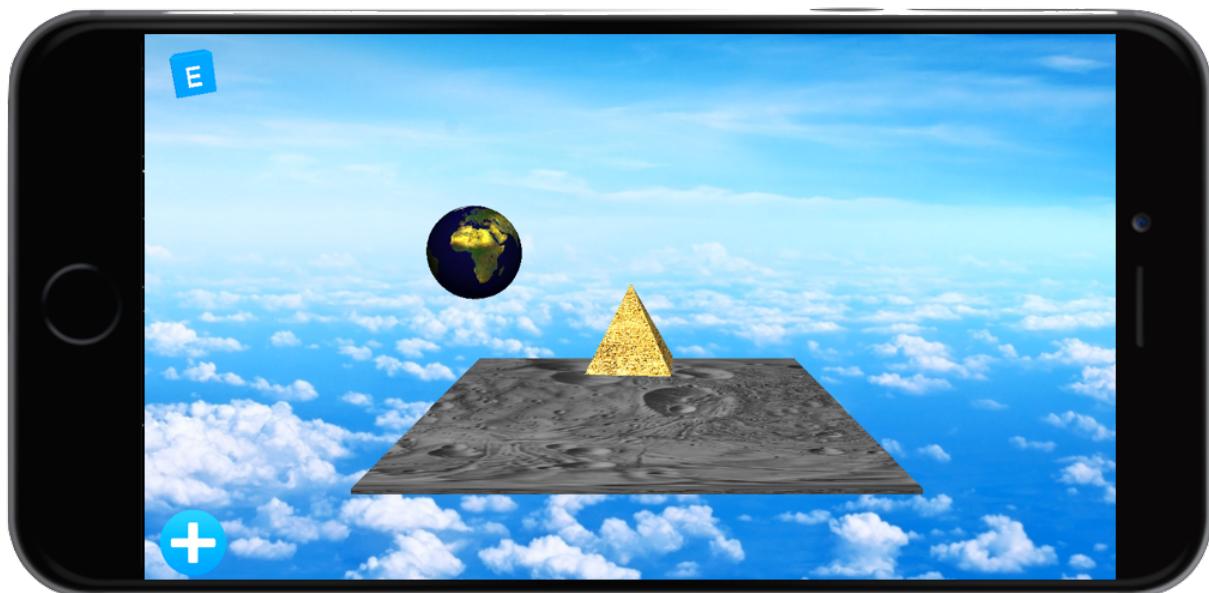
Cette documentation technique explique nos choix d'implémentation, et montre tous les détails à propos :

- *du format de l'application (langage et environnements d'exécution)*
- *du moteur 3D*
- *du diagramme de classe*
- *de la bibliothèque logicielle pour l'interface graphique (GUI toolkit), ici SceneKit pour Swift*



Sommaire

<i>Introduction</i>	<i>I</i>
<i>Sommaire</i>	<i>2</i>
<i>1. Structure du jeu</i>	<i>3</i>
<i>2. Diagramme de classes</i>	<i>6</i>
<i>3. SceneKit, moteur 3D Swift</i>	<i>8</i>



Swift



Swift est le nouveau langage d'Apple, très haut niveau, qui remplace l'Objective-C.

Dévoilé lors de la WWDC le 2 Juin 2014, le Swift est un langage **puissant, léger** et **sûr** qui s'inspire du meilleur de nombreux langages de programmation tels que Objective-C, Haskell, Ruby, Python ou C#.

Notre éditeur de monde Elementary est un jeu vidéo en Swift basé sur l'API Scene Kit d'Apple (moteur graphique 3D).

Un monde d'opportunités

Le 8 juin 2015, lors de la **WWDC15** (Worldwide Developer Conference) Apple a annoncé que Swift va prochainement passer en version 2.0 et devenir un langage **Open Source**.

Cela offre aux développeurs de nouvelles opportunités. Apple fournira très prochainement un compilateur Swift pour Linux, et d'autres entreprises qui développent des IDE (comme par exemple **Jetbrains**) vont annoncer un IDE Swift pour Windows. En effet, le passage en Open Source permettra de déployer ce langage dans d'autres architectures.

