Laboratorio di Applicazioni Mobili
Applicazione in ambiente iOS

Ivan Heibi

Università di Bologna, Bologna, Italia,
ivan.heibi@studio.unibo.it

Abstract. Lo scopo di questo progetto è la realizzazione del gioco di carte "La Briscola" in ambiente iOS, l'ambiente di sviluppo ed il linguaggio utilizzato sono rispettivamente Xcode e Swift. Il gioco prevede la presenza di una entità computer che si comporta e gioca autonomamente, quindi rappresentata come una intelligenza artificiale. Le capacità e potenzialità del giocatore lato PC potranno essere modificate prima di iniziare la partita, insieme ad alcune altre caratteristiche del gioco.

1 Introduzione

Il progetto è una attuazione pratica per uno dei campi informatici più in voga del momento ovvero quello delle applicazioni mobili, attualmente i due principali colossi che si contendono il mercato in questo ambito sono Google ed Apple con rispettivamente i sistemi operativi Android ed iOS, il loro successo e dovuto anche alla notevole praticità e comodità che essi offrono, soprattuto con la sempre maggiore diffusione di Smartphone.
Per questo progetto si è deciso di realizzare un'applicazione in ambiente iOS, a tale scopo si è fatto uso dell'ambiente di sviluppo integrato Xcode completamente mantenuto da Apple e distribuito esclusivamente su sistema operativo OS-X, il linguaggio di programmazione usato è Swift, una alternativa più performante e recente rispetto al più vecchio linguaggio Objective-C.
L'applicazione sviluppata è un gioco di carte, in particolare "La Briscola", un progetto di questo tipo ci permette di affrontare ed approfondire al meglio le problematiche e gli aspetti più importanti che uno sviluppatore di App deve conoscere. La versione attuale del gioco prevede la presenza di due giocatori, uno dei due viene mosso dal lato PC, il giocatore PC viene raffigurato come una intelligenza artificiale in grado di prendere delle decisioni in base a quello che accade sul tavolo di gioco, quindi le sue percezioni sono prese in base alle azioni intraprese dai membri della partita, che di conseguenza andranno ad alterare il suo mondo e le sue conoscenze, il computer potrà far riferimento alla sua memoria per prendere le decisioni che ritiene essere più benevoli al suo gioco, in aggiunta è stato introdotto un fattore di correttezza del giocatore, quindi un giocatore poco corretto avrà la capacità di intraprendere decisioni di gioco sfruttando la sua furbizia che gli permetterà di avere a priori la conoscenza delle carte del suo avversario prima di giocare. Sono state pienamente implementate tutte le regole e caratteristiche della Briscola, e quindi il comportamento del giocatore PC seguirà una strategia di gioco che rispecchia la logica della briscola.
Cominceremo discutendo le caratteristiche e gli aspetti arcitetturali generali del progetto, spiegando quelle che sono state le parti più fondamentali sorte al momento della sua costruzione e di conseguenza quale soluzioni si è voluto adottare, verrà spiegata l'iterazione tra l'utente ed il sistema anche attraverso degli esempi di casi d'uso, dopodiché si passerà ad una trattazione a livello implementativo del progetto specificando gli aspetti tecnici più rilevanti, concluderemo con alcune considerazioni finali e le eventuali possibili estensioni future che possono essere introdotte.

2 Architettura generale

Il pattern architetturale utilizzato nella realizzazione di questa applicazione è il Model-View-Controller (MVC), l'importanza di questo pattern sta nella sua capacità di poter separare i vari compiti fra i componenti software, ovvero tra la logica applicativa e l'interfaccia visiva che va a carico della view.
Esistono due finestre diverse, la prima schermata che l'utente vede nel momento in cui il caricamento dell'applicazione è terminato e parte l'esecuzione

dell'applicazione è una interfaccia che ci permette di poter inizializzare i valori e parametri del gioco, una volta fatto questo l'utente potrà premere uno specifico bottone che gli permetterà di passare alla seconda schermata, ovvero il gioco in se, e di poter quindi cominciare a giocare. Le classi che sono state implementate per descrivere i vari oggetti del gioco sono:

Card :

I componenti principali del gioco sono le carte, quindi si è deciso di raffigurare la carta come un oggetto specifico, i parametri fondamentali che ogni oggetto di tipo Card possiede sono :

1. rank: ovvero il suo valore in punti in base alle regole della briscola. (Asso = 11pt, Tre = 10pt, Re = 4pt, Cavallo = 3pt, Donna = 2pt e il resto delle carte vale 0)
2. number: il numero della carta, l'asso vale 1 ed il numero massimo è il Re che ne vale 10
3. suit: il seme della carta
4. inHand: un booleano che assume valore "true" nel caso in cui la carta è in mano ad un giocatore
5. playerHand: nel caso in cui la carta è in mano ad un giocatore questo parametro indicherà l'Id del giocatore che ha in mano la carta
6. inHandPosition: la precisa posizione della carta in mano al giocatore (se essa è effettivamente in mano ad un giocatore)
7. textureUp, textureDown: sono rispettivamente l'immagine visualizzata nel momento in cui la carta è scoperta, e quella nel caso in cui la carta è coperta.

