UNIVERSITÀ DEGLI STUDI DI PADOVA
Dipartimento di Ingegneria dell'Informazione
Laurea Magistrale in Ingegneria Informatica

Sviluppo di una App educativa su piattaforma iOS con integrazione di un motore di fisica 2D

Andrea Stevanato (1039032)

Relatore: Sergio Congiu

Tutor: Nicola De Bello

09 Dicembre 2013

Indice

Elenco delle figure

Capitolo 1

Introduzione

La storia degli smartphone per il mercato di massa comincia con il successo dell'iPhone di Apple, nel 2007 l'azienda di Cupertino sfida il mercato mondiale col prodotto concepito proprio secondo la filosofia user friendly. Nel giro di pochi anni l'iPhone diventa un prodotto di riferimento per il mercato mondiale degli smartphone. Nel Gennaio 2010, viene presentato il primo iPad, un dispositivo multi-touch, con uno schermo da 9,7 pollici, in grado di riprodurre contenuti multimediali, videogames, e con la possibilità di accedere ad Internet.

L'ulteriore sviluppo tecnologico ha permesso di ottenere dispositivi sempre più piccoli e in grado di offrire funzionalità paragonabili a quelle dei personal computer, tanto che nel primo quarto del 2012 sono state registrate più vendite di smartphone e tablet rispetto ai personal computer. Una delle cause principali di questa tendenza è il grande parco applicazioni messo a disposizione degli utenti e supportato dagli store ufficiali delle piattaforme.

Lo sviluppo di applicazioni per dispositivi mobili è uno dei settori maggiormente in crescita che nel campo IT, e sta acquisendo sempre maggiore attenzione da parte dei programmatori, è sicuramente agevolato dalla maturità delle piattaforme principali, iOS e Android, ed è incentivato dalla facilità di raggiungere il mercato a livello globale.

Perché un prodotto abbia successo è necessario puntare sugli elementi di innovazione e di originalità, per differenziarsi dalle molte alternative che popolano il mercato. E' proprio in questo contesto che si inserisce il lavoro di tesi, la quale si propone di studiare la piattaforma iOS e sviluppare un'applicazione per tablet iOS che soddisfi determinati requisiti.

1.1 Contesto Aziendale

Il lavoro di tesi è stato svolto nel quadro di un tirocinio presso l'azienda Ware's Me di Padova.

Ware's Me è una giovanissima azienda padovana con la inconsueta missione di coniugare tecnologia e arte. Due tra i soci fondatori (Nicola De Bello e Michele Morbiato) hanno quindici anni di comune esperienza imprenditoriale negli ambiti tecnologici, avendo fondato e gestito assieme varie aziende di successo, poi acquisite da gruppi sia nazionali che internazionali. Ad essi si aggiunge il contributo della scrittrice e creativa Silvia Sorrentino, il cui compito è appunto quello di immaginare le possibili applicazioni dei nuovi, innovativi strumenti e veicoli tecnologici, e di Stefano Trainito che cura la direzione artistica dell'impresa.

Ware's Me è attualmente focalizzata sulle frontiere della pubblicazione digitale, sulle opportunità di auto-pubblicazione e sulle innovative potenzialità artistiche offerte dalle piattaforme mobili Apple (e non solo). Ware's Me, ad esempio, offre a editori, autori e illustratori diversi servizi:

- servizi di supporto nei processi di conversione/pubblicazione digitale e/o di auto-pubblicazione.
- la trasformazione di libri per bambini (già esistenti come libri convenzionali) in applicazioni interattive per dispositivi portatili, con un modello di business che prevede per editori/autori solo revenue-sharing, e nessun costo up-front.
- la creazione di App interattive a partire da materiale nuovo, concepito apposta per esistere sia come App che come libro convenzionale. Esso mira a spingere e promuovere la produzione di artisti attraverso il proprio sito.

La prima uscita di un libro per bambini trasformato in App interattiva (nonché il primo caso di successo del modello di business di cui sopra) proviene da un titolo di Kite Edizioni (Chissà, una storia di Marinella Barigazzi illustrata da Ursula Bucher), prestigioso editore specializzato, ed è frutto della partnership Ware's Me-Altera.

L'ultimo successo in abito educativo di Ware's Me è l'applicazione Contabosco, un'app dedicata al mondo dei numeri e appartenente al progetto a Little Smiling Minds, il progetto educativo realizzato da Focus, Digital Accademia e Daniela Lucangeli. La novità rispetto alle altre applicazioni è l'area dedicata ai genitori, che permette di seguire i progressi del figlio, le cose che impara e come aiutarlo a far crescere ancora la sua intelligenza.

1.2 Scopo del tironicio

La tesi si inserisce all'interno delle attività di Ware's Me, di trasformazione di libri per bambini (già esistenti come libri convenzionali) in applicazioni interattive per dispositivi portatili. Il progetto, chiamato APP-KID, si avvale di un framework software per la produzione rapida di libri per bambini in formato App, principalmente per piattaforma iOS.

Il framework software è basato sul framework Sparrow, una libreria basata su UIKit di Apple. Il framework UIKit fornisce le classi necessarie per costruire e gestire l'interfaccia utente di un'applicazione per iOS, il sistema operativo Apple utilizzato in tutti i dispositivi mobili dell'azienda (iPod, iPhone, iPad).

La libreria gestisce l'applicazione in un singolo oggetto e inoltre fornisce oggetti e metodi per la gestione degli eventi, la visualizzazione dell'interfaccia, la gestione delle finestre, la gestione delle delle visuali, i controlli dell'applicazione ed è specificamente progettata per una interfaccia touch screen.

Il framework Sparrow consente di creare applicazioni interattive per dispositivi iOS in maniera semplificata rispetto ad UIKit, automatizzando vari aspetti dello sviluppo di applicazioni multimediali e quindi aumentando l'efficienza nello sviluppo di tali applicazioni. L'obiettivo principale del framework è la creazione di giochi 2D, ma Sparrow può essere utilizzato per tutte le applicazioni grafiche e multimediali come nel caso di APP-KID che non tratta veri e propri giochi 2D bensì applicazioni multimediali con elementi dinamici.

Il progetto APP-KID permette di semplificare ulteriormente lo sviluppo delle applicazioni multimediali e di standardizzarle secondo un modello di applicazione comune, in modo da poter effettuare la trasposizione di un libro illustrato dalla versione cartacea a quella multimediale con il minimo sforzo, pur mantenendo la possibilità di apportare le modifiche e le personalizzazioni richieste.

Le applicazioni multimediali di APP-KID hanno quindi una struttura comune e condividono le seguenti funzioni di base:

- Schermata iniziale Ware's Me.
- Schermata iniziale Editore, Autore(i) e Titolo
- Menù iniziale con le seguenti voci principali: Lettura, Indice, Gioca, Opzioni, Crediti.

- Menù con indice grafico delle pagine del libro, con la possibilità di accedere direttamente ad una singola pagina.
- Lettura pagina per pagina con o senza audio.

In ogni pagina dell'applicazione sono presenti le seguenti funzioni e i relativi pulsanti:

- Avanti.
- Indietro.
- Menù principale.
- Menù indice di gioco.
- Funzioni audio.
- Animazioni interattive che vengono attivate automaticamente una volta sfogliata la pagina.

In ogni applicazione possono essere inoltre implementate altre funzioni come:

- Puzzle, giochi di colorazione o scopri le differenze.
- Funzione di registrazione e riproduzione audio per l'utente, integrata nelle singole pagine.
- Multilingua per un massimo di 3 lingue supportate: italiano, inglese e francese.

APP-KID può essere utilizzato per creare applicazioni multimediali più complesse, che differiscono dalla struttura standard appena presentata. In seguito alla realizzazione delle prime applicazioni, visto il buon successo riscontrato nel pubblico, si è pensato di ampliare e migliorare l'offerta approfondendo la parte dedicata ai giochi, che pur essendo semplici da realizzare possono risultare poco avvincenti se confrontati con ciò che popola il mercato attualmente, proprio per la loro semplicità.

Per potenziare l'offerta ludica di APP-KID si è pensato di dotare il framework di una funzione particolare: la simulazione della fisica. La simulazione della fisica viene comunemente realizzata tramite un software detto motore fisico. Il motore fisico è svincolato dal resto dell'applicazione, e si occupa solamente di simulare la fisica degli oggetti per dare loro un credibile un movimento realistico. Anche se possono essere utilizzati per altre applicazioni, i motori fisici nascono principalmente come librerie per l'uso nei videogiochi.

L'obiettivo della tesi è stato quello di sviluppare un'applicazione APP-KID, con l'integrazione di un motore fisico 2D per lo sviluppo di mini giochi all'interno dell'applicazione.

Capitolo 2

Analisi delle esigenze

Lo sviluppo di un'applicazione di tipo APP-KID deve soddisfare diversi requisiti, imposti soprattutto dal contesto aziendale nel quale si inserisce, dalle prospettive del mercato alle quali è rivolta e dalla piattaforma di sviluppo utilizzata. Fondamentale è l'esecuzione efficiente, in termini di risorse utilizzate e gestione della memoria, in ognuno dei dispositivi supportati.

I dispositivi da supportare sono molti, bisogna porre molta attenzione ai dispositivi con background di almeno due anni: in questi dispositivi la memoria disponibile è ridotta rispetto a quelli di ultima generazione, così come le risorse del processore e del chip grafico. E' quindi essenziale effettuare test di performance su tutti dispositivi attualmente supportarti.

Il problema dell'efficienza è di capitale importanza quando, come in questo caso, le risorse messe a disposizione dai dispositivi (iPhone, iPad e iPod) sono ridotte rispetto a quelle di un pc ordinario; inoltre non è da sottovalutare l'aspetto del consumo della batteria: la batteria ha una durata molto limitata soprattutto per favorire una maggiore portabilità e delle ridotti dimensioni del dispositivo, è necessario quindi essere efficienti in termini di calcolo computazionale.

Un altro aspetto importante è la gestione della memoria, soprattutto nel caso dei dispositivi mobili, in cui vi è memoria in quantità limitata e non espandibile. Una non corretta gestione della memoria porta ad un problema chiamato *Memory Leak*[1] che causa l'esaurimento della memoria disponibile portando l'applicazione in crash. Questo appena descritto è un requisito critico da testare prima della fase di pubblicazione dell'applicazione. Ogni

[1]Memory Leak, in italiano perdita di memoria, è un consumo non voluto di memoria, dovuto alla mancata deallocazione della stessa, di variabili/dati non più utilizzati da parte dei processi che vengono lasciati caricati in memoria.

applicazione prima di essere messa in vendita nello store virtuale, viene attentamente controllata nei contenuti, nel funzionamento e nell'efficienza generale dallo staff Apple, grazie a questa procedura non verranno messe in vendita applicazioni con contenuti inappropriati o che causano il crash del dispositivo.

Per quanto riguarda la scelta del motore fisico 2D all'interno dell'applicazione APP-KID, è necessario che anch'esso soddisfi i requisiti delle risorse e gestione della memoria citati in precedenza. Come seconda cosa è importante che la libreria si inserisca nel framework di sviluppo senza comportare modifiche delle librerie attualmente utilizzate in modo mantenere una retrocompatibilità con le applicazioni già esistenti. Per questo motivo è apprezzabile che la libreria sia progettata sulla base del framework attuale senza creare ambiguità o duplicare funzioni già esistenti ma adattandosi alla sua struttura in modo da risultare coerente agli occhi dello sviluppatore.

Il framework APP-KID, rappresentato nella Figura 2.1 si appoggia a sua volta ad altri framework: UIKit e Media Layer (appartenenti ad iOS SDK), Sparrow e al motore di fisica 2D. Per la creazione di menù il framework può utilizzare direttamente UIKit che fornisce vari oggetti utili alla creazione grafica dei menù in maniera nativa, mentre per la parte multimediale dell'applicazione, dalla grafica all'audio e alla gestione degli input, utilizza Sparrow.

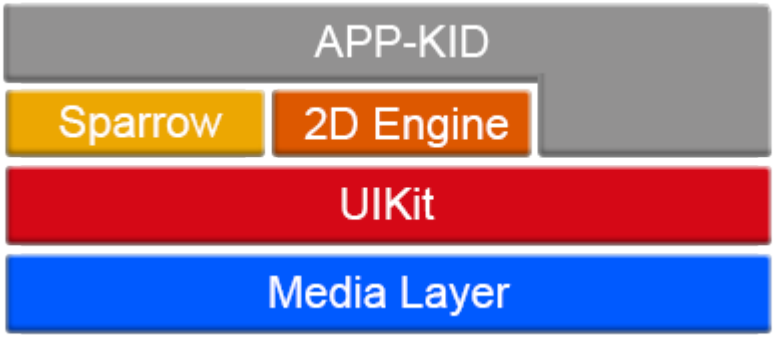

Figura 2.1: Livelli di framework

La libreria Sparrow è Open Source[2], è possibile quindi creare estensioni per libreria, è immediato quindi integrare il *2D Engine* a questo livello di architettura.

[2]Open source indica un software i cui detentori dei diritti ne permettono e favoriscono il libero studio e l'apporto di modifiche da parte di altri programmatori indipendenti.

Capitolo 3

Tecnologie Utilizzate

Nel seguente capitolo verranno analizzati gli strumenti di lavoro utilizzati durante lo sviluppo dell'applicazione, la maggior parte risulteranno delle scelte obbligate visto che sono messe a disposizione da Apple stessa.

L'applicazione è stata realizzata in modo specifico per piattaforme iOS, in particolare per dispositivi iPad, verrà quindi presentato SDK (Software Development Kit) specifico per i device portatili Apple. L'SDK comprende un insieme di framework messi a disposizione da Apple per la realizzazione di applicazioni sui propri dispositivi. Gli aggiornamenti dell'SDK (aggiornamento di framework esistenti o introduzione di nuovi) avviene su base annuale in contemporanea alla presentazione di nuovi device; i nuovi framework mettono a disposizione librerie che permettono di utilizzare le nuove feature che sono state introdotte con l'ultima generazione dei dispositivi.

Xcode è l'IDE (Integrated Development Environment), realizzato e messo a disposizione da Apple, utilizzato per lo sviluppo dell'applicazione. Per quanto riguarda il linguaggio di programmazione viene quello ampiamente utilizzato è l'Objective-C.

3.1 Linguaggio Objective-C

Objective-C, spesso citato anche come Objective C o ObjC o Obj-C, è un linguaggio di programmazione orientato agli oggetti, sviluppato da Brad Cox alla metà degli anni ottanta presso la Stepstone Corporation. Come lo stesso nome suggerisce, l'Objective C è un'estensione a oggetti del linguaggio C. La sua diffusione è principalmente legata al framework OpenStep di NeXT e al suo successore Cocoa, presente nel sistema operativo Mac OS X di Apple. Di seguito alcuni accenni storici e le principali caratteristiche.

3.1.1 La storia

Nel 1983 i fondatori della Stepstone, Brad Cox e Tom Love, sviluppano un nuovo linguaggio di programmazione partendo dal C, aggiungendovi un pre-processore capace di supportare le nuove capacità introdotte da Smalltalk (un linguaggio di programmazione orientato agli oggetti).

Questo nuovo linguaggio viene chiamato *Objective-C*, la differenza sostanziale rispetto ad altri linguaggi che hanno modificato il C (per esempio, C++ o C#), è che per ottenere una sua versione a oggetti , Objective-C lascia intatta tutta la parte sul C, mantenendo una retrocompatibilità totale.

Il coinvolgimento di Apple inizia in maniera indiretta nel 1985 quando Steve Jobs, dopo aver fondato la NeXT, nel 1988 acquista la licenza dell'Objective-C realizzando un proprio compilatore Objective-C e nuove librerie per sviluppare l'interfaccia utente di quello che sarebbe stato il NeXT Operating System.

Nel 1996 Apple acquista NeXT assieme alla licenza dell'Objective-C e al suo sistema di sviluppo Project Builder (in seguito rinominato Xcode), e con questi nuovi strumenti comincia a sviluppare quello che nel 2002 sarebbe stato presentato come Mac OS X.

Nel 2007 Apple presenta Objective-C 2.0, aggiungendo molte nuove proprietà alla prima versione del linguaggio, e lo utilizza per creare il primo iPhone OS. Un anno dopo, nel 2008, Apple apre le porte a sviluppatori di terze parti rilasciando l'iPhone SDK.

L'Objective-C oggi, viene utilizzato per sviluppare applicazioni su piattaforme MacOS X e iOS.

Figura 3.1: Storia dell'Objective-C

3.1.2 Objective-C per lo sviluppo in iOS

Objective-C nasce con lo scopo specifico di aggiungere il supporto agli oggetti al linguaggio C, questo significa che Objective-C supporta la sintassi del C aggiungendo il supporto per gli oggetti.

Ogni oggetto associa una porzione di dati ad una particolare operazione che può utilizzare o modificare tali dati. In Objective-C queste operazioni sono note come *metodi dell'oggetto* mentre la porzione di dati su cui agiscono vengono chiamate le *variabili d'istanza*. In Objective-C le variabili d'istanza degli oggetti sono interne all'oggetto: lo sviluppatore può accedervi solamente utilizzando i metodi definiti nella classe che rappresenta l'oggetto. Proprio come nel C, anche qui le variabili locali vengono nascoste dal resto del programma, infatti un oggetto nasconde sia le sue variabili d'istanza che le implementazioni dei suoi metodi.

Objective-C implementa tre modi di gestire la memoria. Uno è dinamico e completamente automatico, basato su un sistema chiamato *Automatic Garbage Collection* (AGC). Tipico di molti linguaggi orientati agli oggetti, libera automaticamente le risorse occupate da un oggetto secondo una data politica. Completamente opposto a quello dinamico e completamente automatico vi è il sistema manuale che si basa sulla gestione del *Manual Reference Counting* (MRC), che di fatto conta il numero di volte che un oggetto è riferito; quando il numero dei riferimenti è uguale a zero l'oggetto viene rimosso dalla memoria. Il terzo modo è l'*Automatic Reference Counting* (ARC), introdotto a partire da OS X 10.6 e iOS 5, che mantiene il paradigma del Reference Counting facilitando e alleggerendo il lavoro da parte dello sviluppatore.

Per permettere la comunicazione tra oggetti distribuiti su processi diversi, sia operanti sulla stessa macchina sia su macchine diverse, Objective-C supporta il *remoting*, ovvero la comunicazione tra tali oggetti remoti attraverso l'invio di messaggi.

Objective-C supporta anche il multithreading, ovvero l'esecuzione di più processi su più core del processore all'interno di una stessa applicazione mediante l'uso del *Grand Central Dispatch* (sviluppato da Apple per ottimizzare l'esecuzione delle applicazioni su sistemi multi core).

3.2 Xcode

Xcode è l'ambiente di sviluppo utilizzato per creare, testare, effettuare il debug delle applicazioni su piattaforma iOS. L'ambiente comprende diversi

tools di sviluppo che aiutano lo sviluppatore a realizzare le applicazioni, quelli più importanti sono:

- **Instruments (XRay)** è uno strumento utilizzato per analizzare e visualizzare le performance di un'app. In particolare mostra una time line per qualsiasi evento accada durante l'esecuzione dell'applicazione, come ad esempio attività della CPU, memoria allocata, dati di rete e gestione dei file, tutti insieme possono essere mostrati in grafici o per effettuare statistica. Un esempio è mostrato nella Figura 3.2.