I. Structure du jeu

Elementary est développé en **Swift** avec l'IDE Xcode d'Apple pour les plateformes **iOS** (iPhone, iPad, iPod). Cet éditeur de monde est une **app universelle** (version unique pour tous les appareils iOS) et répond de manière "responsive" aux différentes tailles d'écran (iPhone 4, 6+, iPad mini ou Air).

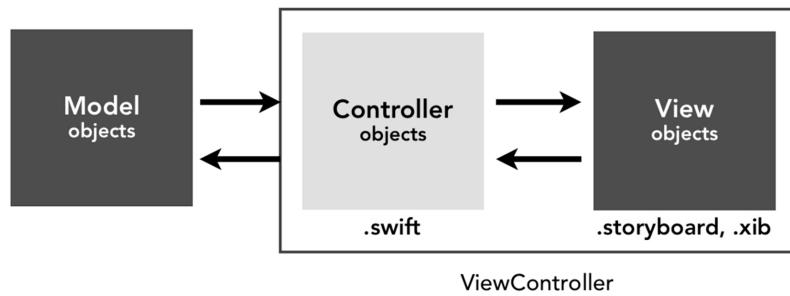
Etant donné la structure du jeu et son langage, le porter sur Mac (OS X) est tout à fait envisageable car peu de changements sont à effectuer (même logique, même langage, mais de simples ajustements pour une interface non tactile).

De plus, l'application pourrait être portée sur d'autres plateformes concurrentes dans les prochaines années (voir colonne de gauche - un monde d'opportunités).

Le Swift est un langage haut niveau orienté objet et nous reviendrons plus tard sur la structure objet de notre jeu dans la deuxième partie (diagramme de classe).

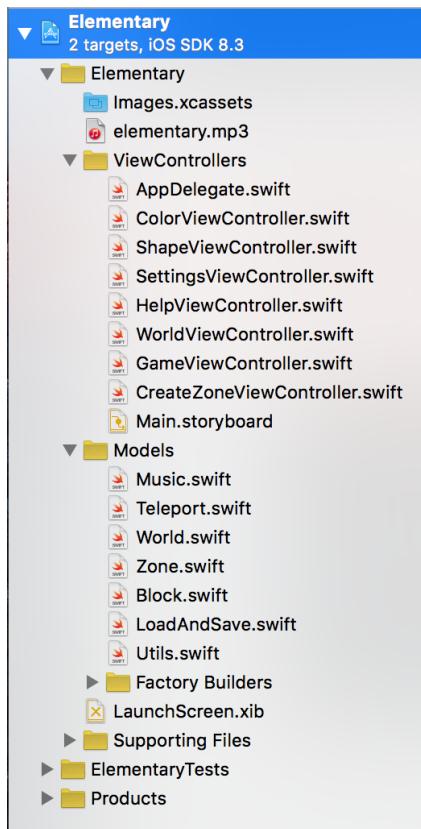
Dans **Xcode**, nous devons respecter un **design pattern** de type **MVC** (Model View Controller) afin de bien décorrler la vue, ses contrôles de la logique.

Model-View-Controller (MVC)



Notre application suit donc ce modèle avec :

- une logique avec notamment les classes World, Zone, Block
- des contrôleurs de vue (ViewControllers.swift) qui vont associer du code aux éléments graphiques (Scène SceneKit, boutons, fenêtres, champs de textes, pop ups, sliders et labels)
- une vue (main.storyboard) qui présente l'architecture visuelle de l'app (agencements et liens entre les différentes fenêtres, et "segments" ou segues qui les lient entre elles).