Table :

Rappresenta il tavolo di gioco e tutto quello che accade su di esso, in aggiunta verrà interrogato nel momento in cui si ha il bisogno di applicare alcune regole che riguardano la briscola, i parametri di questa classe sono :

1. briscola: il seme della briscola nel gioco
2. cards: sono le carte presenti sul tavolo di gioco che ogni giocatore a turno lancia sul tavolo
3. score: lo score di ogni giocatore della partita
4. turnOrder: l'ordine di gioco per la mano attuale

ComputerMind :

È una classe che raffigura il cervello del giocatore lato macchina, che come qualunque altro giocatore normale reale avrà delle percezioni su quello che avviene durante la partita, che andranno a modificare il suo stato mentale ovvero alcuni specifici parametri :

1. briscola: il seme della briscola attualmente in gioco
2. cardsOnHand: le carte in mano
3. memory: la sua memoria, che viene definita come l'insieme delle carte che secondo il PC non sono ancora state giocate
4. myId: l'id del giocatore
5. turnOrder: da quale giocatore parte l'attuale mano
6. cardsCheated: l'insieme delle carte che la macchina è riuscita a vedere con l'inganno

Come già detto MVC è il modello architetturale applicato, data la presenza di due finestre diverse avremo due controller diversi per ognuna di esse, ed avremo la possibilità di spostarci tra i due con un apposito navigatore, un Navigation-Controller. L'idea base di un NavigationController è quella di mantenere una coda di Views visitati, che verrano inserite ed estratte a seconda delle operazioni di navigazione tra views che l'utente fa. Una volta impostati i parametri del gioco, un'apposito bottone ci permetterà di spostarci alla seconda View, quella dedicata alla schermata del gioco, la Figura 1 mostra in generale lo scheletro del programma con l'utilizzo del navigation controller.
Di seguito descriviamo in dettaglio i nostri viewController :

initViewController :
Permette la visualizzazione di un menu di impostazioni che ci è consentito fare prima di iniziare il gioco, le impostazioni che vanno definite sono :

1. Nome : il nome del giocatore.
2. Game Speed : la velocità del gioco.
3. Agent Memory : la capacità di memorizzazione dell'avversario (PC)
4. Cheating Factor : quanto l'avversario di fronte (PC) è scorretto

GameViewController :
Visualizza e mette in scena il gioco, quindi avrà il compito di richiamare alcuni metodi presenti nel modulo che si occupa di creare l'intera scena.

L'ultima classe è la GameScene, questa andrà a costruire l'intera scena del gioco seguendo la sua logica e le sue regole, quindi a il modello e la base principale che verrà visualizzata all'interno della view di GameViewController.