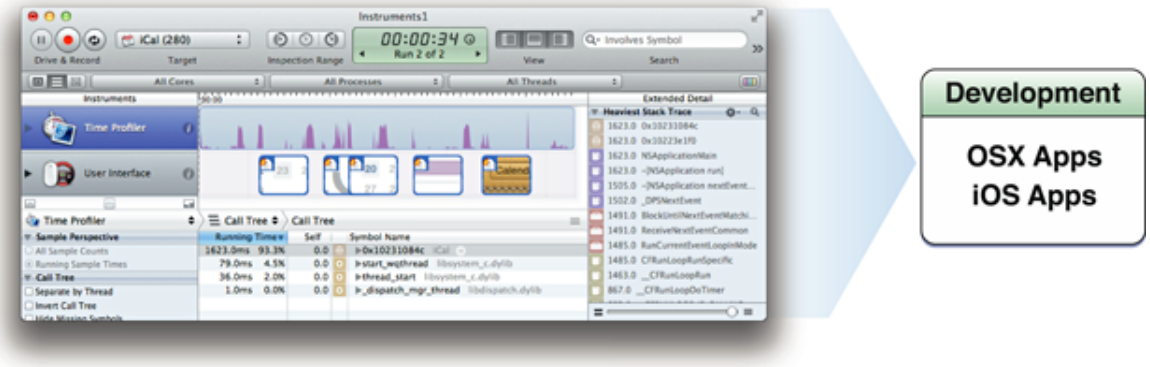

Figura 3.2: Instruments

- **iOS Simulator** è lo strumento che permette di eseguire le applicazioni direttamente nel calcolatore in cui viene fatto lo sviluppo. Possono essere simulati tutti i device attualmente supportati e tutte le versioni di iOS precedenti a quella attuale. iOS Simulator non permette la simulazione dell'accelerometro e della fotocamera, in quanto sono dispositivi hardware il cui utilizzo deve essere testato direttamente sul device fisico. La Figura 3.3 mostra l'iPad Simulator.

- **Application Loader** è un tool che aiuta lo sviluppatore a preparare le proprie app alla vendita su App Store. Le app, per essere inserite nello store di Apple, devono seguire un complesso iter di approvazione, dove viene verificato il funzionamento dell'applicazione e il rispetto delle linee guida imposte da Apple.

- **Printer simulator** (simulatore di stampa), **Graphic tools** (tool grafici) e molti altri strumenti sono disponibili nell'area Download della iOS Developer Library.

Figura 3.3: iOS iPad Simulator

Di seguito verrà descritto il cuore di Xcode, l'editor del codice, dove vengono scritti e gestiti tutti i file del progetto. La Figura 3.4 mostra come è organizzato lo spazio di lavoro: nella parte sinistra abbiamo il *Project Navigator*, nella parte centrale l'editor di testo, nella parte destra il *File Utilities* e infine nella parte inferiore l'area di *Output/Debug*

Altre utili caratteristiche di Xcode sono le seguenti:

- **Interface Builder**. E' possibile costruire ogni vista relativa alla propria applicazione utilizzando un apposito editor chiamato *Interface Builder*. Questo permette di inserire oggetti all'interno dell'interfaccia utente e di specificarne molti dettagli grafici (posizione, funzionalità, tipologia) e anche la loro connessione con l'applicazione stessa. Esiste comunque la possibilità di implementare tali oggetti via codice.

- **Assistant Editor**. Durante la fase di programmazione si è spesso costretti a modificare diversi file correlati tra loro. L'*assistant editor* permette al programmatore di visualizzare i file in questione nella stessa finestra senza bisogno di aprire più editor alla volta. Questa caratteristica velocizza di molto il flusso di lavoro, aiutando molto il programmatore.

- **Identificazione e correzione degli errori**. Xcode verifica il codice sorgente in tempo reale, evidenziando sia errori ortografici che errori di programmazione. La riga errata viene evidenziata fornendo i dettagli relativi all'errore, talvolta vengono proposte delle possibili soluzioni per risolverli.

- **Controllo del Codice Sorgente**, permette di salvare i file di progetto in una repository (Git, Bitbucket ecc.) direttamente da Xcode, mantenendo così un controllo della versione del progetto. Grazie a questa caratteristica è possibile lavorare in team allo stesso progetto, mantenendo uno storico delle modifiche fatte dai vari programmatori.

3.3 iOS SDK

3.3.1 Modello MVC

Una qualsiasi applicazione per iOS o per OS X che utilizza gli strumenti messi a disposizione dai vari framework, se progettata correttamente, è basata su un modello detto *Model View Controller* (MVC). MVC è un vero e proprio

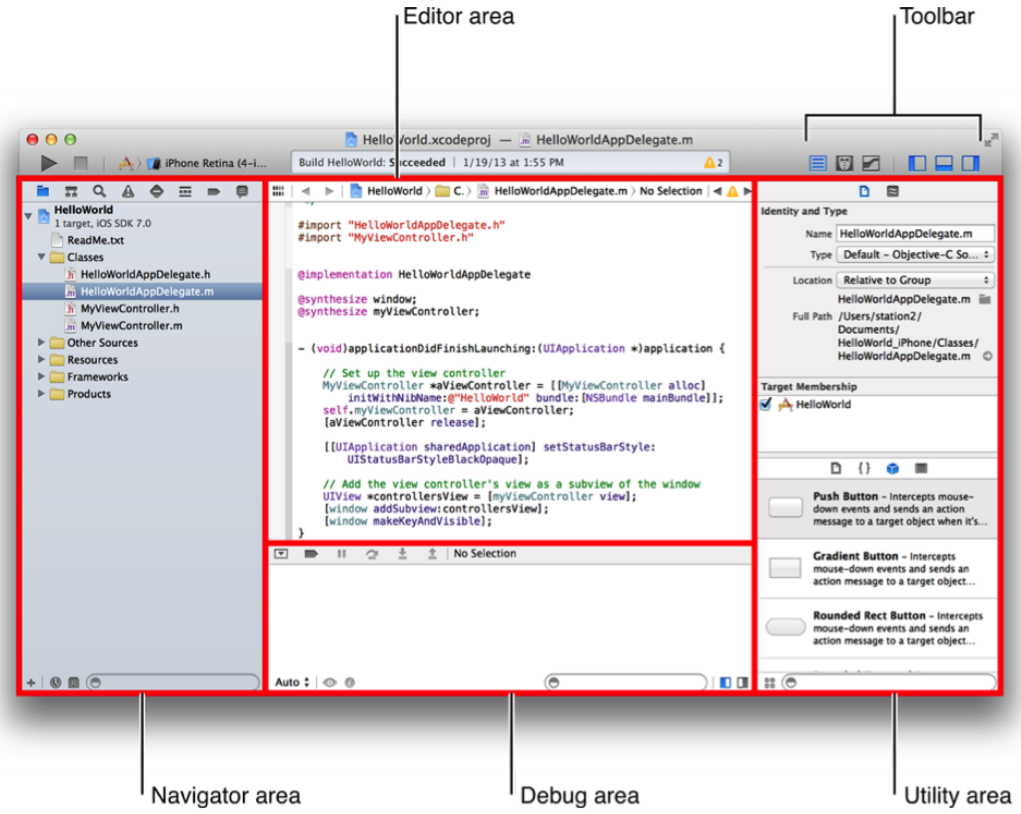

Figura 3.4: Xcode IDE

pattern architetturale creato negli anni '80 per consentire la presentazione multipla (in diverse forme) di un oggetto in varie interfacce grafiche senza dover modificare o adattare l'oggetto da presentare. MVC non solo realizza la separazione tra dati e interfaccia utente ma svincola anche la logica di controllo dell'applicazione dai dati. Il modello è basato sulla separazione dei compiti fra tre componenti software che interpretano tre ruoli principali:

- **Model**: gli oggetti che costituiscono il modello rappresentano particolari conoscenze e competenze, contengono i dati di un'applicazione e definiscono la logica che regola la manipolazione dei dati. Un'applicazione MVC ben progettata ha tutti i dati importanti incapsulati in oggetti del modello. Tutti i dati che fanno parte dello stato persistente dell'applicazione devono risiedere negli oggetti del modello una volta che i dati vengono caricati nell'applicazione. Idealmente, un oggetto del modello non ha alcun collegamento esplicito con l'interfaccia utente utilizzata per presentarlo e modificarlo. Tuttavia nella pratica la separazione dall'interfaccia non è sempre la cosa migliore, infatti vi è un certo margine di flessibilità nel modello realizzato in iOS, ma in generale un oggetto del modello non dovrebbe occuparsi di come è realizzata l'interfaccia.

- **View**: Un oggetto di tipo vista è in grado di visualizzare, e in alcuni casi di modificare, i dati del modello dell'applicazione. La view non dovrebbe essere responsabile per la memorizzazione dei dati che presenta, ma può memorizzare nella cache alcuni dati per motivi di prestazioni. Un oggetto della vista può essere responsabile della visualizzazione di solo una parte di un oggetto del modello, di un oggetto intero del modello o anche di molti oggetti del modello differenti. Le view sono disponibili in molte varietà diverse e tendono ad essere riutilizzabili e configurabili fornendo coerenza tra applicazioni diverse nello stesso sistema. Infatti il framework *UIKit* definisce un gran numero di oggetti di visualizzazione che possono essere riutilizzati assicurando lo stesso funzionamento in diverse applicazioni, garantendo un elevato livello di coerenza per aspetto e comportamento tra le applicazioni. Un oggetto di visualizzazione deve assicurarsi che i dati del modello siano visualizzati correttamente, di conseguenza ha bisogno di conoscere le modifiche apportate al modello.

- **Controller**: Poiché gli oggetti del modello non devono essere legati ad interfacce specifiche, hanno bisogno di un modo generico per segnalare il cambiamento. Un oggetto di controllo funge da intermediario tra gli oggetti della view dell'applicazione e gli oggetti del modello. I controller

hanno spesso il compito di fare in modo che la view abbia accesso agli oggetti del modello che devono presentare e quello di agire come canale attraverso il quale l'interfaccia sia aggiorna secondo le modifiche dei dati. Gli oggetti del controller possono anche configurare e coordinare le attività di una applicazione e gestire i cicli di vita di altri oggetti. In un tipico modello MVC, quando gli utenti immettono un valore o indicano una scelta attraverso un oggetto di visualizzazione, il valore o la scelta viene comunicato ad un oggetto di controllo. L'oggetto di controllo può interpretare l'input dell'utente in alcune applicazioni specifiche o può indicare ad un oggetto del modello cosa fare con questo ingresso.

Sulla base dello stesso input dell'utente, alcuni oggetti del controller, potrebbero anche indicare ad un oggetto di visualizzazione di modificare il proprio aspetto o il proprio comportamento. Al contrario, quando un oggetto del modello cambia, il modello comunica il cambiamento al controller, che penserà ad aggiornare di conseguenza gli oggetti di visualizzazione.

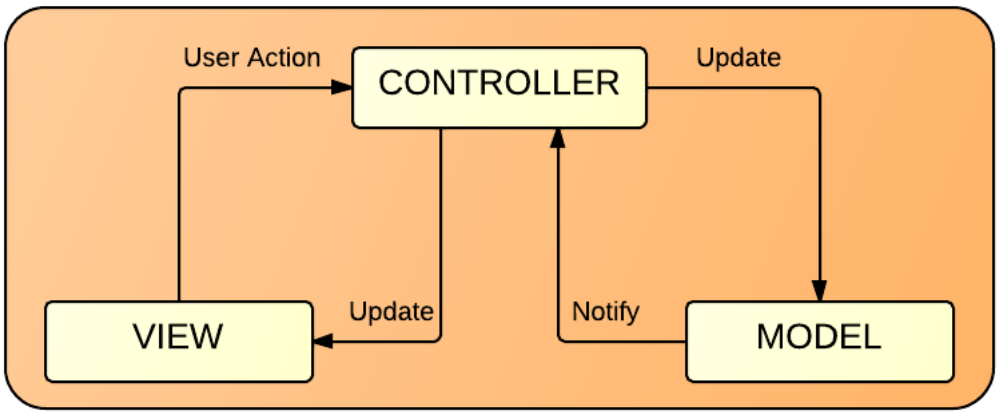

Figura 3.5: Paradigma MVC

Il modello MVC presenta alcuni aspetti positivi e negativi per lo sviluppo software. Le applicazioni progettate, secondo questo modello, godono di una separazione tra dati ed interfaccia molto solida essendo realizzata a livello progettuale. Inoltre la separazione della logica dell'applicazione nel controllore garantisce una facile espandibilità e manutenibilità dell'applicazione. Il

modello risulta vantaggioso anche nel caso in cui si sviluppino più applicazioni con elementi comuni, infatti la separazione dei compiti in moduli di tre tipi, permette di riutilizzare le classi velocizzando la fase di sviluppo.

D'altra parte, il modello risulta svantaggioso nel caso in cui non ci sia la necessità di riutilizzare parti del programma, infatti l'overhead dovuto alla progettazione piuttosto complessa e alla necessità di creare classi aggiuntive rispetto a quelle richieste da modelli più semplici può essere molto grande.

3.3.2 Architettura iOS

L'architettura del sistema operativo iOS si articola in più livelli sovrapposti. Al livello più basso, iOS agisce come intermediario tra l'hardware sottostante e le applicazioni che appaiono sullo schermo. Le applicazioni native non di sistema, è raro che comunichino direttamente con l'hardware del dispositivo, infatti esse comunicano con l'hardware attraverso un ben definito sistema di interfacce che proteggono le applicazioni dalle differenze di componenti che caratterizzano l'insieme di dispositivi mobili per i quali sono sviluppate. Questa astrazione fa si che sia facile creare applicazioni che funzionino con in maniera uniforme e stabile su dispositivi con diverse capacità hardware.

L'implementazione delle tecnologie di iOS può anche essere vista come una serie di strati, che sono mostrati in Figura 3.6. Ai livelli più bassi del sistema ci sono i servizi fondamentali e le tecnologie sulle quali si basano tutte le applicazioni mentre i livelli più alti contengono i servizi e le tecnologie più sofisticati.
Quando si sviluppa una applicazione, dove possibile, si deve prediligere l'uso di *framework* di alto livello piuttosto che dei servizi offerti dai livelli inferiori. Infatti i frameworks di livello superiore sono stati creati appositamente per fornire astrazioni orientate agli oggetti dei servizi e delle funzioni offerti dai costrutti dei livelli inferiori. Queste astrazioni, in genere, rendono molto più facile la scrittura di codice, perché riducono la quantità di righe di codice da scrivere e racchiudono caratteristiche potenzialmente complesse, come i socket per le connessioni e i thread. Sebbene forniscano una interfaccia per le tecnologie di livello inferiore, non si sovrappongono ad esse nascondendole dal punto di vista dello sviluppatore: i frameworks di livello inferiore sono ancora disponibili per gli sviluppatori che preferiscono usarli o che vogliono utilizzare gli aspetti di tali strutture che non sono esposti dagli strati superiori.

Apple espone la maggior parte delle sue interfacce di sistema in pacchetti speciali chiamati *framework*. Un framework è una directory che contiene una libreria dinamica condivisa e le risorse (ad esempio, file di intestazione, immagini, applicazioni di supporto, ecc.) necessarie per a supporto di tale

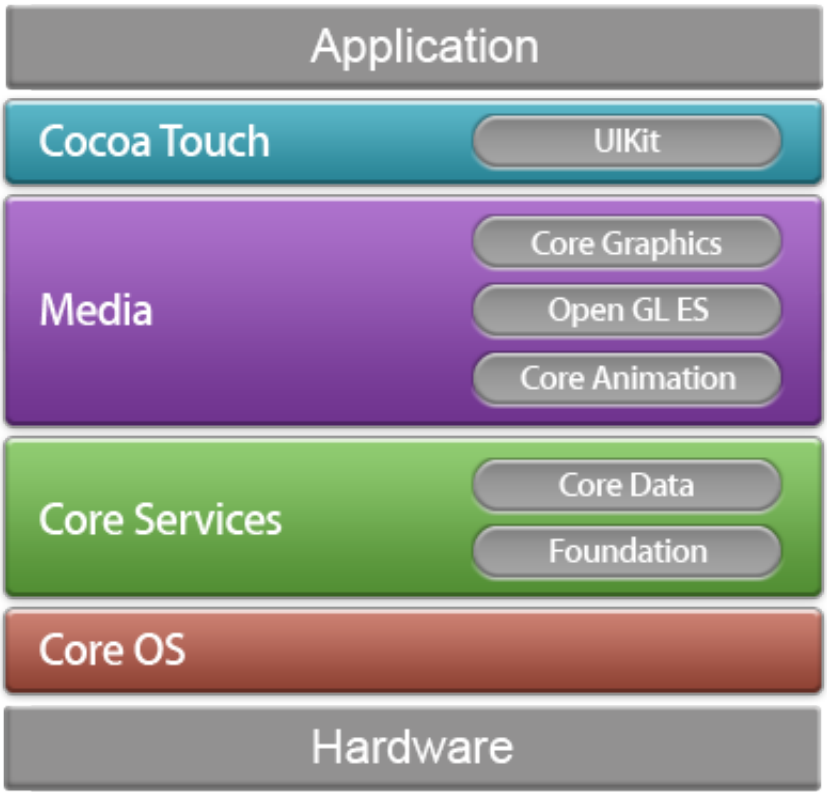

Figura 3.6: Architettura a livelli del sistema operativo iOS

libreria. Per utilizzare i frameworks è sufficiente collegarli al progetto dell'applicazione proprio come si farebbe con qualsiasi altra libreria condivisa. Collegare un framework al progetto permette di accedere alle funzioni che espone e permette agli strumenti di sviluppo sapere dove trovare i file di intestazione e le risorse contenute nel framework.

3.3.3 Cocoa Touch Layer

Cocoa Touch è un insieme di frameworks orientati agli oggetti che forniscono un ambiente base per applicazioni su iOS. Cocoa presenta due aspetti; un aspetto di esecuzione ed un aspetto di sviluppo. L'aspetto esecutivo si occupa di fornire e gestire l'interfaccia utente, come la SpringBoard, MobilePhone ed altre applicazioni. L'aspetto di sviluppo, che interessa principalmente gli sviluppatori, da il supporto alle tecnologie chiave come il multitasking, le operazioni touch, le notifiche, e molti servizi di alto livello del sistema.

3.3.3.1 Airdrop

AirDrop, introdotto con la versione 7 di iOS, permette agli utenti di condividere foto, documenti, URLs e altri tipi di dati con gli iOS Device nelle immediate vicinanze. Il supporto ad AirDrop avviene in maniera nativa utilizzando la classe *UIActivityViewController* . Questa classe permette al sistema di fornire servizi standard come il copia-incolla, la condivisione su social media, l'invio tramite sms o e-mail e molto altro ancora.

3.3.3.2 Text Kit

Text Kit è un motore di render del testo che permette di cambiare dinamicamente il comportamento del testo all'interno dell'applicazione. Ad esempio il colore o la dimensione di una parola possono essere modificati dinamicamente non appena questa viene riconosciuta dal sistema.

Il *Dynamic Type* è una tecnica che permette di regolare lo spessore del testo in funzione della sua dimensione. Rimpicciolendo una parola, questa viene automaticamente resa più leggibile tramite l'aumento dello spessore del carattere e viceversa.

3.3.3.3 UIKit Dynamics

Le classi UIKit Dynamics permettono di animare le interfacce utente con effetti grafici realistici (forza di gravità, collisioni, ecc.). Le finestre vengono animate da un motore fisico 2D nascosto allo sviluppatore, così da non richiedere nessuna conoscenza della realizzazione fisica del motore.

3.3.3.4 Multitasking

Il modello di multitasking in iOS è stato progettato per massimizzare la durata della batteria e allo stesso tempo permettere alle applicazioni di lavorare in background.

Quando l'utente preme il tasto Home all'interno di un'applicazione, questa salva il contesto attuale e viene portata in uno stato di background. Dopo pochi istanti l'applicazione viene sospesa dal sistema se non ha altro codice da eseguire, in questo modo viene preservata la batteria e aumentata la reattività del sistema. L'app rimane quindi in memoria (pronta per essere riattivata) ma senza eseguire alcun codice, in alcuni casi le applicazioni possono richiedere al sistema di eseguire codice in background per i seguenti motivi:

- Un'applicazione può richiedere uno slot finito di tempo per terminare dei task importanti.
- Un'applicazione che supporta servizi specifici(playback audio, geolocalizzazione) può richiedere slot di tempo a intervalli specifici per supportare questi servizi.
- Un'applicazione può usare le notifiche locali per generare avvisi, in determinati istanti di tempo, quando l'app non è in esecuzione.
- Un'applicazione può scaricare contenuti, periodicamente o tramite notifica, dalla rete internet.

C'è da sottolineare che il supporto al multitasking non richiede lavoro aggiuntivo da parte del programmatore, in quanto è presente nativamente in ogni applicazione. Il sistema invia delle notifiche di cambio stato all'applicazione così da permettere il salvataggio dei dati utente.