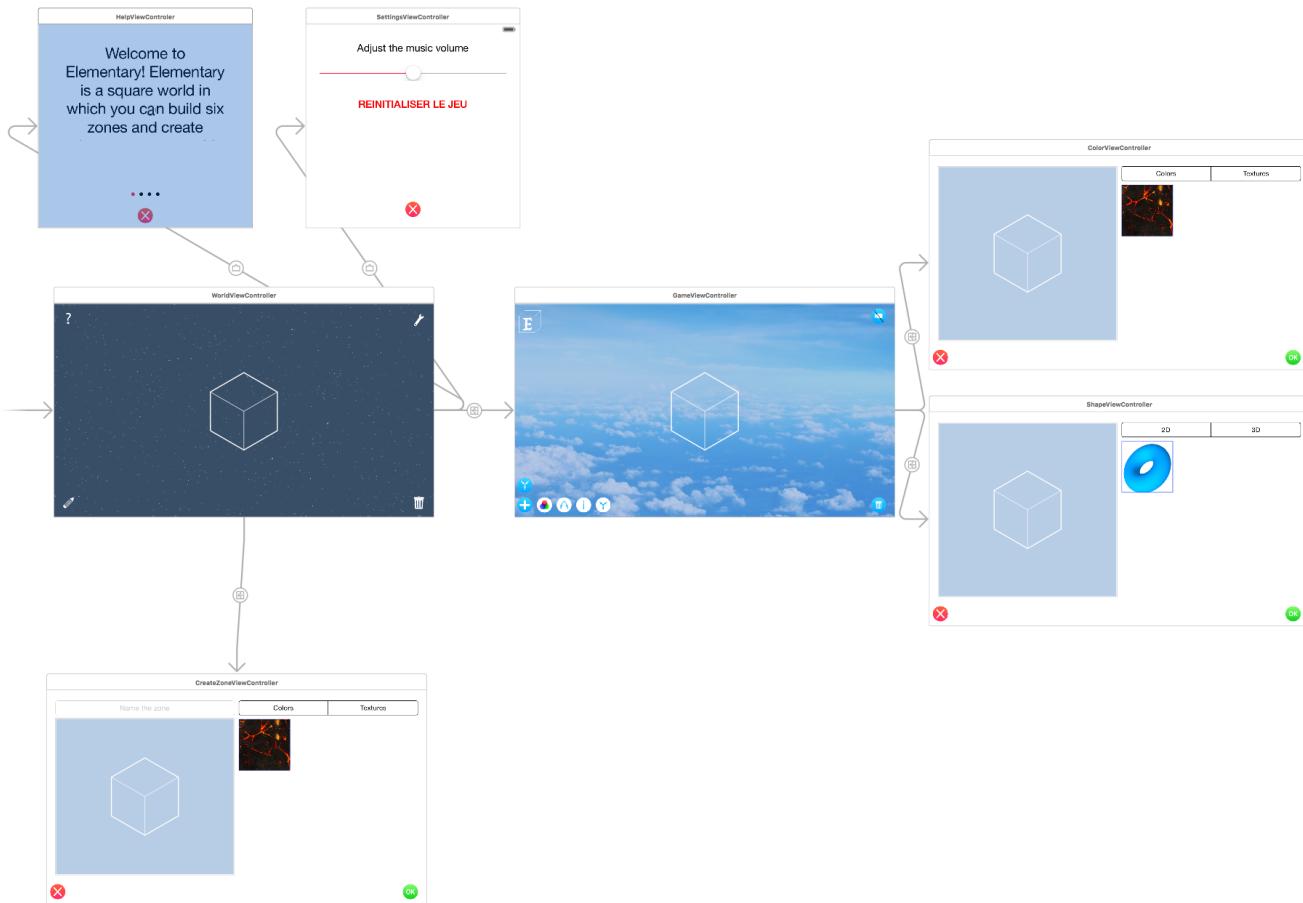
Voici l'**architecture MVC de notre app**. Ci dessous, on voit le contenu de notre **main.storyboard**.

L'application démarre sur **AppDelegate.swift** puis affiche **WorldViewController.swift** (la planète Elementary).

L'utilisateur peut alors accéder aux menus popups réglages et aide. Il peut ainsi créer, modifier et/ou supprimer une zone dans **CreateZoneViewController.swift**.