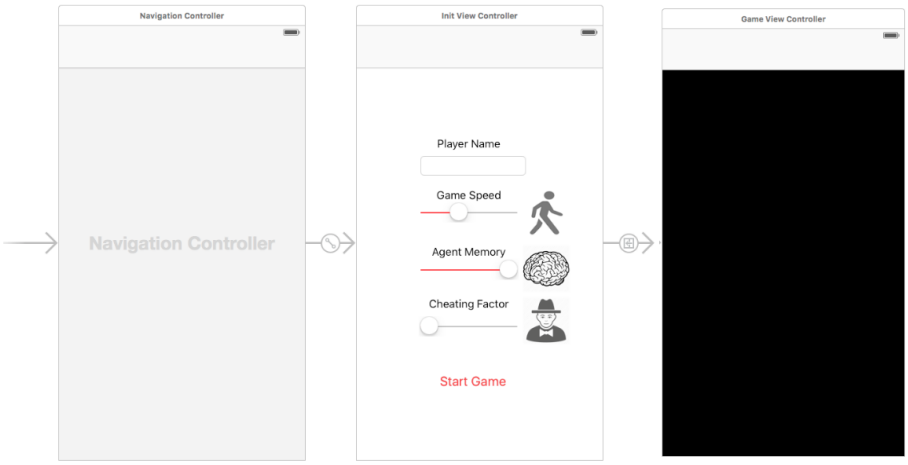

Fig. 1. Scheletro dell'applicazione

2.1 Interfaccia del gioco

La Figura 2 mostra ciò che l'utente vede nel momento in cui viene avviata la partita, come possiamo vedere le nostre tre carte in mano sono scoperte (a differenza di quelle dell'avversario), per poter giocare una carta è necessario toccarla e trascinarla al centro del tavolo (in una certa area rettangolare), se la carta non viene posizionata correttamente al centro del tavolo essa riprende la sua posizione di partenza iniziale (in mano al giocatore). Una volta che anche il nostro avversario (computer) avrà fatto la sua mossa, il vincente della mano si aggiudicherà le carte, ed esse verranno spostate affianco a lui, e automaticamente verrà estratta una carta dal mazzo che prenderà la posizione dell'ultima carta giocata.

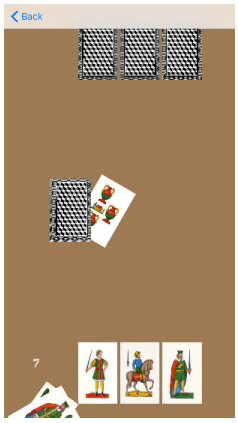

Fig. 2. Schermata del gioco

2.2 Interazione con l'utente e casi d'uso

Prima di iniziare una partita la finestra che comparirà è quella di impostazione del gioco, nella quale sarà richiesto di inserire: il proprio nome, la velocità del gioco, la capacità di memorizzazione dell'avversario (PC) ed il suo livello di scorrettezza. Tutte queste impostazioni sono controllate tramite oggetti di tipo slider, con dei valori che vanno da un min-max (Per esempio la velocità va da 1-9). Una volta fatto questo si clicca sul button 'Start Game' per visualizzare il tavolo di gioco ed iniziare la partita.
Ad ogni giocatore vengono date 3 carte, nel momento in cui tocca a noi giocare si sceglie la carta da giocare e la si trascina al centro del tavolo. Le regole del gioco e la modalità di assegnazione dei punti sono quelle classiche della briscola. Di seguito nella Figura 3 mostriamo il diagramma dei casi d'uso per quanto riguarda il lato utente, le operazioni che il giocatore può fare dipendono dai

touches che esegue (ovvero il trascinamento delle carte), dopo aver deciso di giocare una carta ed averla gettata sul tavolo di gioco, viene fatto un controllo sulle carte presenti sul tavolo, nel caso in cui tutti i giocatori abbiano già giocato si calcola il vincitore della mano e si ridistribuiscono delle nuove carte ai giocatori e si passa al turno del giocatore successivo. Nel caso in cui non ci sono più carte da giocare allora la partita giunge al termine e viene dichiarato il vincitore.

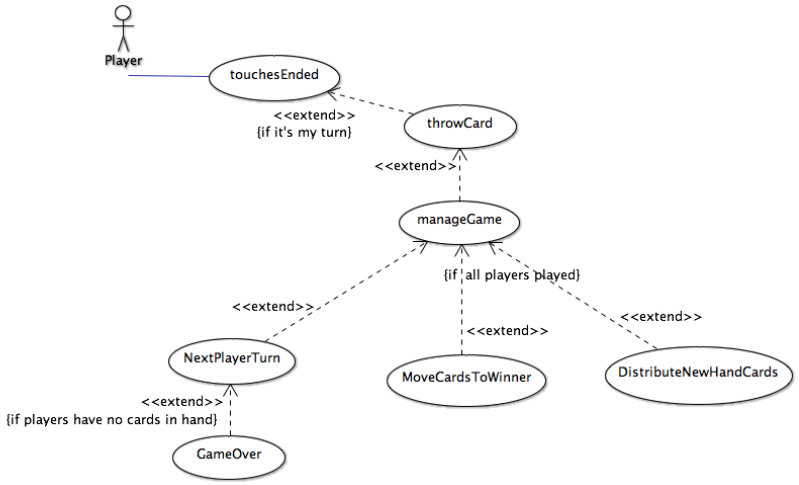

Fig. 3. Diagramma dei casi d'uso per il lato utente

2.3 Intelligenza artificiale e casi d'uso

Il nostro avversario (il giocatore PC) sarà una istanza della classe ComputerMind. Per simulare al meglio una entità di questo tipo si è fatto riferimento al concetto di "Agente intelligente", ovvero una entità che è in grado di percepire quello che sta accadendo nell'ambiente nel quale si trova e di eseguire delle azioni adeguate in base agli input percepiti.
In questo caso l'ambiente nel quale l'agente si trova è il gioco, e tutto quello che lui riesce a percepire è dovuto ad una simulazione del senso della vista (cosi come per un essere umano), quindi le informazioni che può ottenere sono quelle presenti sul tavolo (oggetto Table) e le carte in suo possesso. In aggiunta le sue informazioni rispetto a quelle di un essere umano possono essere ampliate con l'ausilio di due capacità importanti :