3.3.3.5 Autolayout

L'autolayout, presente dalla versione 6 di iOS, consente la costruzione dinamica delle interfacce utente, in maniera da rendere più semplice il layout

grafico su dispositivi con diverso rapporto di dimensione dello schermo. Usando l'autolayout si definiscono regole per la disposizione degli elementi nella vista dell'utente, ad esempio si può specificare la distanza di un elemento da un angolo del display, questa verrà rispettata per qualsiasi dimensione dello schermo abbia il device.

Le entità utilizzate in Autolayout sono chiamate *vincoli*, questi comportano una serie di benefici:

- Localizzazione attraverso scambio di stringhe.
- Identificazione di elementi dell'interfaccia utente per la scrittura da destra a sinistra in lingue come l'ebraico e l'arabo.
- Separazione delle responsabilità tra oggetti nei livelli delle viste e dei controller.

3.3.3.6 Storyboards

Introdotto in iOS 5 Storyboards è lo strumento raccomandato da Apple per il design dell'interfaccia utente nelle applicazioni. Storyboards permette di disegnare tutte le interfacce utente in un unico ambiente grafico, così è possibile avere una visione completa di tutte le view e i controller associati e capire come funzionano insieme.

Un'altra importante feature di storyboards è la possibilità di definire transizioni tra un controller e l'altro: queste possono essere create tramite l'editor di Xcode o definite dal programmatore tramite codice. Le view e i controller vengono raggruppati in un unico file storyboard, tuttavia è possibile utilizzare diversi file storyboard per organizzare interfacce complesse.

In fase di compilazione, Xcode prende il contenuto delle storyboard e lo divide in più parti discrete che possono essere caricate singolarmente per migliorare le prestazioni. L'applicazione non deve interfacciarsi direttamente con questi pezzi in cui è diviso lo storyboard ma ha a disposizione varie classi della libreria *UIKit* per accedere ai contenuti di uno storyboard dal codice.

3.3.3.7 Conservazione stato dell'UI

La funzione di conservazione dello stato dell'UI rende immediato il ripristino dell'interfaccia utente dell'applicazione all'ultimo stato in cui l'utente l'ha lasciata. Quando l'applicazione viene posta nello stato di background dal sistema, può salvare lo stato della sua interfaccia; l'utente quando riaprirà l'applicazione la troverà nello stesso stato in cui l'ha lasciata, dando così l'apparenza di non essere mai stata messa in background.

Il supporto per la conservazione dello stato è integrato in UIKit, che fornisce l'infrastruttura per salvare e ripristinare l'interfaccia delle applicazioni. Allo sviluppatore non è richiesto alcun onere per implementare questa funzionalità.

3.3.3.8 Servizio di notifiche push

Introdotto in iOS 3.0, il servizio di notifiche push di Apple fornisce un modo per avvisare gli utenti di nuove informazioni o eventi, quando le applicazioni non sono in esecuzione. Tramite questo servizio, è possibile visualizzare le notifiche di testo, aggiungere un badge all'icona dell'applicazione, o attivare avvisi acustici sui dispositivi degli utenti in qualsiasi momento.

Queste notifiche consentono agli utenti di sapere che un qualche evento è accaduto e che possono aprire l'applicazione relativa all'evento per ricevere le informazioni correlate. Le notifiche push richiedono una connessione internet attiva per essere ricevute, in quanto devono interfacciare il dispositivo con eventi ricevuti dal mondo esterno, come ad esempio risultati sportivi, notizie, previsioni meteo, tutti in tempo reale.

Dal punto di vista strutturale, bisogna tener conto delle seguenti operazioni per implementare il servizio di notifica push nelle applicazioni iOS. In primo luogo, l'applicazione deve richiedere la consegna delle notifiche ed elaborarne immediatamente i dati una volta consegnate. In secondo luogo, è necessario fornire un processo sul lato server per generare le notifiche. Questo processo è eseguito sul server locale e utilizza il servizio Apple Push Notification per attivare le notifiche.

3.3.3.9 Notifiche locali

Le notifiche locali, introdotte in iOS 4, completano l'attuale meccanismo di notifica push negli iOS device, mettendo a disposizione un meccanismo per la generazione delle notifiche a livello locale, senza far utilizzo di un server esterno che genera la notifica.

Le notifiche locali richiedono a priori il contenuto del messaggio e l'orario di notifica, cosa importante è che non necessitano di una connessione ad internet attiva per il loro funzionamento; il sistema si occuperà di visualizzare il messaggio di notifica, anche se l'applicazione relativa è in stato di background. Questo tipo di notifica può essere utilizzata per visualizzare messaggi, riprodurre suoni o impostare un badge di avviso nell'icona dell'applicazione.

3.3.3.10 Riconoscimento dei gesti

Il riconoscimento dei gesti è realizzato da oggetti che vengono associati alle view e servono a rilevare i gesti eseguiti dall'utente nello schermo del device. Dopo aver creato un oggetto *Gesture Recognizers* per una determinata vista, a questo vengono associate le funzioni da eseguire quando viene rilevato un particolare gesto; l'oggetto applica le euristiche definite dal sistema per riconoscere il tipo di tocco che è stato effettuato, infatti fare tutto ciò senza il supporto dei framework risulta essere molto complicato.

Tutti i tipi di gesti riconoscibili dal sistema sono basati sulla classe *UIGesture-Recognizer*, è possibile definire sottoclassi personalizzate o utilizzare una delle sottoclassi definite in UIKit per gestire qualsiasi dei seguenti gesti standard:

- Singolo touch o sequenze di due o più touch.
- Pinch to zoom (zoom in e out).
- Panning o trascinamento.
- Swipe con due o più dita in qualsiasi direzione.
- Pressione prolungata.
- Rotazioni.
- Apertura o chiusura di tutte le dita in un dato punto.

3.3.3.11 AirPrint

Introdotto in iOS 4.2, *AirPrint* consente alle applicazioni di inviare contenuti in modalità wireless a stampanti nella rete locale. UIKit fornisce il supporto per la maggior parte delle operazioni relative alla stampa. Gli oggetti della libreria forniscono le interfacce di stampa, gli strumenti per eseguire il rendering del contenuto stampabile e gestiscono la programmazione e l'esecuzione dei job di stampa sulla stampante.

I job di stampa inviati dall'applicazione verranno passati al sistema di stampa, che gestisce il processo di stampa vero e proprio. I job di stampa di tutte le applicazioni, su un dispositivo, vengono messi in coda. Gli utenti possono ottenere lo stato dei job di stampa dall'applicazione Print Center e possono anche utilizzare questa applicazione per annullare i processi di stampa. Tutti gli altri aspetti della stampa sono gestiti automaticamente dal sistema. La stampa wireless è disponibile solo su dispositivi che supportano il multitasking.

3.3.3.12 Supporto a documenti

Dalla versione iOS 5, il framework UIKit ha introdotto la classe UIDocument per la gestione dei dati associati a documenti dell'utente. Questa classe rende molto più facile l'implementazione di applicazioni basate sui documenti, in particolare le applicazioni che archiviano documenti su iCloud.

Oltre a fornire un contenitore per tutti i dati relativi al documento, la classe UIDocument fornisce il supporto incorporato per la lettura e la scrittura asincrona dei dati dei file, salvataggio sicuro dei dati, il salvataggio automatico dei dati e il supporto per la rilevazione di conflitti iCloud.

3.3.3.13 Condivisione con display esterno

La condivisione su display esterno permette ai dispositivi collegarsi ad un monitor esterno attraverso dei cavi adattatori. Quando è collegato, lo schermo associato può essere utilizzato dall'applicazione per visualizzare il contenuto. Le informazioni sullo schermo, comprese le sue risoluzioni supportate, sono accessibili attraverso le interfacce del framework UIKit. È inoltre possibile utilizzare questo framework per associare le finestre dell'applicazione con uno schermo o un altro.

3.3.3.14 Viste e controller standard

La maggior parte dei framework di sistema forniscono delle view e dei controller standard per la costruzione delle interfacce di sistema più utilizzate. Le view controller standard (del corrispondente framework) devono essere utilizzati quando si vuole implementare una delle seguenti operazioni:

- Mostrare o modificare informazioni sui contatti (Address Book Framework).
- Creare o modificare eventi nel calendario (Event Kit Framework).
- Comporre sms o e-mail (Message Framework).
- Scattare una fotografia o utilizzare una foto dal rullino utente (UIKit Framework)
- Realizzare un video clip (UIKit Framework).
- Aprire un documento (UIKit Framework).

3.3.3.15 Address Book Framework

Il framework Address Book sono delle librerie usate per creare, modificare e selezionare i contatti tramite delle view standard di sistema. Questo framework semplifica di molto il lavoro richiesto per mostrare informazioni sui contatti e assicurano la stessa interfaccia in tutte le applicazioni che ne fanno uso. Tutte le informazioni sui contatti personali sono custodite in un database centralizzato, in modo che possono essere disponibili a tutte le applicazioni che ne vogliano far uso. L'*Address Book* fornisce le seguenti funzioni:

- Accesso alle informazioni del contatto.
- Modifica alle informazioni del contatto.
- Interfaccia utente per la visualizzazione delle informazioni.
- Database protetto dove salvare le informazioni.

3.3.3.16 Event Kit Framework

Il framework Event Kit fornisce delle view controller per presentare le interfacce di sistema standard per la creazione, modifica e visualizzazione di eventi correlati all'applicazione di sistema Calendario.

3.3.3.17 Game Kit Framework

Il framework *Game Kit* viene introdotto per la prima volta in iOS 3, fornisce funzionalità di rete attraverso le quali lo sviluppatore può integrare funzionalità di networking nella propria applicazione con il minimo sforzo. Nelle release successive di iOS è stato introdotto il Game Center il quale permette agli utenti di condividere i propri risultati di gioco online, creando delle classifiche online con tutti gli utenti.
Game Center fornisce inoltre il supporto alle seguenti funzionalità, molto importanti per chi intende sviluppare giochi evoluti:

- Profilo personale: consente agli utenti di creare un proprio profilo di gioco. Gli utenti possono accedere a Game Center e interagire con gli altri utenti in maniera anonima dietro il proprio alias. I giocatori possono impostare messaggi di stato e aggiungere altri giocatori come amici, sviluppando una sorta di social network dedicato ai giochi.
- Tabella dei punteggi: consente all'utente di inviare il punteggio ottenuto durante il gioco a Game Center. Questa funzione è molto utile per creare una classifica globale di tutti gli utenti che giocano alla medesima applicazione.

- Matchmaking: consente di creare giochi multiplayer connettendo tra di loro giocatori che hanno fatto l'accesso a Game Center. Non è necessario che i giocatori siano nella stessa locazione in quanto viene creata una sessione di gioco che può sfruttare sia la rete wifi sia la rete dati cellulare.
- Obiettivi: consentono la registrazione del progresso che il giocatore ha fatto durante il gioco.
- Sfide: consentono ad un giocatore di sfidare un amico fino al raggiungimento di un obiettivo o di un determinato punteggio.
- Gioco a turni: è possibile implementare il supporto per i turni di gioco, in modo da creare partite persistenti il cui stato viene memorizzato in iCloud.

3.3.3.18 iAd Framework

Il framework *iAd*, presente da iOS 4, consente di visualizzare annunci pubblicitari all'interno dell'applicazione. Gli annunci sono integrati nelle view standard e il momento di visualizzazione viene deciso dallo sviluppatore. Il servizio viene fornito direttamente da Apple tramite l'iAd Service, che si fa carico della presentazione e della reazione al tocco degli annunci. Per lo sviluppatore è presente un portale dove vengono mostrati i ricavi e statistiche di tali annunci.

3.3.3.19 Map Kit Framework

Il *Map Kit Framework* fornisce una scroll view con all'interno una mappa geografica da integrare nella propria interfaccia utente. Le classi di questo framework forniscono inoltre, funzionalità per personalizzare il contenuto e l'aspetto della mappa, come ad esempio mostrare la posizione attuale, annotazioni, flag per indicare punti di interesse.

Questo framework integrato con l'app Mappe e gli Apple map server facilitano la creazione di applicazioni di navigazione stradale. Le applicazioni possono delegare all'applicazione Mappe, integrata nel sistema operativo, la richiesta di indicazioni stradali su un tragitto o la visualizzazione di punti d'interesse in una determinata località.

3.3.3.20 Message UI Framework

Il framework *Message UI* fornisce supporto per la composizione di SMS o email all'interno dell'applicazione, questo supporto consiste in una view con-

troller che viene presentata al momento della composizione. E' possibile compilare i campi di questa view controller per impostare il destinatario, l'oggetto, il contenuto, il corpo e gli eventuali file allegati da includere nel messaggio.
In fase conclusiva il messaggio può essere inviato al destinatario senza uscire dall'attuale applicazione. I messaggi e le mail in uscita vengono automaticamente passati al sistema che li mette nelle rispettive code di invio.

3.3.3.21 Twitter Framework

Il *Twitter* framework, presente da iOS 6, è utile per chi vuole integrare i social network nella propria applicazione, questo framework permette la realizzazione di view controller per la generazione di tweet e il supporto per l'accesso ai servizi di Twitter.

3.3.3.22 UIKit Framework

Il framework *UIKit* è il più importante framework di iOS essendo responsabile di tutte le funzioni dell'interfaccia utente, dalla creazione delle finestre ai minimi componenti dell'interfaccia fino alla gestione degli input. Gran parte del successo della piattaforma iOS è dovuta a questo framework in quanto fornisce elementi per creare interfacce di ogni tipo in maniera ordinata e coerente tra applicazione e applicazione, semplificando l'uso della piattaforma in generale. Quando un'applicazione viene avviata, la sua funzione main() istanzia un oggetto di tipo UIApplication, che è la classe base per tutte le applicazioni iOS dotate di interfaccia grafica e che provvede l'accesso a tutte le funzioni di alto livello viste fino ad ora e alle funzioni di controllo. UIKit gestisce e fornisce supporto a molti elementi del sistema, di seguito i più importanti:

- Infrastruttura e gestione base dell'applicazione, incluso il ciclo main().
- Gestione dell'interfaccia utente incluso il supporto alle Storyboards.
- Oggetti che rappresentano viste e controlli standard come le *finestre* e le *view*. Una finestra rappresenta uno spazio geometrico sullo schermo mentre una view rappresenta un contenitore per altri oggetti, all'interno la finestra può contenere una sola view, mentre una view può contenere una o più subview. Questi due elementi sono usati insieme per visualizzare gli elementi sullo schermo.
- Text View e Image view: sono semplici classi derivate dalla classe UIView che permettono di visualizzare rispettivamente testo e immagini.

- Transizioni: le transizioni permettono di visualizzare il passaggio da una finestra all'altra tramite un'animazione messa a disposizione dal framework stesso.

- Gestione della *status bar*: la barra di stato è un elemento parte del sistema operativo che compare nella parte superiore dello schermo e che visualizza informazioni di sistema come l'ora, la potenza del segnale, il consumo della batteria.

- Alert sheets: sono piccole finestre di allerta che compaiono al centro dello schermo per avvisare l'utente di un qualche evento o per chiedere conferma di qualche operazione.

- Le *tabelle*, sono oggetti che permettono la visualizzazione di collezioni di dati. Sono molto usate per gestire dati dai database, visualizzare in maniera ordinata file xml e json; sono uno strumento molto flessibile che permette allo sviluppatore di personalizzare ogni cella e definirne il comportamento.

- Navigation Bar: la barra di navigazione permette all'utente di passare da una view all'altra, seguendo il flusso dell'interfaccia utente definito dal programmatore. In questo modo l'utente visualizza le varie schermate come fossero pagine di un libro e utilizza la barra di navigazione per cambiare view oppure per attivare particolari funzioni della view corrente.

- Supporto per la condivisione di contenuti tramite email, Twitter Facebook e altri servizi.

- Informazioni sul device, sul sensore di prossimità, sulla fotocamera e gestione del controllo remoto tramite auricolare.

Nella Figura 3.7 viene presentata la gerarchia delle classi di UIKit. Come si può vedere la classe root di tutti gli oggetti è la *NSObject*, da cui poi derivano tutte le altre classi del framework.
Ad esempio, si può osservare come tutti gli elementi che compongono l'interfaccia grafica di un'applicazione (UIImageView, UILabel, UIWindow ecc.) derivano tutte dalla classe *UIView*, a sua volta questa eredita dalla classe *UIResponder* che infine risponde alla classe root *NSObject*.

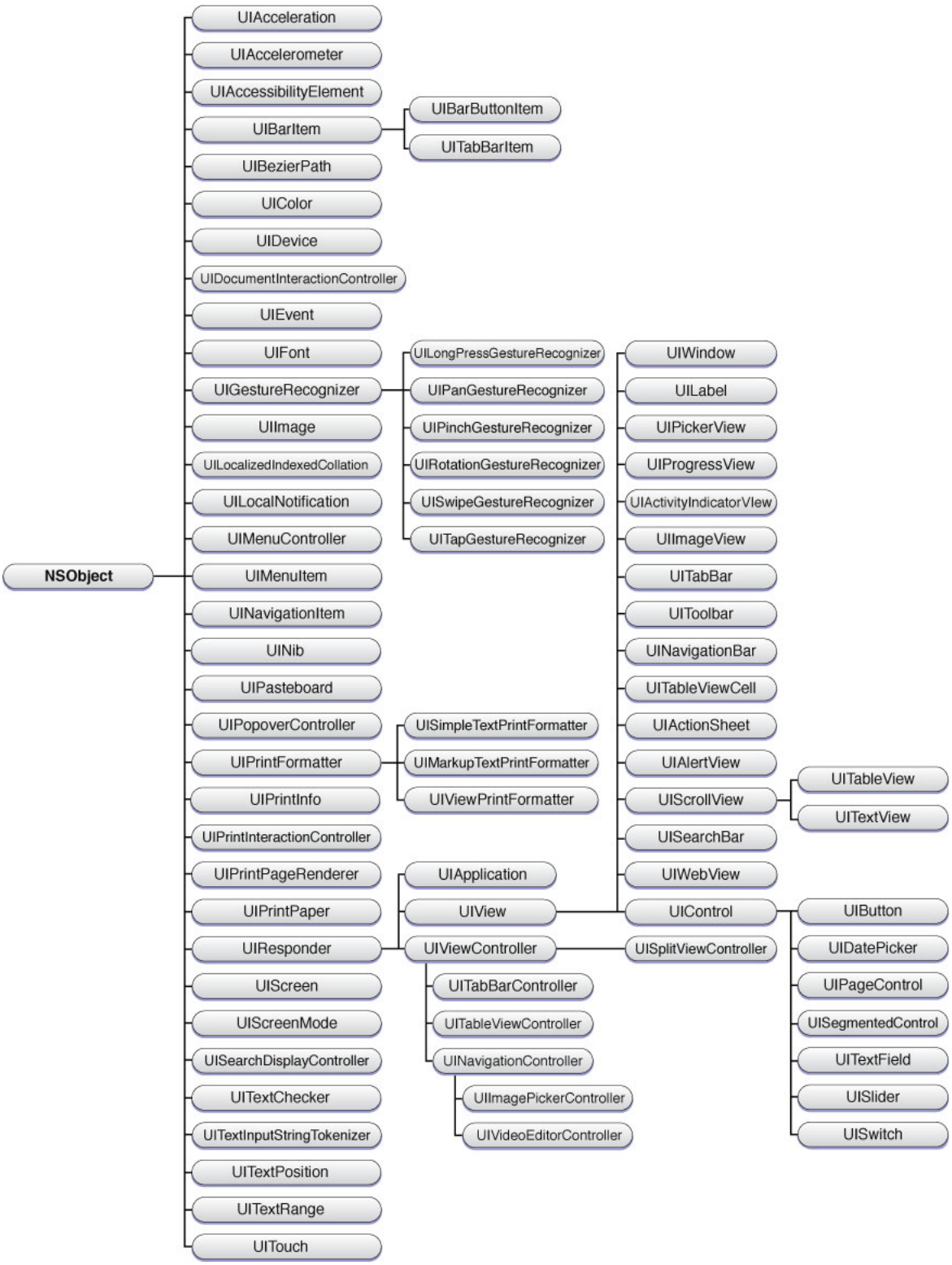

Figura 3.7: Gerarchia delle classi in UIKit

3.3.4 Media Layer

Il ruolo di tale livello e quello di fornire supporto alle tecnologie multimediali, ovvero audio, video, animazioni e grafica con il fine di creare la migliore esperienza multimediale possibile su un dispositivo mobile. Le tecnologie di questo livello sono state progettate per rendere più facile la creazione e la gestione degli elementi multimediali all'interno delle applicazioni.