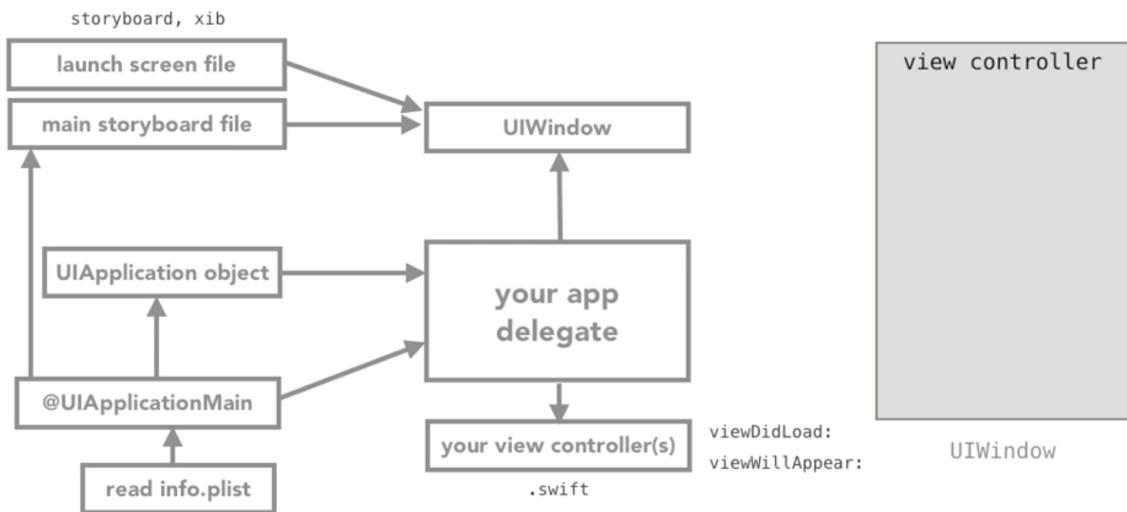
Il rentre ensuite dans la zone sélectionnée, qui appelle alors le **GameViewController.swift**.

De là, il pourra accéder à **ShapeViewController.swift** pour créer de nouvelles formes, et à **ColorViewController.swift** pour appliquer soit une couleur, soit une texture, à sa forme.



Pour mieux comprendre le fonctionnement de notre app, regardons le cycle de vie d'une app sous iOS :

iOS Application Lifecycle in Swift



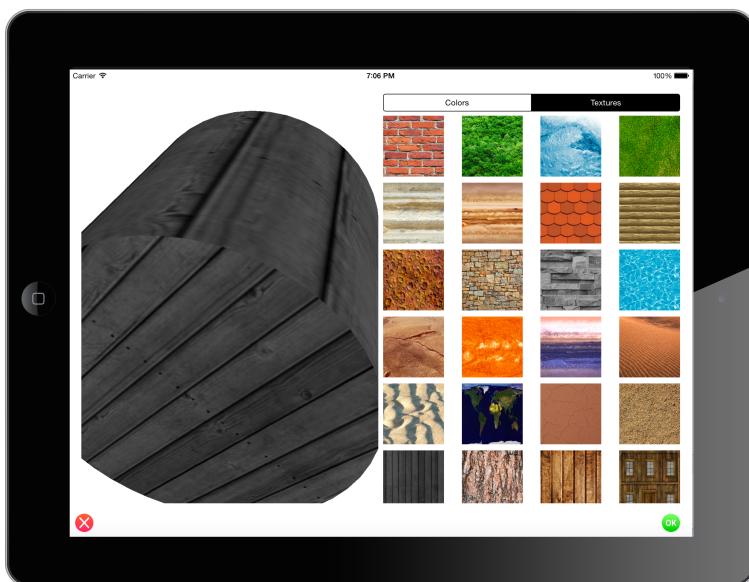
Les **délégués** sont une notion importante en Swift, pour pallier à un héritage simple (et non multiple).

Une classe autonome va **déléguer** des pouvoirs à une autre classe (on spécifie le lien avec le fichier .swift concerné dans l'Interface Builder). Ici, notre AppDelegate.swift va lancer notre premier ViewController, WorldViewController.swift.