1. Memoria: un oggetto ComputerMind possiede al suo interno un vettore che rappresenta le carte che sono già state giocate sul tavolo, un PC con una memoria avanzata avrà la capacità di ricordare un maggior numero di carte rispetto a tutte quelle che sono state giocate.

2. Fattore di scorrettezza: un ComputerMind possiede in aggiunta un vettore di 3 elementi, questi elementi sono le carte che esso è riuscito a vedere in maniera furbesca dalla nostra mano, più questo fattore è alto è maggiore sarà la sua capacità nel scovare le nostre carte in maniera scorretta.

L'avversario può capitare in due scenari diversi nel momento in cui è il suo turno:

1. Il primo a giocare nella mano corrente : in questo caso prende in considerazione le carte che è riuscito a sbirciare all'avversario (nel caso di esito positivo) e le carte che sono già state giocate in precedenza, quindi sceglie la carta che ha la possibilità maggiore di vincita rispetto a quelle che ha in mano. Da notare che una strategia di questo tipo pensa solo a massimizzare la vincita locale, senza prendere in considerazione scenari futuri.
2. Il secondo a giocare : prima di scegliere deve controllare la carta che è stata giocata, nel caso in cui la carta sul tavolo è una carta che non può battere, allora deve minimizzare la perdita di punti giocando la carta in suo possesso che ne vale meno. Altrimenti se è in grado di vincere, la scelta della carta è fatta in questo modo :
   (a) Se posso vincere solo con una briscola gioco quella meno potente
   (b) Se posso vincere con una carta dello stesso seme allora scelgo la carta che mi da più punti

   In entrambi i casi se la vincita è pari a 0 punti, allora controllo l'eventualità di perdere la mano con 0 punti sul tavolo.

Lo scenario appena descritto può essere sentitizato nella Figura 4 sottostante che rappresenta i casi d'uso del computer.

3 Aspetti implementativi

Il modello architetturale generale seguito per questo progetto come già detto è il MVC (Model View Controller), per come è stato costruito questo lavoro le classi di tipo UIViewController ci permetteranno di gestire al meglio le view dell'applicazione, quindi una classe di questo tipo avrà il compito di aggiornare il contenuto delle view in base ai cambiamenti che avvengono da parte dell'utente. Abbiamo due diverse classi di questo tipo che sono state collegate tramite un UINavigationController che ci permette di spostarci in modo gerarchico tra le view. Viene infatti specificata la finestra del menu di impostazione come prima view, questo viene fatto impostando il parametro root-view-controller con il valore initViewController.
Dato che i parametri del gioco vengono impostati all'interno della prima view, occorre specificare nel metodo prepareForSegue() definito all'interno del initViewController quali sono i parametri da passare alla seconda view (GameViewController), le operazioni scritte in questo metodo danno la possibilità di accedere a le variabili di un'altra view e di reimpostarli con i valori ottenuti localmente.
Possiamo spigare le scelte più rilevanti a livello implementativo dividendo il discorso in due rami: grafico/animazione e quello logico, andiamo a trattare in maniera singolare i due aspetti.