3.3.4.1 Audio

Le tecnologie audio messe a disposizione in iOS sono state progettate per fornire la migliore esperienza audio per gli utenti. Questa esperienza include la possibilità di riprodurre e registrare audio di alta qualità e di attivare la funzione di vibrazione sui dispositivi che ne sono dotati. Il sistema fornisce diversi modi per riprodurre e registrare contenuti audio.
I framework che compongono il *Core Audio* sono rappresentati nella Figura 3.8, sono ordinati per livello dal più alto al più basso. Durante la fase di sviluppo è importante ricordare che i framework di livello più alto sono più semplici da usare perché mettono a disposizione funzioni di alto livello, mentre quelli di livello inferiore offrono maggiore flessibilità e controllo, ma richiedono più lavoro. Di seguito verrà data una breve descrizione dei framework inclusi in Core Audio.

I *low-level Services* includono:

1. L'*I/O Kit* che interagisce con i driver audio.
2. Il livello di *audio hardware abstraction* (audio HAL), che fornisce delle interfacce indipendenti dall'hardware e dai driver di ogni singolo device.
3. Il *Core MIDI*, fornisce un livello di astrazione software per manipolare e lavorare con flussi MIDI
4. *Servizi di temporizzazione*, forniscono accesso al clock di sistema.

I *mid-level services* includono servizi per conversione di formato audio, lettura e scrittura su disco di flussi audio, in particolare:

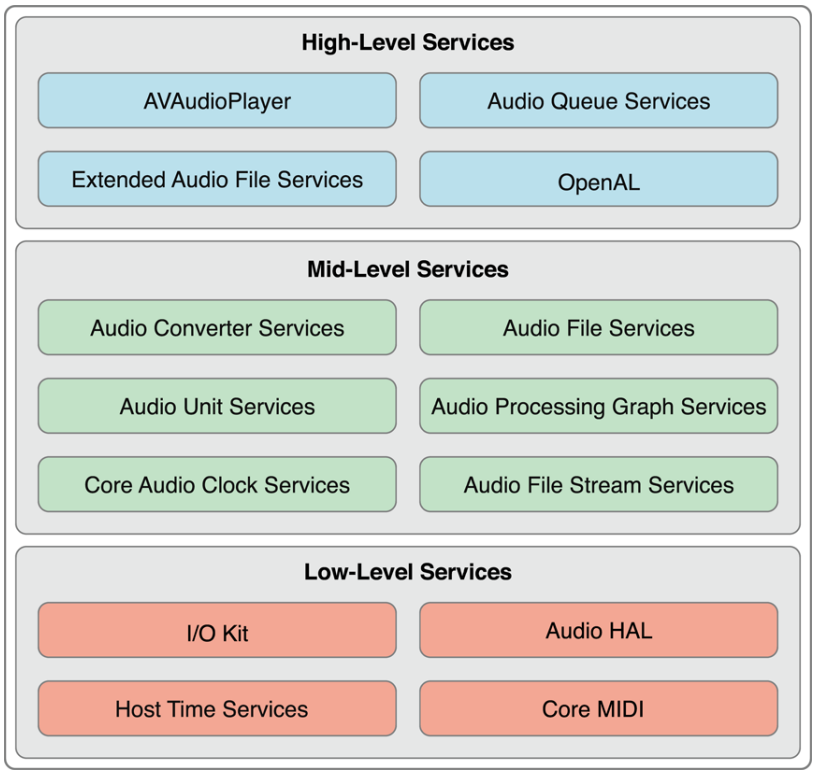

Figura 3.8: Core Audio

- Servizi di conversione audio, permettono all'applicazione di utilizzare convertitori audio e quindi di funzionare con differenti formati.
- Servizi per file audio, permettono la lettura e scrittura di dati audio da e per il disco fisso del device.
- *Audio Unit Services* and *Audio Processing Graph Services*, permettono alle applicazioni di lavorare con plugin DSP (digital signal processing come mixer ed equalizzatori.
- Servizi di *file streaming*, permettono alle applicazioni di analizzare flussi audio, per creare servizi di streaming audio attraverso la connessione di rete.
- *Core Audio Clock Services* per sincronizzare audio e MIDI nel migliore dei modi durante le conversioni time-based.
- *Servizi di assistenza*, una piccola API, che aiuta la gestione dei vari formati audio all'interno dell'applicazione.

I *high-level services* forniscono interfacce semplificate che combinano caratteristiche degli strati inferiori.

- *Audio Queue Services* consentono di registrare, riprodurre, mettere in pausa, in loop e la sincronizzazione audio. Utilizzano autonomamente tutti i codec necessari per gestire i formati audio compressi.
- La classe *AVAudioPlayer* fornisce delle semplici interfacce per riprodurre audio con funzioni di loop, riavvolgimento e avanzamento veloce.
- *Servizi file audio estesi*, combina funzionalità provenienti da Audio File Services and Audio Converter services. Predispone un'unica interfaccia per leggere e scrivere suoni audio compressi e non.
- *OpenAL* è una implementazione, interna in Core Audio, della libreria open-source OpenAL per l'audio posizionale. Riceve il supporto del sistema grazie ad un mixer 3D, questa funzionalità viene maggiormente utilizzata durante lo sviluppo di videogames.

Infine è bene inoltre ricordare quali formati audio sono supportati dal sistema operativo, nel caso di iOS sono i seguenti:

- AAC

- Apple Lossless (ALAC)
- A-law
- IMA / ADPCM (IMA4)
- PCM lineare
- Microsoft GSM 6.10
- AES3-2003
- DVI / Intel ADPCM IMA

3.3.4.2 Grafica

La grafica di alta qualità è una parte molto importante delle applicazioni iOS. La più semplice e efficiente via per creare la grafica è quello di utilizzare immagini pre-renderizzate con le view e i controlli standard forniti in UIKit e lasciare al sistema il compito di disegnarle. Esistono tuttavia situazioni in cui è necessario scendere ad un livello più basso per la creazione della grafica; in questi casi è possibile utilizzare le seguenti tecnologie per la gestione dei contenuti grafici:

1. *Core Graphics.* Il framework Core Graphics (noto anche come Quartz) gestisce il rendering nativo vettoriale 2D e basato sulle immagini attraverso le API di disegno 2D Quartz. Quartz è lo stesso motore di disegno vettoriale che viene utilizzato in OS X.

   Core Graphics può essere usato per:

   - Disegni basati su percorso.
   - Rendering con anti-aliasing.
   - Gradienti, immagini e colori.
   - Trasformazioni tra coordinate spaziali.
   - Creazione, visualizzazione e parsing di documenti PDF.

2. *OpenGL ES.* L'Open Graphics Library Embedded System (OpenGL ES) è una libreria open-standard grafica multiuso dedicata alla creazione di contenuti digitali 2D e 3D, alla progettazione meccanica e architettonica, alla prototipizzazione virtuale, alla simulazione di volo, ai videogiochi e ad altro ancora. L'implementazione di Apple di questa

libreria, lavora a stretto contatto con l'hardware del dispositivo, per fornire un frame rate elevato nelle applicazioni di gioco full-screen.

In particolare OpenGL ES viene usato per:

- Creare grafica 2D e 3D.
- Creare oggetti grafici complessi, come visualizzazione di dati, simulazione di volo, o video giochi.
- Accedere all'hardware grafico.

3. *Core Animation.* Questo framework permette di creare animazioni avanzate ed effetti grafici. UIKit stesso fornisce animazioni che sono costruite usando la tecnologia Core Animation, ma se si necessità di animazioni complesse, è possibile utilizzare direttamente Core Animation. Le interfacce di Core Animation fanno parte del pacchetto QuartzCore, si può creare una gerarchia di oggetti di livello, che possono essere manipolati, ruotati, scalati, trasformati e molto altro.

   Core Animation è usato per:

   - Creare animazioni personalizzate.
   - Aggiungere funzioni di temporizzazione alla grafica.
   - Specificare i vincoli di layout grafici.
   - Supportare l'animazione per fotogramma.
   - Aggiornare più livelli grafici in un solo update atomico.

   Infine Core Animation è molto utile per realizzare interfacce utente dinamiche, senza che le prestazioni dell'applicazione ne risentano, come accadrebbe utilizzando API grafiche di basso livello come OpenGL EL.

4. *Core Image.* Permette la gestione avanzata di video ed immagini. Vengono messi inoltre a disposizione filtri per operazioni semplici quali ad esempio ritocco e correzione di foto oppure per la gestione di operazioni più avanzate come il riconoscimento facciale e delle features. Queste operazioni non modificano mai le immagini originali, ma lavorano su una copia di esse, rendendo sempre disponibile l'immagine iniziale. La velocità e l'efficienza di tali operazioni è dovuta all'utilizzo della CPU disponibile e della potenza di elaborazione della GPU da parte del framework.

5. *Image I/O.* Fornisce interfacce per la lettura e la scrittura della maggior parte dei formati di immagini esistenti. Inizialmente Image I/O faceva parte del framework Core Graphics.

6. *Core Text.* Viene utilizzato per la gestione del layout ed il rendering di testo all'interno dell'applicazione. Il motore di layout di Core Text è stato pensato per effettuare le operazioni di disposizione del testo in maniera semplice.

7. *Assets Library.* Il framework Assets Library consente di accedere alle foto e video nella libreria fotografica del device. La Assets Library fornisce un sistema basato su query per il recupero di foto e video dal dispositivo dell'utente, in quanto le immagini e video sono gestiti da un vero e proprio database. È inoltre possibile salvare nuove foto e video all'interno della libreria fotografica.

3.3.4.3 Video

iOS fornisce diverse tecnologie per riprodurre i contenuti video. Sui dispositivi con l'hardware appropriato, è possibile utilizzare queste tecnologie per registrare video e incorporarli all'interno delle applicazioni.

Il sistema fornisce diversi metodi per riprodurre e registrare contenuti video che si possono scegliere a seconda delle esigenze. Quando si sceglie una tecnologia video, è bene ricordare che i frameworks di livello superiore possono semplificare notevolmente il lavoro da fare per supportare le più comuni funzionalità di cui si ha bisogno.

I seguenti frameworks (ordinati dal livello più alto al più basso) sono utili per inserire funzionalità video nelle proprie applicazioni:

- Il framework Media Player fornisce una serie di interfacce semplici da usare per la riproduzione di filmati a schermo intero o in finestre.

- L'AV Foundation framework fornisce un insieme di interfacce Objective-C per la gestione della registrazione e riproduzione di filmati.

- Core Media descrive i tipi di dati di basso livello utilizzati dalle strutture di livello superiore e fornisce interfacce di basso livello per la manipolazione dei contenuti multimediali.

3.3.5 Core Services Layer

Il *Core Services Layer* contiene i servizi di sistema fondamentali per le applicazioni. I servizi più importanti tra questi sono il Core Data e il Foundation Framework, che definiscono i tipi di base che tutte le applicazioni utilizzano. Questo livello contiene inoltre singole tecnologie per supportare funzionalità quali la localizzazione, iCloud, i social media, e il networking.

Di seguito verranno descritti i due framework principali:

Foundation Framework Il *Foundation* framework fornisce servizi base di sistema per tutte le applicazioni, comprese quelle native. Viene utilizzato svariati casi:

- Creare e gestire collezioni di dati, come array e dizionari.
- Accedere ad immagini e altre risorse che risiedono nella propria applicazione.
- Generare e ricevere notifiche.
- Trovare automaticamente altri dispositivi nella rete circostante.
- Manipolare flussi di dati.
- Eseguire codice asincrono.

Core Data Il framework Core Data è una tecnologia per la gestione del modello di dati di una applicazione Model-View-Controller. L'uso di Core Data è consigliabile in applicazioni in cui il modello di dati è già altamente strutturato. Invece di definire strutture dati tramite codice, è possibile utilizzare gli strumenti grafici in Xcode per costruire uno schema che rappresenta il modello dei dati. In fase di esecuzione, le istanze delle entità del modello di dati vengono create, gestite e rese disponibili attraverso il framework così da ridurre notevolmente la quantità di codice da scrivere. Core Data sfrutta la tecnologia incorporata in SQLite per memorizzare e gestire i dati in modo efficiente.

Si può usare Core data per:

- Salvare e recuperare gli oggetti dalla memoria.
- Supporto per il copia/incolla.
- Convalidare automaticamente i valori delle *property*.
- Filtrare, raggruppare e organizzare i dati in memoria.

- Gestire i risultati in una tabella con [NSFetchedResultsController].
- Supportare applicazioni basate su documenti.

3.3.6 Core OS Layer

Il livello Core OS contiene funzionalità di basso livello con cui sono costruite la maggior parte delle tecnologie in iOS. Anche se non si utilizzano direttamente queste tecnologie è molto probabile che vi siano altri framework che le utilizzano in maniera indiretta, senza esserne a conoscenza. Ad esempio, in situazioni in cui è necessario affrontare problemi relativi alla sicurezza o alle comunicazioni con un accessorio esterno, lo si fa utilizzando i framework di questo livello.
Di seguito le funzionalità messe a disposizione dal livello Core OS:

- *Accelerate Framework*: Il framework Accelerate contiene interfacce per l'elaborazione di segnali digitali (DSP), algebra lineare ed elaborazione delle immagini. Scrivendo le proprie versioni di queste interfacce, queste vengono ottimizzate per tutte le configurazioni di iOS disponibili. Pertanto, è possibile scrivere il codice una volta ed essere certi che venga eseguito in modo efficiente su tutti i dispositivi.
- *Core Bluetooth Framework*: Il Core Bluetooth consente agli sviluppatori di interagire specificamente con accessori Bluetooth a basso consumo energetico (LE). Le interfacce Objective-C di questo framework consentono di eseguire le seguenti operazioni:
  - Scansione di accessori Bluetooth, connessione e disconnessione da essi.
  - Creare servizi dall'applicazione, mettendole a disposizione a dispositivi esterni, ad esempio utilizzare il device come mouse o come telecomando.
  - Trasmettere informazioni iBeacon dal dispositivo iOS.
  - Preservare lo stato delle connessioni Bluetooth e ristabilire le connessioni quando l'applicazione viene successivamente lanciata.
  - Essere informati delle modifiche alla disponibilità di periferiche Bluetooth

- *External Accessory Framework*: fornisce il supporto per la comunicazione con gli accessori hardware esterni collegati ad un dispositivo iOS-based. Gli accessori possono essere collegati tramite il connettore dock presente nel dispositivo o in modalità wireless tramite Bluetooth. Il framework fornisce le classi per ottenere informazioni su ogni accessorio disponibile e permette di avviare sessioni di comunicazione. Successivamente, si è liberi di manipolare l'accessorio utilizzando direttamente le API di supporto.

- *Generic Security Services Framework*: fornisce un insieme di servizi standard legati alla sicurezza delle applicazioni iOS. Le interfacce di base di questo framework sono specificate in IETF RFC 2743 e RFC 4401. Oltre ad offrire le interfacce standard, iOS include alcune aggiunte per la gestione delle credenziali non specificate dalla norma, ma che sono richieste da molte applicazioni.

- *Security Framework*: oltre alla sicurezza base in già inserita in iOS, è possibile utilizzare questo framework per garantire la sicurezza dei dati che la vostra applicazione gestisce. Questo framework fornisce interfacce per la gestione dei certificati, chiavi pubbliche e private. Esso supporta la generazione di numeri pseudocasuali crittografici sicuri. Supporta la memorizzazione di certificati e chiavi crittografiche nel Keychain, un archivio sicuro per i dati sensibili degli utenti.

  La libreria comune Crypto prevede un ulteriore supporto per la crittografia simmetrica, per codici di autenticazione dei messaggi basati su hash (HMAC) e digest. La funzionalità digest fornisce funzioni che sono compatibili con la libreria OpenSSL, la quale non è disponibile in iOS .

  E' possibile condividere elementi del Keychain tra più applicazioni create. La condivisione di elementi, come le password, rende più facile l'interazione tra applicazioni della stessa suite. Ad esempio, è possibile condividere le password dell'utente senza richiedere nuovamente all'utente l'inserimento della password in ogni applicazione.

- *System*: Il livello di sistema comprende l'ambiente kernel, driver e interfacce UNIX a basso livello del sistema operativo. Il kernel stesso, sulla base di Mach, è responsabile di ogni aspetto del sistema operativo. Gestisce il sistema di memoria virtuale, threads, file di sistema, di rete, e la comunicazione tra processi. I driver a questo livello forniscono l'interfaccia tra l'hardware a disposizione e framework sistema. Per

motivi di sicurezza, l'accesso al kernel e driver è limitato a un numero limitato di framework e applicazioni di sistema.

iOS fornisce un insieme di interfacce per accedere a molte caratteristiche di basso livello del sistema operativo. Le interfacce sono c-based e forniscono il supporto per le seguenti funzionalità:

- Concorrenza (thread POSIX e Grand Central Dispatch).
- Networking socket BSD.
- Accesso ai file-system.
- Standard I/O.
- Protocollo Bonjour e servizi DNS.
- Informazioni locali.
- Allocazione della memoria.
- Calcoli matematici.

- *64-bit Support*: iOS è stato inizialmente progettato per supportare i file binari su dispositivi che utilizzano una architettura a 32 bit. In iOS 7, invece, è stato introdotto il supporto per la compilazione, linking, e il debug di file binari su un architettura a 64 bit. Tutte le librerie di sistema e le strutture sono a 64 bit ready, il che significa che possono essere utilizzate in applicazioni sia a 32-bit e 64-bit. Se compilate per il runtime a 64 bit, le applicazioni possono funzionare più velocemente, visto che il sistema dispone di risorse di processore extra in modalità 64 bit.

3.4 Sparrow Framework

Il framework **Sparrow** è costituito da un insieme di classi che consentono di creare applicazioni multimediali interattive 2D e giochi 2D per la piattaforma iOS di Apple. Di seguito le caratteristiche principali.

Free e Open Source Sparrow è free, quindi non ha costi di licenza d'uso per lo sviluppatore; è un software Open Source, quindi il codice è messo a disposizione della community e ogni sviluppatore può apportare le proprie personali modifiche. Grazie a queste due caratteristiche si è sviluppata una grande community attorno a questo framework che, avendo il codice sorgente a disposizione, può aiutare gli sviluppatori a migliorarlo e a correggerne i bug. Infine utilizzando Sparrow per il proprio progetto è possibile avvalersi di un grande numero di estensioni sviluppate dalla comunità, che spesso semplificano operazioni comuni e ripetitive. In alcuni casi queste estensioni vengono integrate nelle versioni successive del framework.

Objective-C Based Sparrow è una libreria Objective-C che è stata costruita partendo da zero per essere compatibile con iPhone, iPad e iPod Touch. Si può facilmente integrare con le applicazioni UIKit esistenti, accedere direttamente a tutte le API di iOS (come GameCenter, iAd, fotocamera, ecc.) e di beneficiare delle prestazioni native del sistema. Si esce da un sistema black-box[1] per utilizzare in sistema completamente integrato.

Gerarchia di visualizzazione ad albero In Sparrow è stata creata una struttura gerarchica che permette di raggruppare gli oggetti insieme, in maniera da gestirli nella maniera più efficiente possibile. Questa struttura prende il nome di *display-tree* (vedi Figura 3.9): ogni oggetto, visibile o che interagisce con l'utente, deve far parte di un albero gerarchico con relazioni padre-figlio. Il display-tree è molto utile per il sistema che gestisce gli eventi che vedremo di seguito.

Sistema di Eventi Sparrow mette a disposizione sistema per gestire gli eventi, sfruttando le potenzialità della struttura display-tree. Qualsiasi oggetto di visualizzazione all'interno della struttura ad albero può generare eventi. Una volta che gli eventi vengono lanciati da un oggetto, si muovono verso il basso lungo i rami del display tree e visitano ogni nodo padre fino a raggiungere il nodo root dell'albero. Gli eventi comuni, come il tocco sullo schermo o l'entrare nel frame di un oggetto, sono già implementati nel framework e sono pronti per essere utilizzati; tuttavia si possono creare eventi personalizzati a seconda delle esigenze. Nella Figura 3.10 si vede come un oggetto foglia generi un evento e come questo venga propagato fino al nodo root.