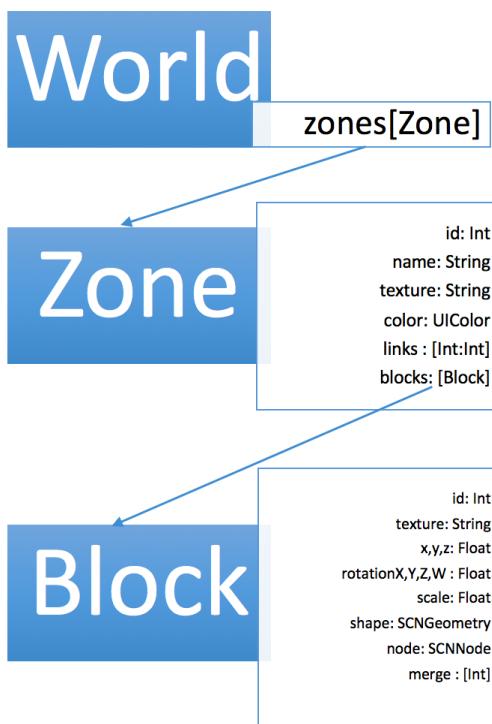
Un exemple d'utilisation des délégués dans Elementary : la classe **UICollectionView**. Cette classe, qui hérite de UIView et donc dUIKit (framework avec la GUI d'iOS pour Swift), permet d'afficher nos palettes de textures ou de formes. Elle est

autonome, et dispose de ses propres méthodes et attributs pour fonctionner (déplacement, sélection d'objets, gestion des cases, des sections...).

Cependant, on doit lui fournir des sources et des comportements, donc on **délègue** cette classe dans notre ShapeViewController.swift ou encore dans notre ColorViewController.swift qui vont se charger de fournir des données et des comportements à cette classe.



2. Diagramme de classes



Notre monde est un cube de six faces, chaque face étant une zone. Il y a donc une relation de **composition** entre notre World et son tableau de 6 Zones.

En effet l'héritage ne se prête pas à l'architecture de notre jeu.

De même pour les zones qui contiennent un tableau dynamique de Block.

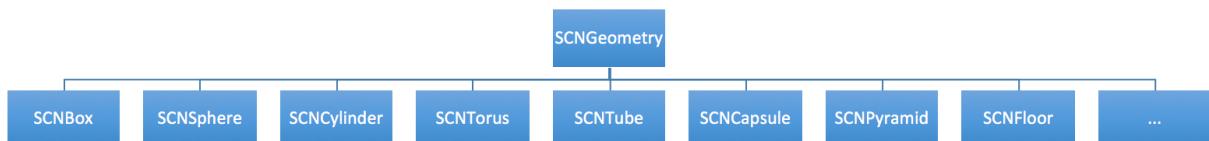
Les zones ont pour propriétés :

- un id unique
- un nom
- une couleur ou texture
- des liens pour la fonctionnalité de téléportation (numéro du monde pointé : numéro objet pointé)
- le tableau dynamique de blocks

Les blocks ont pour propriétés :