[1] In sistema black-box non è noto a priori, né ciò che contiene né come si comporta. È possibile studiarne il comportamento esclusivamente analizzando le risposte che esso produce a fronte delle sollecitazioni che riceve.

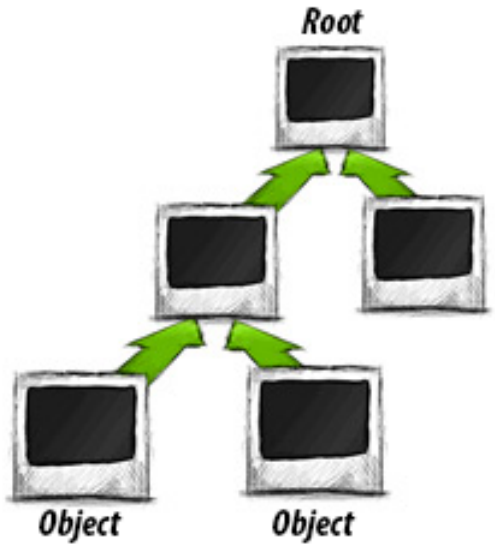

Figura 3.9: Gerarchia di visualizzazione ad albero

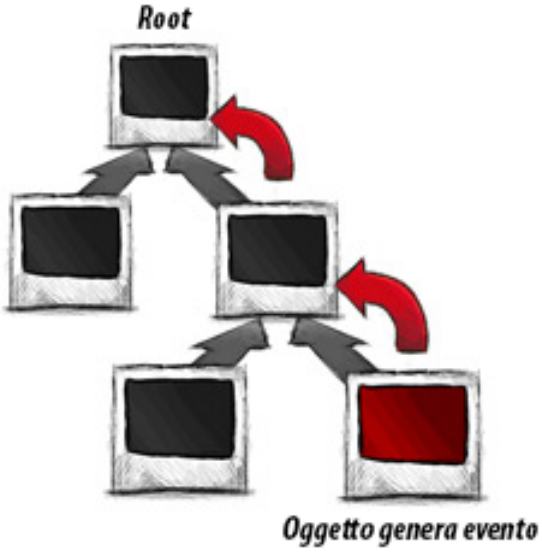

Figura 3.10: Sistema Eventi

Tween Un *Tween* è un oggetto in grado di creare una transizione di una qualsiasi proprietà numerica dell'oggetto, portando tale proprietà da un valore iniziale ad un valore finale in un determinato intervallo di tempo prestabilito; ad ogni tween si possono associare differenti curve di transizione.

Sparrow contiene un proprio sistema per la gestione dei Tween. Questo include l'oggetto Tween, una classe leggera che si occupa solo del progresso della transizione e della relativa proprietà che gestisce, e l'oggetto *Juggler* che gestisce il tween a seconda degli intervalli di tempo con i quali avverrà la visualizzazione. La combinazione di questi due strumenti permette di creare animazioni in maniera più semplice rispetto al metodo offerto dai framework grafici di iOS.

Multitouch Sparrow determina gli eventi touch in maniera semplice e intuitiva, associandoli al corrispettivo oggetto visualizzato a schermo. Il Multitouch è integrato direttamente nella libreria.

3.5 Motore di fisica 2D

Un motore fisico è un pacchetto software che simula un modello fisico newtoniano utilizzando variabili come massa, velocità, frizione alla resistenza del vento e molto altro ancora. Il motore, utilizzando questi dati e le leggi newtoniane, simula il comportamento degli oggetti sottoposti alle forze del mondo (reale o immaginario), in particolare, un motore fisico 2D simula tutti questi elementi in un ambiente in due dimensioni. Questi software trovano applicazione nell'ambito della computer grafica, cinema e videogiochi.

Ci sono due tipologie di motori fisici: quelli *real-time* e quelli ad alta precisione. I motori fisici ad alta precisione richiedono maggiore potenza di elaborazione per calcolare la fisica, di solito vengono utilizzati da scienziati e nel campo cinematografico. Invece, i motori fisica *real-time* sono maggiormente utilizzati nel campo videoludico; in questi motori fisici si effettuano calcoli semplificati con una conseguente diminuzione della precisione, in questo modo si predilige la velocità di computazione che deve essere adeguata al gameplay del gioco.

I motori fisici di questo utilizzati nei videogame hanno due caratteristiche fondamentali: un sistema di *collision detection* e una componente di simulazione della dinamica dei corpi, responsabile della risoluzione delle forze che influenzano gli oggetti simulati. I motori fisici più complessi possono con-

tenere anche la simulazioni dei fluidi, sistemi di controllo di animazione e strumenti di integrazione delle attività.
Le soluzioni già presenti nel mercato sono varie e sono da selezionare in base alle esigenze. In questo progetto specifico è necessario che il motore fisico sia:

- Gratuito e open-source.
- Adatto allo sviluppo su applicazioni mobile.
- Bidimensionale, visto che Sparrow supporta lo sviluppo di applicazioni bidimensionali.

I motori fisici nel mercato che rispettano queste esigenze sono i seguenti:

- Box 2D
- Chipmunk Phyics
- Farseer Physics Engine
- Physics2D

La soluzione che meglio si integra nel framework APP-KID è Box 2D, inoltre questo software è già utilizzato in precedenza per lo sviluppo di altre applicazioni da parte di Ware's Me. Insieme a Box2D è stata utilizzata *SPPhysics*, una libreria di integrazione sviluppata da Ware's Me per poter utilizzare al meglio le funzionalità di Box 2D nel framework APP-KID.
Di seguito verrà fatta una breve descrizione di Box2D.

Box 2D Box 2D è un motore fisico per la simulazione di corpi rigidi a due dimensioni, sviluppato da Erin Catto. E' stato scritto inizialmente in C++ e successivamente sono stati fatti i porting per piattaforme come Java (JBox2D), Flash (Box2DFlash), Nintendo DS, C-sharp XNA, Delphi e Javascript. La tecnologia è totalmente open-source ed è disponibile la documentazione, questo è stato uno dei motivi chiave della sua scelta. Inoltre grazie alla natura free del software è nata una grande community di sviluppatori che aiutano lo sviluppo del progetto, aggiungendo funzionalità e correggendo bug. Un esempio di successo per questa libreria è il famoso gioco Angry Birds sviluppato da Rovio, che ha utilizzato il motore Box2D per il suo sviluppo.

Capitolo 4

Progettazione e sviluppo

4.1 Analisi Framework Sparrow

Sparrow è basato principalmente su due classi di Cocoa Touch: NSObject e UIView. NSObject come detto in precedenza, è la classe radice di Objective-C, gli oggetti ne ereditano una interfaccia base per il sistema di runtime e la capacità di comportarsi come oggetti Objective-C.

La classe UIView definisce un'area rettangolare sullo schermo e fornisce le interfacce per gestire il contenuto di quell'area. Bisogna sottolineare che la classe UIView mette in collegamento il sistema di visualizzazione nativo di iOS e la gestione dei contenuti di Sparrow, infatti la classe SPView di Sparrow deriva direttamente dalla classe UIView.

4.1.1 Classe contenitore: SPView

Un oggetto SPView viene utilizzato da Sparrow per renderizzare al suo interno tutti gli oggetti che compongono la schermata dell'applicazione.

Per avviare l'esecuzione di una applicazione che utilizza Sparrow è sufficiente collegare questa classe con la sottoclasse di SPStage (che definisce le caratteristiche della scena) e chiamare il metodo *-(void)start*. Quando l'applicazione termina è necessario chiamare il metodo *-(void)stop*.

Le altre classi di Sparrow, che derivano da NSObject, servono a caricare, gestire e utilizzare tutti i contenuti e gli eventi che andranno a costituire l'applicazione vera e propria, che verrà solo alla fine visualizzata tramite la classe SPView.

La libreria è suddivisa, come si può vedere dalla Figura 4.1, in 6 package principali che andremo a descrivere di seguito.

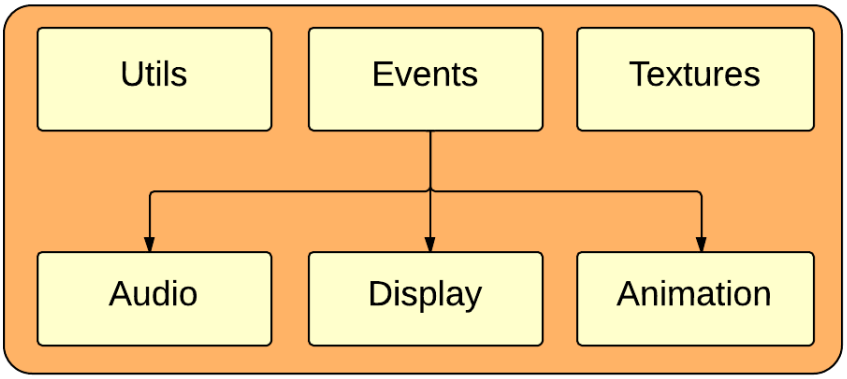

Figura 4.1: Pacchetti della libreria Sparrow

4.1.2 Display

Come si può notare dalla Figura 4.2, la classe root degli oggetti visualizzabili a schermo è la *SPDisplayObject*. Come detto in precedenza, gli oggetti visualizzabili sono organizzati in una struttura con logica ad albero detta display tree.

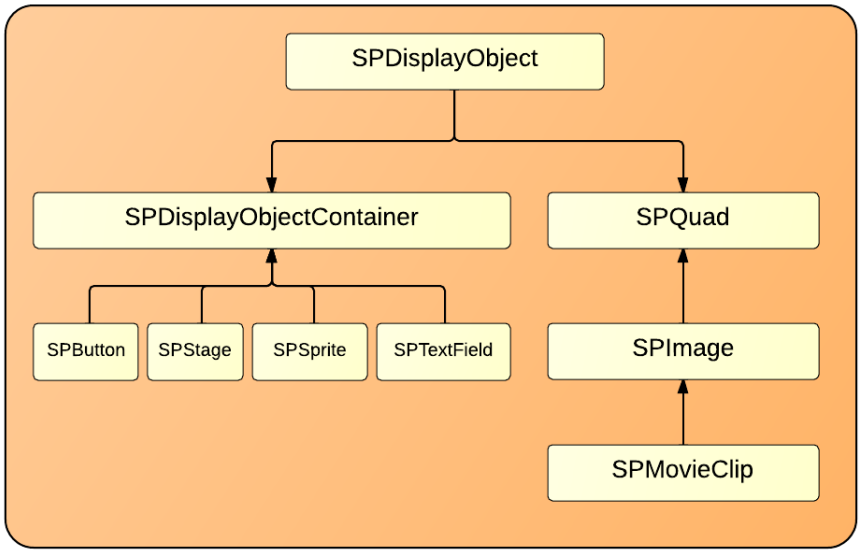

Figura 4.2: Gerarchia del pacchetto Display Object

Il display tree è composto da nodi come SPImage e SPQuad, che vengono utilizzati per visualizzare oggetti sullo schermo, e da *nodi container* (SPButton, SPSprite ecc.), sottoclassi di *SPDisplayObjectContainer*. Un nodo container è un oggetto contenitore, ovvero può contenere nodi di visualizzazione o altri

nodi container. Con questo meccanismo è possibile creare gerarchie di oggetti e far generare eventi ad oggetti appartenenti ad una determinata gerarchia.

Alla root del display tree vi è un oggetto container che deriva da SPStage, da questo oggetto si inizia la creazione del proprio albero di visualizzazione per creare l'applicazione Sparrow.

Andiamo ora ad analizzare le classi principale di questo package.

SPDisplayObject SPDisplayObject è una classe astratta[1] che fornisce metodi e proprietà che tutti gli oggetti di visualizzazione condividono:

- Nome.
- Coordinate x,y di posizione.
- Pivot x,y: punto di rotazione dell'oggetto.
- Dimensione (width, height).
- Fattore di scala orizzontale e verticale (scaleX, scaleY).
- Bounds dell'oggetto relativi al sistema di coordinate dell'oggetto contenitore.
- Rotazione dell'oggetto.
- Opacità (alpha).
- *Touchable*, indica se l'oggetto e i suoi figli possono ricevere eventi touch.
- Visibilità.
- Oggetto genitore (parent).
- Oggetto radice (root).

Una delle proprietà più importanti è la proprietà *Touchable* che permette di riconoscere eventi di tocco sull'oggetto e i suoi figli; quando questa è disattivata, gli oggetti non ricevono eventi touch e si possono definire statici dal lato dell'utente. L'unità di misura per le coordinate e le distanze è il punto, che è diverso dal pixel. La dimensione del punto rispetto al pixel

[1] Una classe astratta definisce una interfaccia senza implementarla completamente. Ciò serve come base di partenza per generare una o più classi specializzate, aventi tutte la stessa interfaccia di base.

dipende dal *contentScaleFactor*. Di default un punto equivale ad un pixel nei dispositivi con display a bassa risoluzione (non Retina Display), mentre nei display ad alta risoluzione (Retina Display) un punto equivale a 2 pixel.

La rotazione di un oggetto è sempre eseguita rispetto al punto di pivot *(pivotX, pivotY)*. Per default, il punto di pivot è *(0,0)* e definisce anche l'origine del sistema delle coordinate dell'oggetto. C'è da considerare inoltre che ogni oggetto ha un proprio sistema di coordinate locali, se si ruota un contenitore, si ruota tale sistema di coordinate e quindi anche tutti gli oggetti all'interno del contenitore ruotano con esso.
Oltre ad utilizzare le sottoclassi già esistenti di *SPDisplayObject*, è possibile creare sottoclassi personalizzate utilizzando i metodi *render support* e *boundInSpace*, questo è utile per creare oggetti con con comportamenti particolari.

SPQuad La classe *SPQuad* rappresenta un rettangolo con un colore uniforme o una sfumatura di colore. È possibile impostare un colore per ogni vertice, I colori sfumeranno automaticamente nell'area di rettangolo. Ad esempio per visualizzare una semplice sfumatura di colore lineare, basta assegnare un colore ai vertici 0 e 1 e un altro colore per vertici 2 e 3. Ad un oggetto SPQuad è possibile sovrapporre una texture, questa proprietà ne fa una delle classi maggiormente utilizzate in Sparrow.

SPImage La classe *SPImage* eredita direttamente da SPQuad; una SPImage visualizza un quad con una texture mappata su di esso, per far ciò è sufficiente utilizzare la classe SPTexture per rappresentare la texture, successivamente la texture viene mappata in un SPQuad.

In una SPImage è possibile assegnare un colore, quando questo viene assegnato, per ogni pixel il colore risultante è la moltiplicazione del colore della texture con il colore del quad. In questo modo si può colorare una texture di un determinato colore.

SPStage SPStage è la classe è la radice dell'albero di visualizzazione. Uno stage rappresenta l'area di rendering dell'applicazione, la larghezza e l'altezza dello stage definiscono il sistema di coordinate della pagina di gioco, mentre la proprietà colour definisce il colore dello sfondo.

E' possibile accedere allo stage di una applicazione in qualsiasi punto del codice utilizzando il metodo *[SPStage mainstage]*, uno stage gestisce e contiene di default un oggetto Juggler, che verrò in seguito analizzato, e che viene

utilizzato per gestire i Tween. La proprietà di *width* e *height* dello stage definiscono il sistema di coordinate della schermata dell'applicazione.

SPSprite Un SPSprite è la classe contenitore di oggetti grafici più semplice. Uno sprite viene utilizzato per raggruppare più oggetti in un unico sistema di coordinate. La classe definisce i metodi che consentono di aggiungere o rimuovere gli oggetti. Questo contenitore mantiene un elenco ordinato di oggetti, definendo i livelli di sovrapposizione secondo la struttura ad albero.

SPtextField Un SPTextField visualizza un elemento di testo, utilizzando sia i font standard iOS sia i font da bitmap inserita dall'utente. È possibile impostare tutte le proprietà normalmente utilizzate per il testo, come il nome del font e la dimensione, un colore, l'allineamento verticale e orizzontale, il bordo. La proprietà border è molto utilizzata durante lo sviluppo, perché permette di vedere i limite del campo di testo.

SPButton Un SPButton è un semplice pulsante composto da un'immagine e da un testo. È possibile associare una texture per mostrare il cambio di stato del pulsante, se non viene fornita una texture, il pulsante viene automaticamente scalato in dimensione quando viene toccato.

Inoltre, è possibile sovrapporre del testo sul pulsante. Per personalizzare il testo, sono forniti gli stessi metodi di SPTextField, il testo può essere spostato in una certa posizione con l'aiuto della proprietà textBounds.
Per rilevare un tocco, è messo a disposizione l'evento `SP_EVENT_TYPE_TRIGGERED`, è consigliabile utilizzare questo evento invece degli eventi touch standard, in questo modo, il pulsante si comporta come un pulsante standard di iOS.
Possono essere visualizzati due tipi di caratteri:

- Font iOS standard. Il testo è mostrato con caratteri standard iOS come Verdana o Arial. Vengono utilizzati questi font per mantenere una grafica simile a quella nativa, consigliabile se il testo non cambia troppo spesso.
- Font da bitmap. Un font bitmap ha, per ogni carattere e per ogni dimensione, un'immagine bitmap corrispondente che viene utilizzata per mostrare il carattere stesso. Vengono utilizzati per creare font personalizzati adatti al contesto dell'applicazione e per creare effetti speciali. I caratteri del font vengono resi disponibili su una grande texture. Per

usare un carattere, questo deve essere registrato con il metodo registerBitmapFont: passando il nome del carattere per la corrispondente proprietà del campo di testo.

4.1.3 Animation

La multimedialità è una parte fondamentale per questo tipo di applicazioni, per questa ragione Sparrow mette a disposizione dei metodi per realizzare animazioni di ogni genere nelle proprie applicazioni. Normalmente, durante lo sviluppo di un'applicazione, vengono utilizzate due tipi di animazioni:

- *Animazioni statiche*: in queste animazioni viene specificato uno stato iniziale, uno stato finale e un tipo di transizione da applicare dell'oggetto di visualizzazione considerato. Queste animazioni vengono impostate dal programmatore, e cominciano al verificarsi di un evento o all'apertura di una vista; l'utente utilizzatore non ha alcun tipo di interazione, si limita a visualizzarle.

- *Animazioni dinamiche:* in queste animazioni può essere specificato uno stato iniziale e finale, ma la transizione dipende dall'utilizzatore dell'applicazione. L'esempio classico è il drag-and-drop[2] di un oggetto tramite un tocco prolungato sullo schermo.

Le classi principali del package Animation sono raffigurate nella Figura 4.3. In alcuni motori di gioco, si ha quello che viene chiamato un run-loop; questo non è altro che un ciclo infinito che aggiorna costantemente tutti gli elementi presente nella scena. Tuttavia in Sparrow, a causa della struttura display tree, il ciclo run-loop non avrebbe senso a causa della separazione nella scena degli oggetti visualizzabili, ognuno di questi dovrebbe tener conto del tempo passato dall'inizio della scena per capire quando entrare in gioco.

Per questo motivo è stata creata la classe **SPEnterFrameEvent**, la quale crea un evento ad ogni fotogramma inviandolo a tutti gli oggetti nella struttura di visualizzazione. Esso contiene le informazioni del tempo passato dall'ultimo fotogramma. In questo modo, si possono rendere le animazioni indipendenti dalla frequenza dei fotogrammi, prendendo in considerazione solo il tempo passato.

[2]Il drag-and-drop indica una successione di tre azioni, consistenti nel cliccare su un oggetto virtuale (quale una finestra o un'icona) per trascinarlo (drag) in un'altra posizione, dove viene rilasciato (drop)

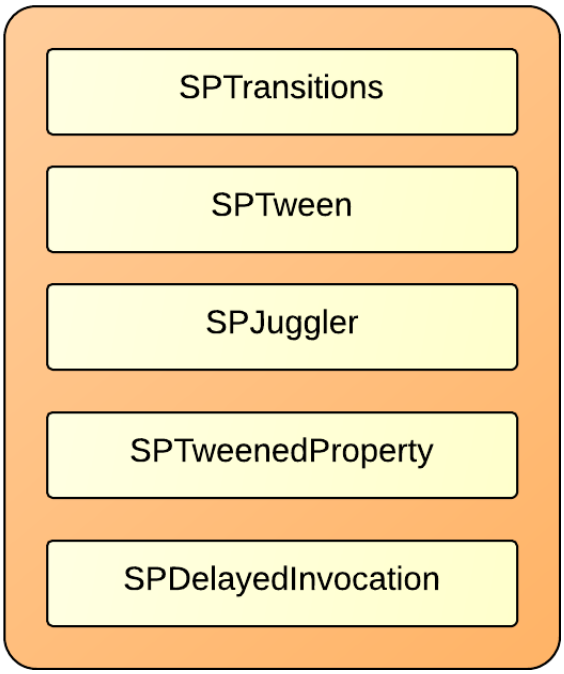

Figura 4.3: Classi del pacchetto animation

SPTween La classe SPTween restituisce un oggetto che è in grado di creare una transizione di una qualsiasi proprietà numerica dell'oggetto, portando tale proprietà da un valore iniziale ad un valore finale in un determinato intervallo di tempo prestabilito; essa utilizza diverse funzioni di transizione per dare le animazioni vari stili.

L'utilizzo principale di questa classe, quindi, è quella di creare animazioni standard come movimenti, dissolvenze, rotazioni, ecc, è sufficiente che la proprietà da animare sia di tipo numerico (int, uint, float, double). È inoltre possibile ritardare l'esecuzione di un tween di un determinato lasso di tempo. I Tween forniscono delle callback che possono essere eseguite durante l'arco della loro vita:

- onStart: invocato all'avvio del tween.
- onUpdate: invocato ad ogni avanzamento.
- onComplete: invocato al termine del tween.
- onRepeat: invocato ogni volta che il tween finisce una ripetizione.

La proprietà *repeatCount* consente di ripetere il tween più volte, mentre la proprietà *inverse* definisce il modo in cui saranno effettuate le ripetizioni.

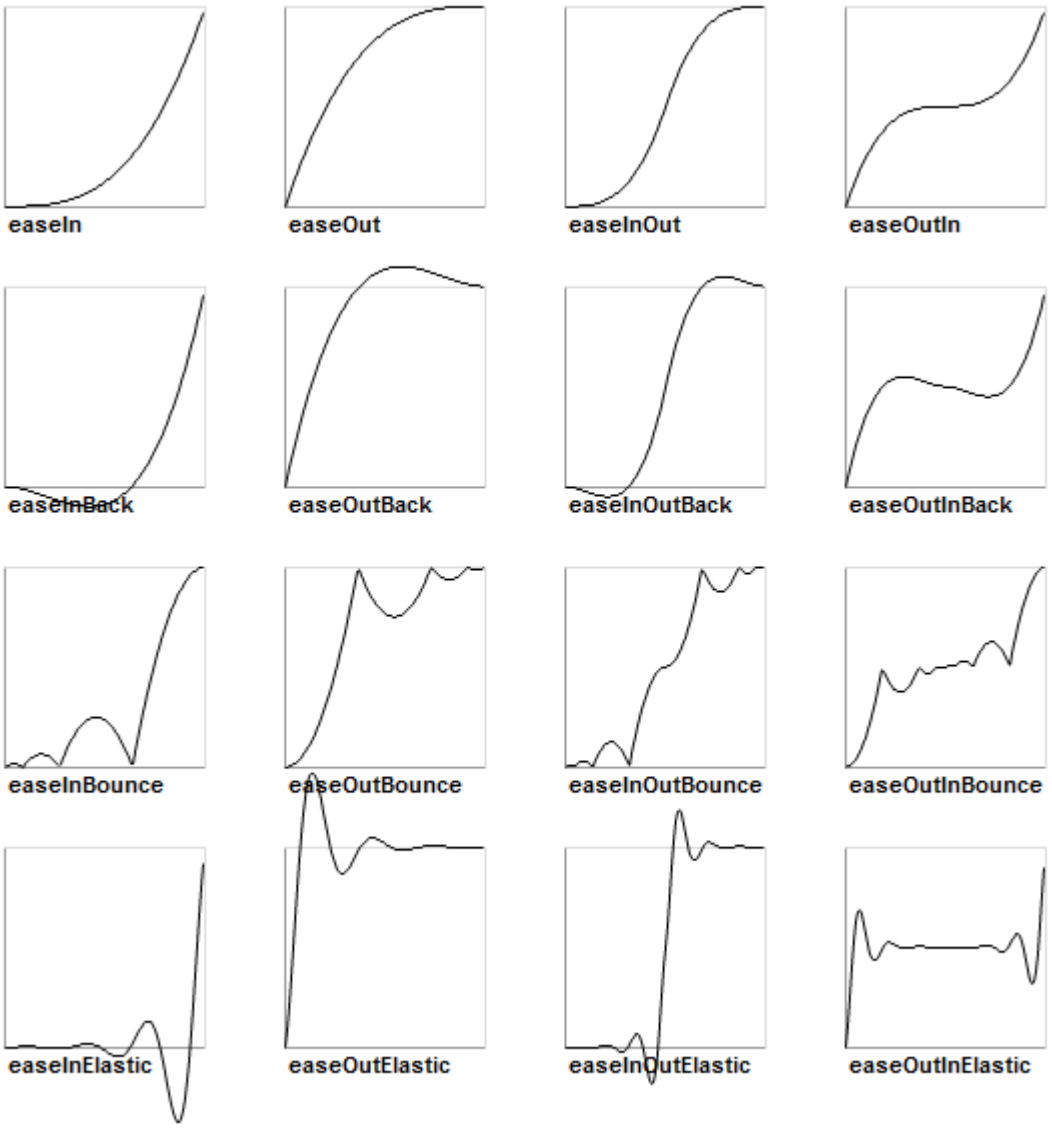

Figura 4.4: Transizioni applicabili ad un Tween

SPJuggler Il SPJuggler prende oggetti che implementano SPAnimatable, come SPTween, e si occupa di eseguirli. Il Juggler è un oggetto semplice, esso non fa altro che salvare un elenco di oggetti SPAnimatable e avanzare il loro tempo di esecuzione ad ogni fotogramma. Gli oggetti vengono inviati al Juggler invocando il metodo *addObject:* e rimossi con il metodo *removeObject*. Inoltre un oggetto può richiedere di essere rimosso dal Juggler con l'invio di un evento `SP_EVENT_TYPE_REMOVE_FROM_JUGGLER`.

E' possibile creare Juggler personalizzati, infatti questi vengono utilizzati per separare le animazioni di gioco da quelle dell'interfaccia, ad esempio per implementare la funzione di pausa nel gioco. Quando si creano Juggler personalizzati bisogna chiamare manualmente il metodo *advanceTime:* per far avanzare l'animazione

SPTransitions La classe **SPTransitions** contiene metodi statici che definiscono le funzioni di transizione dei tween. Tali funzioni saranno utilizzate da SPTween per eseguire animazioni. La funzioni di transizione disponibili sono presenti nella Figura 4.4.

SPTweenedProperty La classe SPTweenedProperty memorizza le informazioni di una singola proprietà di un oggetto SPTween. La sua proprietà *currentValue* aggiorna la proprietà specificata dell'oggetto target. Questa è una classe interna, è consigliato non utilizzarla manualmente.

4.1.4 Textures

Una istanza della classe **SPTexture** memorizza le informazioni che rappresentano un'immagine, perché questa non può essere visualizzata direttamente, ma deve essere mappata su un oggetto di visualizzazione. In Sparrow, l'oggetto di visualizzazione usato è di tipo SPImage. Sparrow supporta diversi formati di file per le textures, i formati più comuni sono il PNG, che contiene un canale alfa, e il JPG (senza un canale alfa). È anche possibile caricare i file nel formato PVR; questo è un formato speciale per dispositivi iOS, molto efficiente, che utilizza direttamente il chip grafico.
La Figura 4.5 mostra le classi del pacchetto Texture di Sparrow.
Sparrow supporta lo sviluppo di applicazioni ad alta risoluzione, ovvero la creazione di una applicazione che supporti allo stesso tempo i display normali e i display a risoluzione maggiore (Retina Display).

Quando il supporto per le texture ad alta definizione viene attivato (tramite il metodo *setSupportHighResolutions*) e si carica una texture da un file chiamato image.png, il sistema andrà automaticamente a cercare un file chiamato

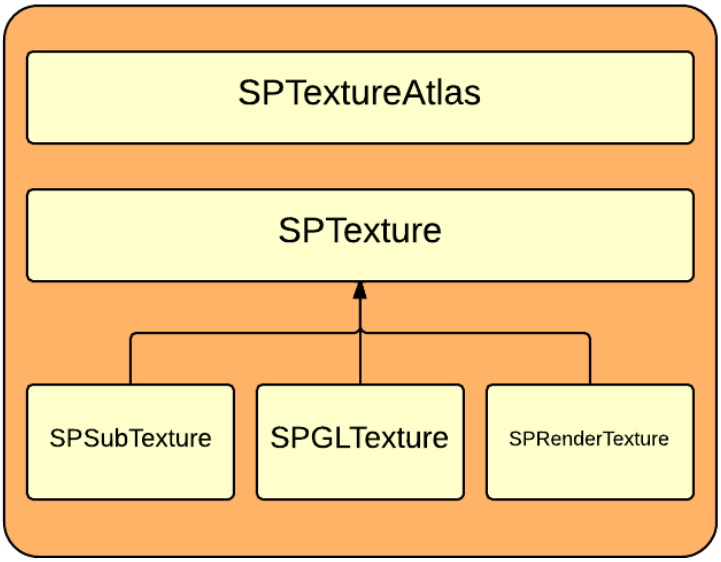

Figura 4.5: Classi del pacchetto texture

image@2x.png e, in caso positivo, caricherà quest'ultimo nel caso l'applicazione sia in esecuzione su un dispositivo con display retina. Questo tipo di nomenclatura viene utilizzato anche nell'iOS SDK di Apple.

Allo stesso modo è anche possibile cambiare la texture a seconda del dispositivo utilizzato: iPhone o iPad. La nomenclatura è quella di aggiungere al nome del file la stringa ipad o iphone subito prima del nome dell'estensione.

La classe **SPTextureAtlas** viene utilizzata per raccogliere tante texture in un'unica grande immagine chiamata *Atlas Texture*. Tale classe viene utilizzata per accedere alle singole texture che fanno parte dell'*Atlas Texture*, questa metodica viene utilizzata principalmente per aumentare le prestazioni e per risparmiare memoria.

4.1.5 Events

In Sparrow qualsiasi oggetto di visualizzazione all'interno del display tree può generare eventi, in quanto SPDisplayObject deriva da SPEventDispatcher. Una volta che un evento si verifica, la sua notifica passa per tutti i nodi dell'albero verso la radice, notificando ogni nodo correlato fino al nodo root.

La struttura del pacchetto Event è illustrata nella Figura 4.6.

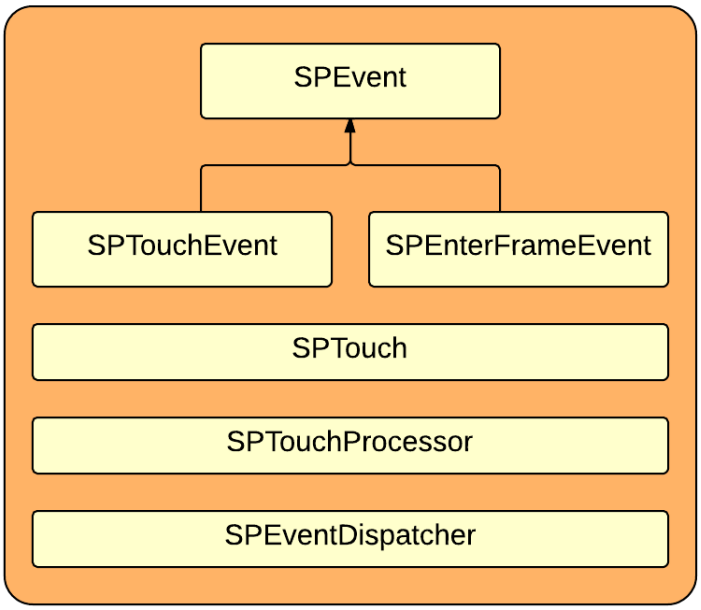

Figura 4.6: Classi del pacchetto event

SPEventDispatcher La classe **SPEventDispatcher** è la base per tutte le classi che inviano gli eventi. Il meccanismo degli eventi è una caratteristica fondamentale dell'architettura di Sparrow, infatti gli oggetti possono comunicare tra loro attraverso gli eventi. Un dispatcher di eventi può inviare eventi (oggetti di tipo SPEvent o una delle sue sottoclassi) per gli oggetti che si sono registrati come ascoltatori. Per identificare il tipo di evento viene utilizzata una stringa che lo descrive. Per inviare un evento si utilizza il metodo di classe *dispatchEvent*, l'evento viene poi ricevuto dagli oggetti che, tramite il metodo *addEventListener* si sono registrati come ascoltatore per quell'evento. Al verificarsi dell'evento, l'ascoltatore può lanciare in automatico un metodo da esso implementato che esegue determinate azioni. Data la struttura ad albero, un ascoltatore può registrarsi per un tipo di evento, non solo per l'oggetto che lo spedisce, ma su qualsiasi oggetto che sia un genitore diretto o indiretto del dispatcher.

SPEvent La classe SPEvent contiene le informazioni che descrivono un evento, l'oggetto SPEventDispatcher crea istanze di SpEvent e le invia a tutti i nodi registrati come ascoltatori. L'oggetto evento contiene informazioni che caratterizzano un evento, come il tipo di evento, il target (l'oggetto che lo invia) dell'evento, e se l'evento deve risalire fino alla root del display tree.

Nel caso si desideri creare un evento personalizzato, è possibile creare una sot-

toclasse di SPEvent ed aggiungere altre proprietà con le informazioni aggiuntive per l'ascoltatore. Le classi **SPEnterFrameEvent** e **SPTouchEvent** sono un esempio di questa pratica, la prima aggiunge delle proprietà relative al tempo trascorso dall'ultimo fotogramma mentre la seconda rileva il tocco sullo schermo di uno o più dita inviando un eventi di tipo SPTouchEvent.

SPTouch La classe SPTouch contiene informazioni relative alla presenza o al movimento di un dito sullo schermo. Quando avviene un evento di tocco si riceve un oggetto di tipo SPTouchEvent, con un evento di questo tipo è possibile eseguire una query per tutti i tocchi che sono stati rilevati sullo schermo. Le informazioni sul singolo tocco sono contenute all'interno di un oggetto SPTouch.

Un tocco sullo schermo è generalmente composto da tre fasi: inizio del touch, spostamento e fine del touch. Un tocco può anche entrare in una fase detta stazionaria: questo avviene solamente quando è attivata la funzione Multitouch, in questo caso non si attiva un evento di tocco. Un esempio è quando si verifica una situazione in cui un dito è fermo e l'altro è in movimento, nell'elenco dei tocchi di quell'evento , sarà presente un tocco nella fase stazionaria.

Un altro aspetto da considerare è la posizione del tocco, è possibile ottenere la posizione corrente e l'ultima posizione sulla schermata rispettivamente con le proprietà *globalX, globalY e previousGlobalX, previousGlobalY*.
La classe **SPTouchProcessor** elabora le informazioni relative ai tocchi sullo schermo e la invia agli oggetti di visualizzazione. Questa è una classe interna, viene utilizzata autonomamente da Sparrow.

4.1.6 Audio

La classe **SPAudioEngine** prepara il sistema per la riproduzione audio e controlla inoltre il volume globale dell'applicazione. Prima di riprodurre qualsiasi tipo di suono, il motore audio deve aprire una sessione audio, questa viene inizializzata con il metodo *start*. Ci sono varie tipologie di sessioni audio che definiscono in il modo in cui iOS gestirà l'elaborazione audio, ad esempio se la musica riprodotta da iPod si mixerà con l'audio dell'applicazione o meno. Al termine dell'applicazione deve essere chiamato il metodo *stop* che termina la sessione audio liberando le risorse di memoria.
Le classi del pacchetto audio sono rappresentati in Figura 4.7.
La **SPSound** è un contenitore per dati audio, proprio come SPTexture contiene i dati di un'immagine, SPSound contiene i dati di una traccia audio, carica i file audio in memoria e li tiene pronti alla riproduzione. È possibile

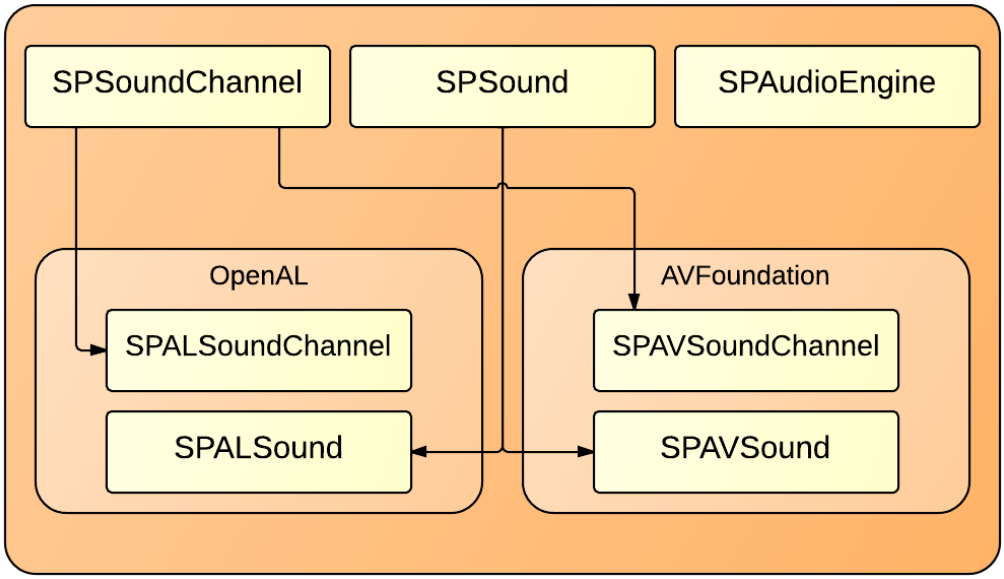

Figura 4.7: Classi del pacchetto audio

utilizzare SPSound per riprodurre direttamente un suono, utilizzando il metodo *play*, ma questo non darà nessun controllo su di esso. Se si desidera controllare la riproduzione con le funzionalità di play, stop e pausa e con il controllo del volume, è necessario creare un oggetto di tipo **SPSoundChannel** tramite il metodo *CreateChannel*.

Nel caso sia necessario riprodurre un suono più volte basterà creare l'oggetto SPSound una sola volta mantenendolo in memoria, all'atto della riproduzione è sufficiente chiamare il metodo *play* o creare un nuovo canale. Un suono viene automaticamente messo in pausa quando l'applicazione viene interrotta da un evento esterno all'applicazione, come un messaggio, alla ripresa continuerà la riproduzione dal punto dove si era interrotto.

Le classi **SPAVSoundChannel** e **SPALSoundChannel** sono le implementazioni della classe SPSoundChannel rispettivamente per creare canali audio utilizzando AVAudioPlayer e OpenAL, non è necessario utilizzarle manualmente in quanto SPSound si occupa in maniera automatica di scegliere la tecnologia appropriata per la riproduzione audio: i file non compressi useranno OpenAL mentre file audio compressi useranno AVAudioPlayer di Apple.

4.1.7 Utils

Figura 4.8.