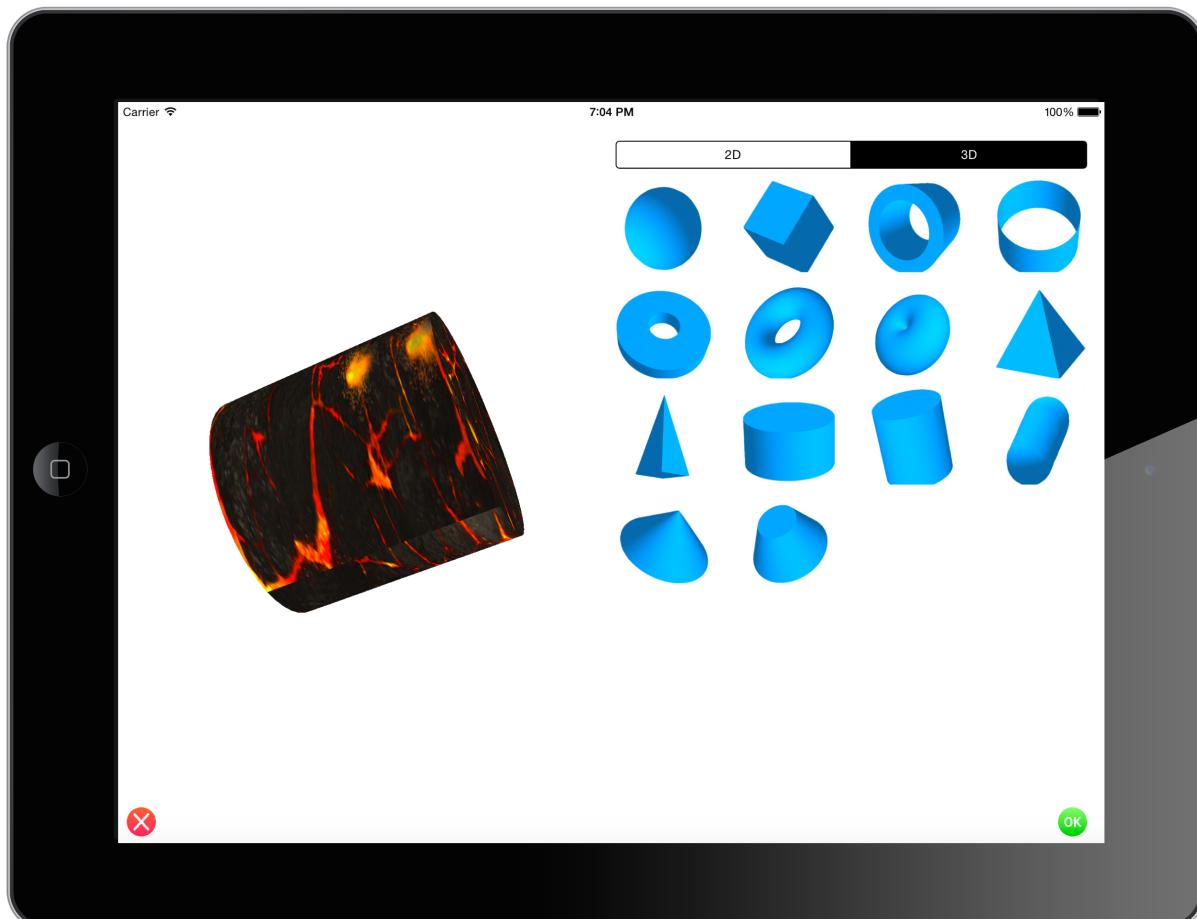
- un id unique
- une texture ou couleur
- une position x,y,z dans le monde
- une échelle pour la taille (ratio par rapport à la taille initiale)
- des coordonnées de rotation en Vecteur 4D (X,Y,Z,W)
- une forme "shape" qui contient l'objet **SceneKit SCNGeometry (voir ci-dessous)**
- un tableau d'entier qui contient les ID des autres blocks avec lesquels il est associé

Dans l'architecture de SceneKit, il y a des relations d'**héritage simple**, notamment pour les **SCNGeometry**.



Nous n'avons pas pu appliquer de relation d'héritage logique entre les formes 3D qui hériterait de formes 2D comme le voudrait la logique géométrique du fait de la conception intrinsèque du Framework SceneKit, qui ne gère pas nativement les formes 2D (SpriteKit s'en occupe, mais ce n'est pas notre technologie). Chaque forme est donc créée et personnalisée avec les éléments donnés par le framework.

Nous reviendrons dans la prochaine partie sur les autres relations d'héritage utilisée dans SceneKit pour expliquer le fonctionnement global du framework d'Apple.