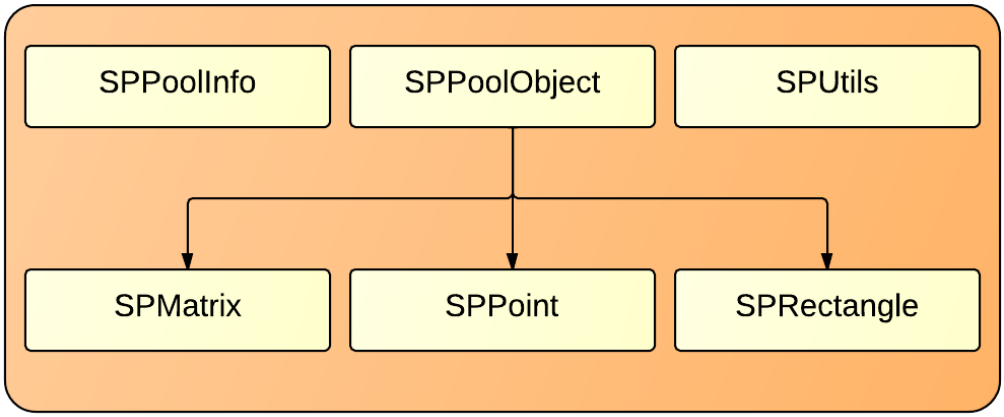

Figura 4.8: Classi del pacchetto util

La classe **SPPoolObject** è una valida alternativa alla classe NSObject per la gestione di aggregati di oggetti. La differenza nell'uso di SPPoolObject è che le sue sottoclassi non deallocano l'oggetto in memoria quando il suo counter dei puntatori raggiunge lo zero, l'oggetto rimane in memoria e viene riutilizzato quando ne viene richiesta una nuova istanza. In questo modo, l'inizializzazione dell'oggetto viene accelerata aumentando le prestazioni. Sparrow utilizza inoltre questa classe per creare oggetti di supporto come SPPoint, SPRectangle e SPMatrix. **SPPoolInfo** viene utilizzato come helper interno da SPPoolObject.

Le rimanenti classi si supporto sono: **SPUtils**, contiene metodi di utilità per uso generale come funzioni matematiche e gestione dei file. La classe **SPPoint** descrive un punto bidimensionale o vettoriale, mentre **SPMatrix** descrive una trasformazione affine per una matrice bidimensionale. Essa fornisce metodi per manipolare la matrice in modo efficiente, e può essere utilizzato per trasformare punti. Infine, **SPRectangle** descrive un rettangolo dal punto dell'angolo superiore sinistro (x, y) e dalla sua larghezza e altezza.

4.2 Analisi dell'applicazione

L'applicazione presa in analisi fa parte del progetto APP-KID, la struttura è molto simile a quella riportata nella sezione 1.2, di seguito andremo a fare una breve descrizione.

L'applicazione **Lana!** è una app concepita e prodotta per bambini dai due ai cinque anni.

Si tratta di un gioco-storia, o meglio di una serie di giochi inseriti in una narrativa, nel quale i bambini sono liberi di esplorare il mondo di lana che si scopre aprendo la cassettiera del menu della app seguendo un percorso lineare, simile allo sfogliare le pagine di un libro, o scegliendo di volta in volta, dal paesaggio che si vede aprendo il cassetto, in quale scena immergersi e in quali attività cimentarsi.

Ogni scena del paesaggio presenta un'animazione primaria, che parte quando si arriva nella scena e serve a coinvolgere il bambino nell'ambiente e stimolare la sua curiosità a giocare, e di una serie di aree interattive che permettono al bambino di giocare una dopo l'altra a dieci attività di gioco diverse, pensate per intrattenere piacevolmente e far lavorare le aree della percezione, dell'attenzione e dell'area della motricità fine che appartengono allo sviluppo in quella fascia di età.

Ad ogni mini-gioco corrisponde una ricompensa consistente in un maglione da impilare nel cassetto del bottino: guadagnati tutti e dieci i maglioni corrispondenti ai giochi, la pila cade per permettere di ricominciare daccapo e si vince un disegno originale che rappresenta una scena o un personaggio della storia, da stampare e colorare.

Il menù principale, visibile nella Figura 4.9, è composto da una cassettiera composta da quattro cassetti che rappresentano dei pulsanti, ognuno con una diversa funzione:

- Il cassetto STORIA: premendo questo pulsante viene avviata la prima pagina di gioco.
- Il cassetto MAGLIONI: ad ogni pagina di gioco completata, si guadagna un premio: questo consiste in un maglione. In questa pagina vengono raffigurati il numero di maglioni vinti durante il gioco.
- Il cassetto BOTTINO: da questa pagina si può accedere a dei minigiochi extra. Essi vengono visualizzati solamente al raggiungimento di un determinato numero di maglioni.
- Il cassetto CREDITS: viene visualizzata una pagina crediti con gli editori, autori e sviluppatori.

Altri elementi della schermata, posti sopra la cassettiera sono animabili, e portano a contenuti extra o ad animazioni dinamiche nello schermo. Un esempio di contenuto extra può essere quello di riordinare un puzzle o colorare un disegno sullo schermo.

Figura 4.9: Schermata del Menù principale

4.2.1 Pagine di gioco

Alle pagine di gioco si accede in maniera sequenziale dal menù principale, tramite il pulsante STORIA. In totale le pagine di gioco sono dieci ed in ognuna di esse sono presenti tre caratteristiche:

- Effetti sonori: gli effetti sonori avvengono all'apertura della pagina di gioco, successivamente una voce narrante illustra e descrive il contenuto della pagina. Le animazioni e i mini-giochi contengono a loro volta effetti audio che vengono riprodotti con l'interazione dell'utente.

- Animazioni: le animazioni vengono usate principalmente per dare dinamicità alla pagina, anch'esse vengono attivate all'apertura della pagina. Nella maggior parte dei casi, gli elementi animati, possono interagire con l'utente e riprodurre altre animazioni sullo schermo.

- Mini-gioco: questo elemento è la parte più complessa di ogni pagina. Al completamento di ogni mini-gioco viene ricevuto un premio e viene data la possibilità di accedere alla pagina di gioco successiva dell'applicazione. I giochi richiedono l'interazione con l'utente principalmente tramite tocchi e trascinamenti di oggetti sullo schermo. Molto spesso viene anche utilizzato il giroscopio elettronico, questo dispositivo hardware integrato nei device, permette di rilevare il movimento del dispositivo su sei assi, riuscendo a percepire anche le più piccole rotazioni e inclinazioni del dispositivo. Utilizzando questa caratteristica si possono, ad esempio, spostare oggetti sullo schermo ruotando ed inclinando il dispositivo e non più compiendo trascinamenti con il dito sullo schermo.

Le pagine di gioco totali sono 10, a queste si sommano altre 3 pagine usate per realizzare dei mini-giochi extra. Alcuni esempi di realizzazione sono visibili in Figura 4.10 e Figura 4.11. Nella realizzazione dei mini-giochi, in base alle richieste di progetto, vengono utilizzate le funzionalità di Sparrow (tween, juggler, videoclip, audio, eventi touch ecc..) insieme alle funzionalità di iOS, ad esempio, l'accelerometro per far muovere la barca in Figura 4.11. Nel caso in cui sia richiesto di rilevare collisioni tra oggetti presenti nel gioco, viene usato il motore di fisica 2D.

Nella sezione successiva andremo a vedere in dettaglio lo sviluppo di un mini-gioco, spiegando le fasi di realizzazione e le libreria aggiuntive utilizzate. Infine vanno fatte alcune considerazioni generali sulle pagine di gioco: essendo un'app dedicata ai bambini, questa si basa sul passatempo e sulla

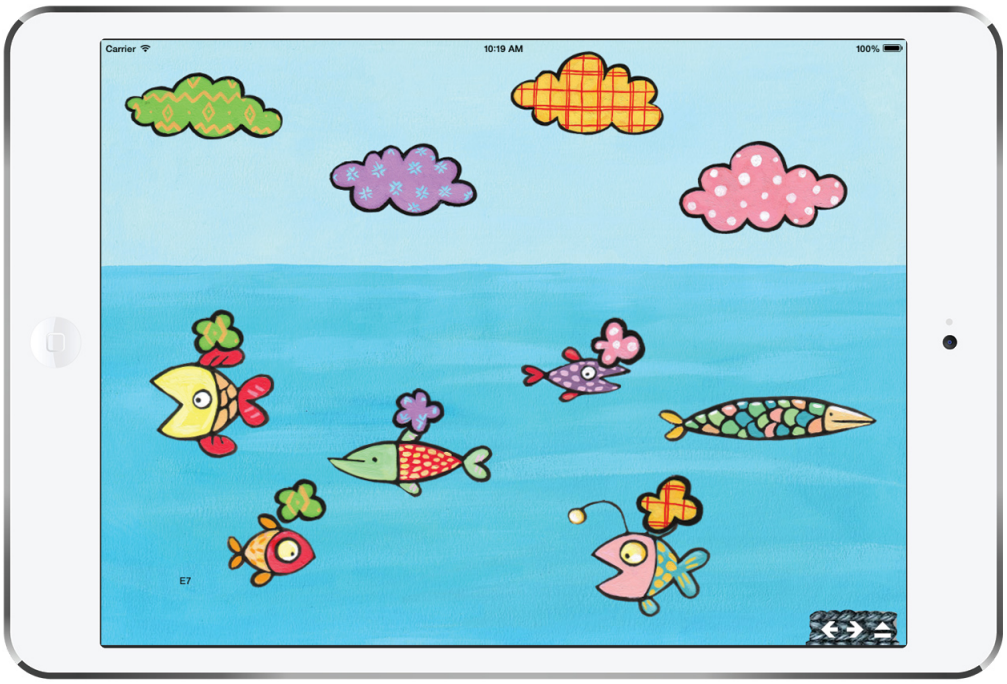

Figura 4.10: Schermata relativa al mini-gioco FishWorld

Figura 4.11: Schermata relativa al mini-gioco ShipWorld

piacevolezza di gioco. Il livello di difficoltà generale dei mini-giochi è adeguato al target utente e inoltre, non vi è nessun vincolo temporale per il completamento di essi.

4.3 Realizzazione di un mini gioco

Il mini-gioco da realizzare consiste nella creazione di un *CarWorld*, cioè una serie di piccole automobili che si spostano in una mappa stradale, costituita da strade che si incrociano. All'incrocio dei percorsi stradali è posizionato un semaforo che, in base al colore (verde, giallo e rosso) autorizza o meno il passaggio dell'auto. Le auto si muovono autonomamente seguendo i percorsi stradali, l'utente interagisce con il gioco tramite il tocco delle auto: toccando un auto questa si ferma, con un altro tocco questa riparte.

Lo scopo del gioco consiste nell'accumulare un punteggio prestabilito effettuando un determinato numero di passaggi corretti agli incroci. Nel caso un automobile passi con il rosso o si scontri con un'altra automobile il punteggio viene azzerato.

Rispetto ad uno scenario reale vengono fatte alcune semplificazioni per permettere all'utente, un bambino, di terminare il gioco correttamente senza troppe difficoltà: le auto presenti nello schermo viaggiano ad una velocità ridotta e vengono evitati scenari con più auto in senso opposto nella stessa strada.
Le classi create per la gestione degli elementi di gioco sono tre:

- CarWorld per la creazione dell'ambiente fisico di gioco.
- Car per la creazione delle auto.
- Semaforo per creare e gestire i semafori.

Oltre alle classi sopracitate sono state utilizzate delle classi per la gestione dei percorsi stradali che andremo a descrivere nelle sezioni successive.

4.3.1 CarWorld

Per la creazione di un ambiente di gioco deve essere utilizzato un *container* di oggetti che poi sarà collegato all'albero di visualizzazione dell'applicazione. Come detto in precedenza, la classe container più semplice è la *SPSprite*, e utilizzando la libreria *SPPhysics* questa classe viene specializzata in *SPWorld*.

La classe SPWorld è viene utilizzata per creare un ambiente base che integra Sparrow, in cui si vuole aggiungere la simulazione della fisica. SPWorld è sottoclasse di SPSprite, quindi può essere usata come un qualsiasi SPSprite, ma è dotata di alcune funzioni aggiuntive che la specializzano; viene ridefinito il metodo *update*, eseguito ad ogni frame, che oltre a far avanzare lo stato di Sparrow esegue uno step di avanzamento della simulazione della fisica. La classe *SPWorld* implementa inoltre il protocollo *ContactListenizer*, un'interfaccia che permette di ricevere gli eventi legati alle interazioni tra gli oggetti del World, come le collisioni, le separazioni e le sovrapposizioni.

Oltre alle classiche proprietà di SPSprite (dimensione e posizionamento container, visibilità, pivot e alpha), SPWorld include un puntatore ad un oggetto della libreria Box 2D chiamato b2World, necessario per la simulazione dell'ambiente fisico. Le altre proprietà della classe sono quelle relative al vettore di gravità, che specifica direzione e verso della forza di gravità ed altri parametri necessari per il setup dell'ambiente fisico.

Per la realizzazione del mini-gioco è stata realizzata una sottoclasse di SPWorld chiamata appunto CarWorld. Il setup iniziale (Codice 4.1) prevede che il vettore di gravità sia nullo e che la dimensione dell'ambiente di gioco sia pari alle dimensioni dello schermo del dispositivo. Di seguito un estratto del codice, l'oggetto *self* è riferito a CarWorld.

```
[self setGravity:CGPointMake(0.0f, 0.0f)];
[self setWidth: 1024.0f];
[self setHeight: 768.0f];
[self setX:0.0f];
[self setY:0.0f];
```

Codice 4.1: Setup iniziale

Nell'interfaccia della classe sono a disposizione diversi metodi per la creazione degli elementi che compongono il gioco e per la gestione degli eventi:

- Il metodo onOverlapBody (del protocollo ContactListenizer) è usato per rilevare le sovrapposizioni tra le auto e i semafori. In particolare si rilevano sovrapposizioni tra auto e auto e tra auto e semaforo. I due tipi di sovrapposizioni vengono distinte dal sistema, e causano effetti diversi nell'ambiente di gioco (suoni, punteggio, animazioni).

- I metodi carSpawn e semaforoSpawn per la creazione di auto e semafori. Questi metodi hanno diversi parametri, come ad esempio la posizione, il tipo di percorso dell'auto o lo stato del semaforo.

- I metodi spawnPoint e SpawnCloud, servono per mostrare a video tramite un tween, rispettivamente la collezione di un punto di gioco e la collisione tra auto.
- Metodi di supporto per la creazione dei path di percorso e dei tween che seguono il path delle auto. Verranno trattati nella sezione 4.3.4.
- Metodi che vengono eseguiti in base agli eventi, un esempio è *onEnterFrame* che viene eseguito all'inizio di ogni frame e viene inviato a tutti gli elementi del display tree.
- Metodi di debug, per la deallocazione della memoria e per il reset del gioco.

In Figura 4.12 viene mostrata l'immagine sfondo principale di CarWorld:

Figura 4.12: CarWorld

Per l'inizializzazione di CarWorld vien utilizzato il Codice 4.2 per creare un'istanza di *CarWorld* all'interno della pagina dell'applicazione. La riga 2 alloca e inizializza un oggetto CarWorld, quella successiva lo rende visibile nello schermo. Con l'istruzione alla riga 4 l'oggetto CarWorld viene aggiunto al display-tree dell'applicazione.

```
1 //Inizializzo CarWorld
2 carWorld = [[CarWorld alloc] init];
3 [carWorld setAlpha:1.0];
4 [self addChild:carWorld];
```

Codice 4.2: Creazione CarWorld

4.3.2 Car

Per quanto riguarda la creazione delle auto è stata creata una sottoclasse *Car* che specializza la classe *SPBody*. Andiamo ora a elencare le caratteristiche principali della classe SPBody.

Un SPBody è un oggetto, che simula la fisica, derivato dalla classe SPDisplayObjectContainer. E' un *container* perché non è semplicemente un singolo oggetto ma può essere composto da più oggetti, sia grafici che fisici. Infatti la possibilità di essere composto da più oggetti fisici è data Box 2D mentre la possibilità di contenere più oggetti grafici è data dalla classe SPDisplayObjectContainer.

Un SPBody ha informazioni sulla sua posizione e velocità, ed possibile applicargli forze, momenti e impulsi. Le principali proprietà che ne determinano il comportamento sono le seguenti:

- Posizione nel *World* di riferimento.
- Angolo rispetto *World* di riferimento.
- Velocità lineare ed angolare del body.
- Velocità di dumping (lineare e angolare), usata per ridurre la velocità del body.
- Tipo di fisica: statica, cinematica, dinamica.
- Scala di gravità applicata al body.
- Flag booleani per indicare lo stato (attivo o non) ed altri parametri.

I comportamenti fisici dei body sono tre, e hanno le seguenti caratteristiche:

- **Body Statico**: un body statico si comporta come se avesse massa infinita, ha quindi velocità nulla, e non si sposta quindi durante la simulazione. I body statici possono essere spostati manualmente dall'utente e non si scontrano con body statici o cinematici.

- **Body Cinematico**: un body cinematico si muove durante la simulazione secondo la sua velocità iniziale. I body cinematici non rispondono a forze esterne, essi possono essere spostati manualmente dall'utente, ma normalmente un corpo cinematico viene spostato impostando la sua velocità. Un body cinematico non collide con altri body statici o cinematici.
- **Body Dinamico**: un body dinamico è un body completamente simulato. Può essere spostato sia manualmente dall'utente sia secondo forze applicate o eventuali collisioni con altri body. Un corpo dinamico può collidere con tutti i tipi di corpo, inoltre ha sempre massa finita e diversa da zero. Se si tenta di impostare la massa di un body dinamico a zero, questa acquisirà automaticamente massa di valore uno.

Una *Car* è quindi una sottoclasse di SPBody che, oltre alla caratteristiche appena citate, ha altre due proprietà riportate nel Codice 4.3:

```
@property BOOL isStopped;
@property (nonatomic, retain) SPJuggler* juggler;
```

Codice 4.3: Proprietà Car

La proprietà *isStopped* indica quando l'auto è stata fermata con un tocco sullo schermo, mentre il *juggler* serve per far avanzare o fermare l'animazione dell'auto con il touch dell'utente.

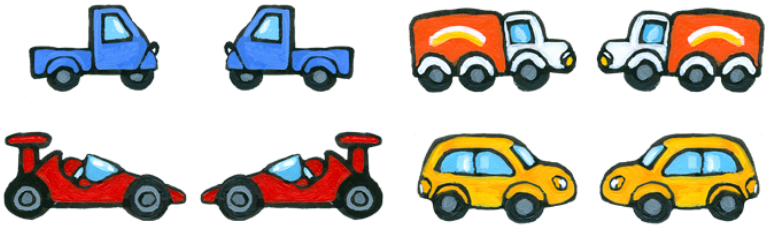

Figura 4.13: Immagini utilizzate per la realizzazione delle auto

Nel caso dell'auto ci sono due oggetti che formano l'oggetto Car. Un oggetto è di tipo grafico, costituito da una SPImage (le immagini utilizzate sono in Figura 4.13) questa viene aggiunta al body dell'oggetto Car tramite il metodo *addChild:*. Il secondo oggetto del body è quello che si occupa del comportamento fisico, ed è di tipo *b2Body* appartenente alla libreria Box2D; questo viene creato automaticamente all'atto di creazione del body. Il tipo di fisica utilizzata per le auto è quella dinamica, infatti è necessario rilevare le collisioni tra auto-auto e auto-semaforo.

4.3.3 Semaforo

La realizzazione dei semafori prevede, come per le auto, la specializzazione delle classe SPBody. Anche questo caso valgono tutte le proprietà citate in precedenza per l'SPBody, oltre a questa è stata creta una proprietà *isRed* per visionare lo stato del semaforo.

Tramite il file d'interfaccia del semaforo, visibile nel Codice 4.4, viene reso disponibile all'esterno il metodo *changeSemaforo*. Questo metodo cambia lo stato del semaforo portandolo da verde a rosso e viceversa.

```
@interface Semaforo : SPBody

+(id) semaforo;

@property BOOL isRed;

/** Cambia stato semaforo da rosso a verde e viceversa*/
-(void) changeSemaforo;

@end
```

Codice 4.4: Interfaccia del semaforo

A differenza dell'oggetto Car, nel semaforo ci sono più oggetti grafici che lo compongono.