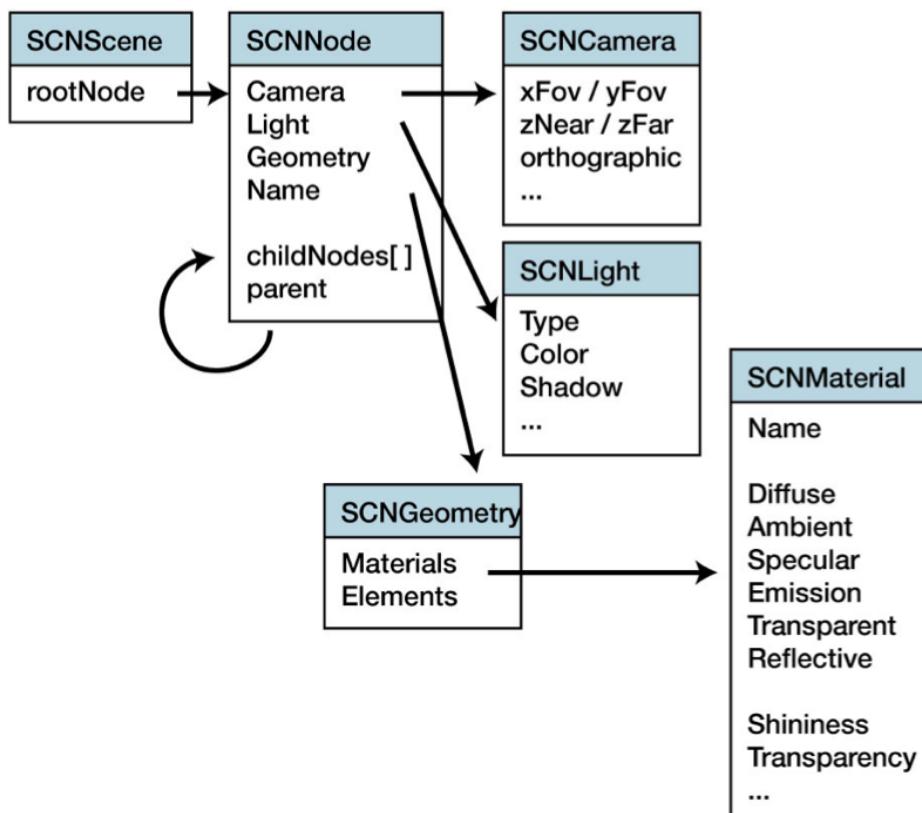


3. SceneKit, moteur 3D Swift

SceneKit est un framework puissant, très haut niveau. Il est cependant très complexe car capable de gérer de nombreux facteurs du monde réel, notamment :

- les grandeurs physiques, telles que la gravité
- la gestion poussée des collisions
- la gestion des particules
- les lumières
- les caméras
- les transformations avancées (transformations vectorielles ou matricielles, quaternions, pivot et rotation)

Voici comment s'architecte globalement le framework avec des relations d'héritage :



Notre jeu est affiché dans une scène **SCNScene** où l'on dispose un **rootNode** (objet racine), qui va être le parent d'autres **Nodes** (nos différents blocks).



Lorsque nous allons utiliser la fonctionnalité **MERGE** de fusion de plusieurs blocks, nous allons créer un Node parent, afin de regrouper nos nodes fils.

Par exemple, cette maison comporte un node cube et un node pyramide pour le toit. Ils sont tous réunis dans un node parent.

Une fois associés, on peut les déplacer **d'un seul coup**, grâce à ce node parent.

Les **SCNLight** et **SCNCamera** gèrent les lumières du monde et la caméra (qui a sa propre position, son angle...).

Chaque Node est composé de sa **SCNGeometry** étudiée plus haut. La Geometry contient notamment les tailles (pour une pyramide, par exemple, x,y,h et pour une sphère un radius), et un **SCNMaterial** qui contiendra la couleur (ou la texture) mais aussi la réflexion (par défaut blanche) et l'émission (utilisée par la sélection en rouge d'éléments dans Elementary).

Conclusion

Notre application a été **codée avec soin**, et **commentée** pour une meilleure compréhension. De plus, elle est prévue pour se mettre **facilement à jour**.

Nous restons à disposition pour toute information technique complémentaire, à propos de l'architecture de notre app, du Swift ou de son Framework SceneKit.