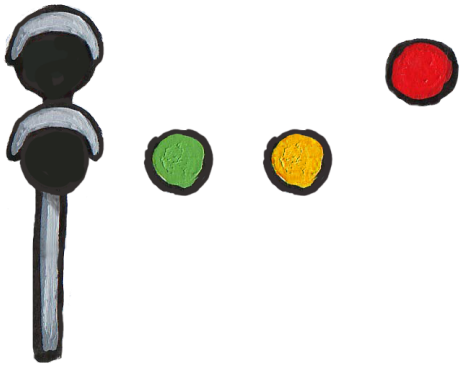

Figura 4.14: Immagini utilizzate per la realizzazione di un semaforo

Le immagini utilizzate sono visibili in Figura 4.14. La prima immagine a sinistra costituisce l'oggetto grafico base per il semaforo, successivamente a seconda dello stato, vengono aggiunte o rimosse al display-tree del semaforo le immagini dei colori (verde, giallo e rosso). Anche il semaforo contiene l'oggetto *b2Body* per la gestione del comportamento fisico. Il tipo di fisica

utilizzata è quella dinamica, infatti è necessario rilevare le collisioni auto-semaforo quando lo stato del semaforo è rosso.

```
//Creo due semafori
sem1 = [self semaforoSpawn:CGPointMake(428, 475) :1];
sem2 = [self semaforoSpawn:CGPointMake(538, 540) :0];
sem3 = [self semaforoSpawn:CGPointMake(785, 100) :1];
sem4 = [self semaforoSpawn:CGPointMake(680, 250) :0];
//Aggiungo un box al centro dell'incrocio
[sem1 addBoxWithName:@"incrocio1" ofSize:CGSizeMake(20, 20)
      atLocation:CGPointMake(400, 450)];
[sem2 addBoxWithName:@"incrocio2" ofSize:CGSizeMake(20, 20)
      atLocation:CGPointMake(620, 520)];
[sem3 addBoxWithName:@"incrocio1" ofSize:CGSizeMake(20, 20)
      atLocation:CGPointMake(760, 150)];
[sem4 addBoxWithName:@"incrocio2" ofSize:CGSizeMake(20, 20)
      atLocation:CGPointMake(620, 200)];
```

Codice 4.5: Creazione dei semafori

La creazione programmatica dei semafori è visibile in Codice 4.5, il metodo *semaforoSpawn* ha come parametri la posizione nello schermo del semaforo e il suo stato iniziale. Ad ogni semaforo viene agganciato al body un Box con il metodo *addBoxWithName*. Il box è un'estensione del body del semaforo, formato da un quadrato non visibile sullo schermo. Il box viene posizionato all'interno degli incroci, per rilevare le collisioni con le auto, mentre l'immagine del semaforo è posizionata esternamente alla strada.

4.3.4 Creazione dei percorsi

Lo sviluppo di questo mini-gioco richiede alle auto di seguire dei percorsi predefiniti, ovvero le strade. La configurazione utilizzata per le strade è mostrata in Figura 4.15, si possono notare tre strade, che non formano nessun percorso chiuso (loop), in particolare: due strade orizzontali e una strada verticale.

Il percorso verticale essendo rettilineo può essere approssimato con una retta, mentre i percorsi orizzontali essendo non rettilinei, non possono essere approssimati con una semplice retta ma utilizzando più curve concatenate tra loro. Per modellare i percorsi è stata utilizzata la **curva di Bèzier**, una particolare curva parametrica che, nella sua forma più semplice può rappresentare una retta e che ha grande applicazione nella computer grafica. La curva di Bézier è un tipo di curva semplice da usare, e che può descrivere molte forme. Questa curva viene notoriamente utilizzata con svariati

Figura 4.15: Tracciati seguiti dalle auto

scopi: per la creazione dei caratteri nei font grafici, nella progettazione del design di veicoli, nella grafica vettoriale e in strumenti di animazione 3D per rappresentare percorsi.

Nel campo videoludico, le curve di Bézier vengono utilizzate per descrivere percorsi: ad esempio un circuito da percorrere nei giochi di auto o delle linee da seguire in giochi come Flight Control (controllore di volo).

Le curve di Bézier sono molto popolari perché la loro descrizione matematica è compatta, intuitiva ed elegante. Sono facili da calcolare, facili da usare anche in dimensioni superiori (3D in su), e possono essere messe insieme per rappresentare qualsiasi forma si desideri.

Una curva di Bézier è descritta da una funzione matematica di parametro t. Ogni valore della funzione è un punto sulla curva, che dipende dal parametro t, e da una serie di punti, detti *punti di controllo*. Il primo e l'ultimo punto di controllo sono rispettivamente il punto di inizio e fine della curva, mentre per i restanti punti di controllo la curva non vi passa attraverso.

Il valore del parametro t è positivo e varia da 0 a 1 compresi. Il valore 0 corrisponde al punto iniziale della curva, mentre il valore 1 corrisponde al punto finale della curva. Valori intermedi corrispondono ad altri punti della

curva. Nel nostro caso è stata utilizzata la curva di Bèzier di ordine tre, detta anche *Curva di Bèzier Cubica*, di seguito una breve descrizione.

Curve di Bézier cubica Una curva di Bézier cubica è composta da quattro punti P0, P1, P2 e P3 nel piano o in uno spazio tridimensionale. La curva ha inizio in P0 si dirige verso P1 e finisce in P3 arrivando dalla direzione di P2. La curva non passa mai per i punti P1 o P2, infatti questi punti sono necessari solo per dare alla curva informazioni sulla direzione da seguire. La distanza tra P0 e P1 determina quanto la curva si muove nella direzione di P2 prima di dirigersi verso P3. La forma parametrica della curva è la seguente:

$$B(t) = P_0(1-t)^3 + 3P_1t(1-t)^2 + 3P_2t^2(1-t) + P_3t^3, t \in [0,1]$$

Le curve esprimibili con questa funzione sono infinite, per riassumere i comportamenti principali della curva al variare dei punti di controllo si può osservare la Figura 4.16. Si può notare come tutte le curve abbiano i punti di inizio e fine alla stessa distanza, ma ogni curva ha un andamento diverso in base alla posizione dei punti di controllo. Da sinistra verso destra si osservano le seguenti curve:

(a) una curva senza flesso.

(b) una curva con flesso.

(c) due curve rette.

(d) una curva con cuspide.

(e) una curva con loop.

Percorsi di Bézier Nel caso si debba rappresentare una curva di Bèzier complicata, come quella del percorso delle auto, ci sono due opzioni:

- usare una singola curve di Bèzier di grado maggiore.
- suddividere la curva complicata in più segmenti, usando una curve di Bèzier di grado minore per ogni segmento.

La seconda opzione è quella che porta a creare un *percorso Bézier* (Bèzier Path), che è molto più semplice ed efficiente da utilizzare rispetto alle curve con alto grado.

La Figura 4.17 mostra la tecnica utilizzata per la costruzione di un Bèzier path. In ogni segmento (eccetto quello iniziale e finale) i punti di inizio

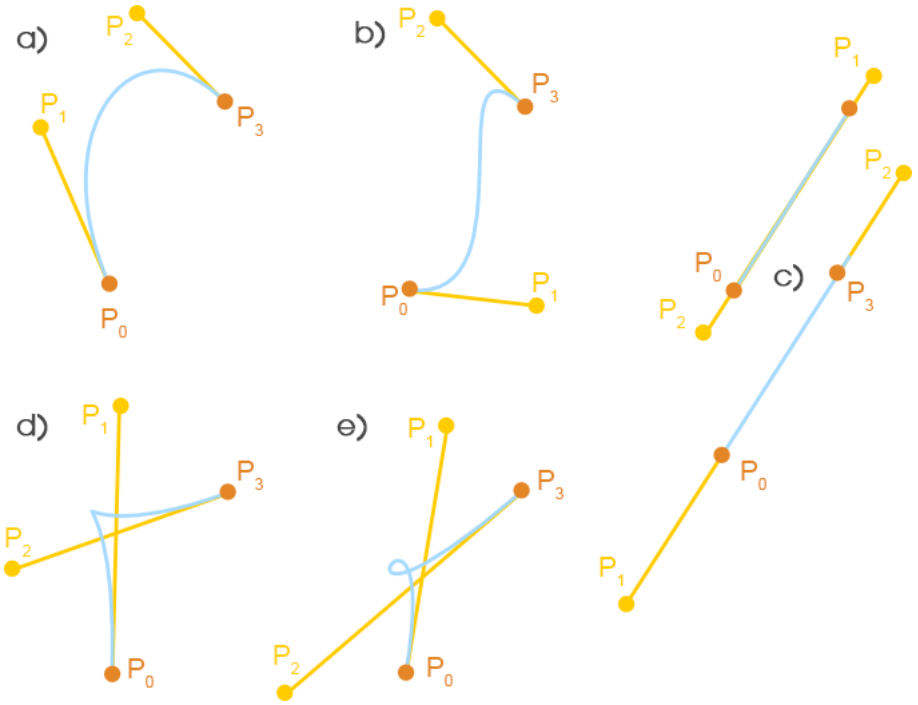

Figura 4.16: Esempi di curve di Bézier 2D

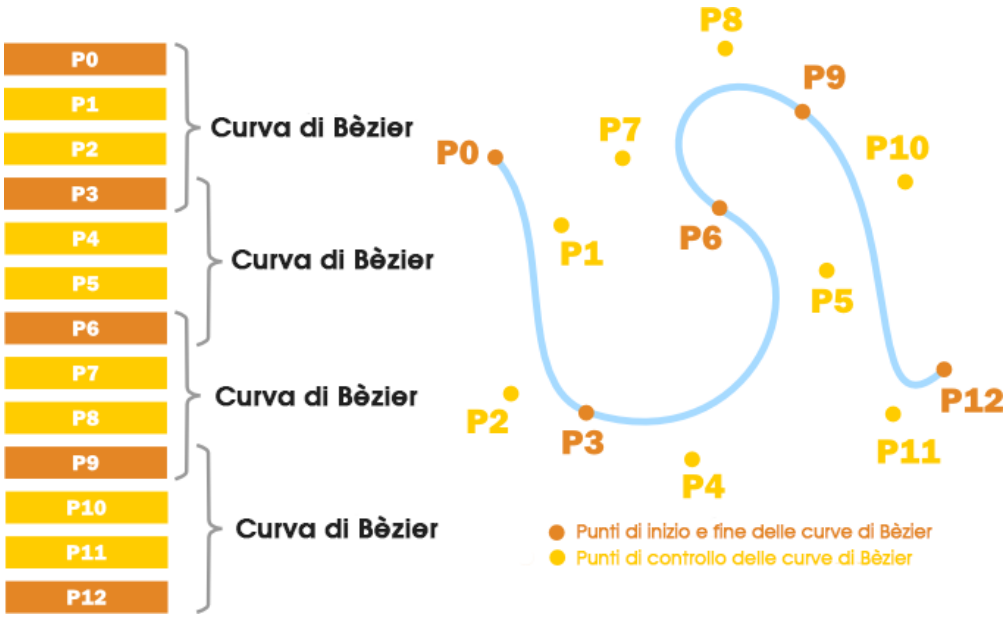

Figura 4.17: Creazione di un Bèzier Path

e di fine coincidono rispettivamente con il punto terminale del segmento precedente e con il punto iniziale del segmento successivo. Come si può notare tutti i segmenti sono delle curve di grado 3.
Nel caso in esame, è stata utilizzata una classe di Sparrow che permette di creare dei *Bèzier Path* applicando come input delle curve di Bèzier di grado 3, dove ognuna costituisce un segmenti della curva. Le firme dei metodi di supporto utilizzati, sono visibili nel Codice 4.6, questi usano al loro interno i metodi di Sparrow per la creazione dei percorsi.

```
/**Creo il percorso per la pista in base al tipo di
    percorso passato per parametro*/
- (void) createPath:(BBCubicBezierPath *) path type:(int)
    pathType

/**Creo un tween per il BBCubicBezierPath path che ha come
    target l'SPBody target con un ritardo temporale delay*/
- (void) createTweenWithPath:(BBCubicBezierPathTween *)
    tween forPath:(BBCubicBezierPath *) path forTarget:(
    SPBody *)target withDelay:(double)delay
```

Codice 4.6: Metodi per la creazione dei Bèzier Path

Nell'estratto di Codice 4.7 vi è un esempio di costruzione di un percorso corrispondente alla strada orizzontale del gioco. E' stata inoltre effettuata una modifica alla versione originale di Sparrow, aggiungendo la proprietà *versus* al percorso, in modo da far identificare il verso di questo, così da mostrare l'immagine dell'auto nel verso corretto.

```
//strada orizzontale in alto
path.versus = leftToRight;
[path addSegmentWithA:[SPPoint pointWithX:left y:360]
                    b:[SPPoint pointWithX:left y:360]
                    c:[SPPoint pointWithX:92   y:210]
                    d:[SPPoint pointWithX:213  y:331]];

[path addSegmentWithA:[SPPoint pointWithX:213 y:331]
                    b:[SPPoint pointWithX:264 y:381]
                    c:[SPPoint pointWithX:361 y:216]
                    d:[SPPoint pointWithX:505 y:319]];

[path addSegmentWithA:[SPPoint pointWithX:505 y:319]
                    b:[SPPoint pointWithX:548 y:348]
                    c:[SPPoint pointWithX:672 y:290]
                    d:[SPPoint pointWithX:732 y:284]];
```

```
[path addSegmentWithA:[SPPoint pointWithX:732 y:284]
                    b:[SPPoint pointWithX:786 y:280]
                    c:[SPPoint pointWithX:824 y:362]
                    d:[SPPoint pointWithX:892 y:353]];

[path addSegmentWithA:[SPPoint pointWithX:892 y:353]
                    b:[SPPoint pointWithX:972 y:340]
                    c:[SPPoint pointWithX:right y:302]
                    d:[SPPoint pointWithX:right y:302]];
```

Codice 4.7: Esempio di percorso

Il metodo *addSegmentWithA: b: c: d:* aggiunge al path una curva di Bèzier con inizio nel punto **a**, punti di controllo **b**, **c** e punto terminale **d**. Come spiegato in precedenza i punti di inizio e fine nei segmenti intermedi coincidono.

Capitolo 5

Testing

Il collaudo del software (detto anche testing) è un procedimento che fa parte del ciclo di vita del software, utilizzato per individuare gli errori nel codice del software in corso di sviluppo.

Generalmente vengono fatti due tipi di test: uno durante la fase di realizzazione, viene fatto tramite l'iOS Simulator, in maniera molto frequente per verificare che l'introduzione di nuove funzionalità non dia problemi critici. Una seconda fase viene fatta al completamento dell'applicazione sia su iOS Simulator che su dispositivi hardware.

Durante la prima fase di test, ogni sua classe è stata testata in corso d'opera, tramite il simulatore iOS presente in XCode, vengono provati gli scenari più comuni durante l'utilizzo dell'applicazione e vengono valutate le prestazioni e l'utilizzo di memoria di quella singola pagina dell'applicativo. Non sono stati richieste particolari misurazioni delle prestazioni ma solamente di verificare che l'utilizzo non degradi le prestazioni dell'applicazione. Un esempio comune è quello di testare una pagina e monitorare l'allocazione della memoria, se questa cresce continuamente allora ci sono problemi di deallocazione che devono essere risolti.

La seconda fase di testing viene effettuata al termine dell'applicazione, quando tutte le funzioni sono operative e il design delle immagini di ogni pagina è definitivo. In questo modo si va a testare quello che realmente entrerà nei dispositivi degli utenti. Tutte le pagine che compongono l'applicazione vengono testate sequenzialmente, in questo modo vengono testate tutte le transizioni tra le varie pagine e il sorgere di eventuali errori.

Una cosa molto importante da fare, durante l'utilizzo globale dell'applicazione, è l'analisi delle prestazioni con *Instruments*, che consente di monitorare la memoria, riconoscere i thread pesanti e l'utilizzo della CPU.

Per quanto riguarda il testing dell'applicazione esposta in 4.3, i test specifici effettuati sono i seguenti:

- Verifiche sul consumo di memoria, essendoci una creazione continua di elementi grafici, le auto, è essenziale verificare la corretta deallocazione delle loro istanze.
- Verifiche sul percorso seguito delle auto; durante il loro tragitto le auto non devono collidere se non all'interno di un incrocio.
- Verifiche sul posizionamento dei semafori e dell'effettivo rilevamento di passaggi con il rosso all'interno degli incroci.
- Sincronizzazione e coerenza tra effetti visivi e effetti audio.
- Verifica del sistema di punteggio e della terminazione del gioco.

Una volta terminata la fase di testing da parte dello sviluppatore, se richiesto, l'applicazione può essere distribuita ad un ristretto gruppo di persone per effettuare un'ulteriore verifica di funzionamento. Per far ciò si utilizza un servizio esterno (TestFlight) che permette di caricare la propria applicazione e distribuirla a determinati utenti per ricevere feedback continuo. Questa fase di test prende il nome di *beta test*, l'applicazione viene utilizzata non solo dagli sviluppatori, ma anche da una ristretta cerchia di utenti come autori e clienti. In questo modo, vengono segnalati errori o richieste di modifiche prima della distribuzione nell'App Store ufficiale di Apple.

Prima di inviare l'applicazione ad Apple occorre effettuare delle verifiche sul software, in modo da sincerarsi che rispetti le regole elencate da Apple nello *Human Interface guidelines* (HIG). Alcune linee guida presenti nel documento sono le seguenti:

- Il software non abbia crash.
- Le componenti visuali siano intuitive.
- Uso di immagini ad una risoluzione adeguata.
- Coerenza dei pulsanti e delle loro relative funzioni.

Terminata questa fase l'applicazione viene inoltrata ad Apple, tramite Xcode, per la pubblicazione. L'applicazione viene testata dai tecnici Apple per verificarne il suo effettivo funzionamento. Superata la fase di verifica l'applicazione viene inserita nell'App Store e può essere scaricata da tutti gli utenti iOS dai propri dispositivi.

Capitolo 6

Conclusioni e sviluppi futuri

Dopo la fase di testing si è potuto verificare che la parte di software realizzata funziona correttamente. Nell'ambito della realizzazione dei mini-giochi è risultato essenziale l'utilizzo della libreria *Box2D* e della libreria di integrazione *SPPhysics*, infatti grazie a queste librerie lo sviluppo risulta semplice e immediato, e soprattutto flessibile a cambiamenti in corso d'opera del progetto.

Per quanto riguarda eventuali sviluppi futuri, nel caso ci sia richiesta dal mercato, è possibile rinnovare l'intero framework APP-KID utilizzando la nuova libreria *SpriteKit* messa a disposizione direttamente da Apple.

Questo framework, introdotto a Giugno 2013 con iOS7, consente allo sviluppatore di gestire la grafica, gli effetti e la fisica nella creazione di giochi 2D. Gli oggetti in movimento possono essere distinti dagli scenari, con la possibilità di implementare la fisica negli oggetti di gioco: controllare la gravità, effettuare movimenti inerziali o di massa, implementare effetti speciali come esplosioni e sparatorie.

A differenza di Sparrow e Box 2D che sono prodotti esterni, Apple con questa libreria offre strumenti ottimizzati per iOS e OS X integrati nativamente in XCode, che saranno sempre compatibili con le nuove versioni dei sistemi operativi e che probabilmente sono più efficienti, in termini risorse e velocità, di qualsiasi altra libreria prodotta da terze parti.

Ci sono da fare alcune considerazioni in merito all'adozione di questo framework. Innanzitutto può risultare molto oneroso in termini di risorse aziendali, passare da un framework tuttora supportato e moderno ad un framework completamente nuovo ancora da scoprire.

In secondo luogo, può risultare difficile la coesistenza di due framework nell'ambiente di lavoro, giungendo in una situazione in cui molte applicazioni

vengono aggiornate con il framework precedente, mentre le nuove applicazioni devono essere costruite da zero con il nuovo framework.

Concludendo, l'esperienza di tirocinio si è rivelata essere molto positiva portando l'acquisizione di competenze che spaziano anche al di fuori del progetto stesso: in primis l'approfondimento del linguaggio di programmazione Objective-C, l'uso di librerie utilizzate anche da grandi aziende, come Sparrow e Box2D.

Inoltre, sono state acquisite competenze sulla piattaforma di sviluppo XCode, prerogativa essenziale per lo sviluppo di applicazioni nel mondo Apple, e di tutti gli strumenti utili per gestione di un progetto in ambito aziendale. Molto importante è stato infine, comprendere le fasi di vita di un progetto: progettazione, implementazione e testing, essenziali per realizzare un ottimo prodotto.

Bibliografia

[1] A. Picchi, O*bjective-c 2.0 per iOS e Os X.*

[2] D. Mark , J. Nutting , J. LaMarche , F. Olsson, *Beginning iOS 6 Development: Exploring the iOS SDK, Apress.*

[3] Apple: iOS Developer Library *developer.apple.com/library/ios*

[4] Sparrow Framework Documentation *http://gamua.com/sparrow/help/*