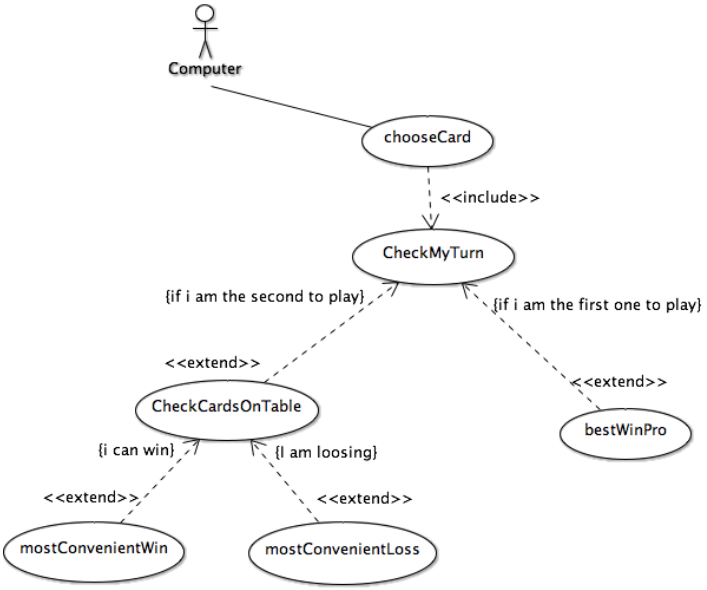

Fig. 4. Diagramma dei casi d'uso per il lato PC

3.1 Grafica e animazione

L'idea per la costruzione di un gioco animato di questo tipo è quella di utilizzare dei nodi di tipo SpriteKit (SKNode), ovvero dei componenti che se messi all'interno di un oggetto di tipo scena (SKscene) possono essere animati. Per poterli integrare all'interno della nostra view essa deve essere trasformata in una view che permetta la visualizzazione di una scena che contiene nodi di tipo SpriteKitNodes, ciò di cui abbiamo bisogno dunque è una SKView.
La classe GameScene a questo proposito è una classe di tipo SKScene, mentre gli elementi di tipo SKNodes da inserire al suo interno sono le carte. Una volta inseriti tutti gli oggetti di tipo SpriteKit avremo la possibilità di effettuare operazioni animate su di esse, come per esempio: spostamento in una nuova posizione, rotazione e il livello di profondità. Tutte queste operazioni verrano utilizzate in maniera opportuna in diverse fasi e movimenti del gioco.
Per rendere l'animazione ancora più realistica si è fatto uso di oggetti di tipo SKAction che permettono un cambiamento grafico del nodo in maniera fluida e controllata con un certo intervallo di tempo.

3.2 Logica del gioco

I metodi che vanno a definire la logica e il modello concettuale del gioco sono presenti all'interno della classe GameScene(). Le fasi del gioco sono diverse:

Inizializzazione, durante il gioco e la terminazione. Possiamo trattare ognuna di queste fasi separatamente per far comprendere meglio le scelte più rilevanti fatte.

Inizializzazione: La prima cosa da definire sono le strutture dati da utilizzare: Il mazzo di carte è un vettore di oggetti di tipo Card che ha dimensione pari a 40 (10 carte per ogni seme), un vettore di tre Card per ogni giocatore che quindi rappresenta le carte che possiede in mano, un vettore di giocatori umani ed uno per i computer (questo è stato fatto per rendere più agevole una futura espansione del gioco a più partecipanti).
A questo punto la fase di preparazione al gioco consiste nel mescolare le carte (dealCards()) che viene fatta con lo spostamento delle carte in posizioni nuove casualmente scelte all'interno dell'array, dopodiché vengono distribuite tre carte a ciascun giocatore, ovvero viene riempito il vettore delle carte in mano con oggetti di tipo Card che sono stati rimossi dall'array del mazzo. Fatto questo vengono impostate le informazioni che tutti i giocatori vedono, ovvero i parametri della Table (es: ordine di gioco e la briscola), e vengono infine creati i giocatori PC di tipo ComputerMind inizializzando le loro prime informazioni (percezioni).

Durante la fase di gioco: In questa fase tutto quello che facciamo è catturato da eventi UIEvent di tipo UITouch. Per decidere che operazione intraprendere bisogna controllare la posizione nel momento in cui il touch di un oggetto Card è concluso, per esempio viene fatto un throwCard sul tavolo di gioco se la posizione è adeguata.
Le nostre azioni sono un trigger per una risposta automatica da parte del computer che al suo interno possiede i metodi che definiscono la sua strategia di gioco. Tutte le azioni fatte in questa fase consistono in un continuo aggiornamento dei vettori definiti nel gioco (quindi faremo operazioni di tipo pop e push).

Terminazione: Se tutti gli array presenti nel sistema (carte sul tavolo e quelle in mano) sono vuoti, allora viene decretato un vincitore dalla gameOver(), che controlla lo score di ciascun giocatore interrogando la Table.

4 Conclusioni e sviluppi futuri

L'obbiettivo di questo lavoro è stato quello di realizzare un'applicazione mobile per ambiente iOS, per rendere il lavoro intrigante si è deciso di creare un gioco di carte, in particolare la briscola.
Questo progetto ha dato l'opportunità di approfondire meglio le tecniche di sviluppo nel campo di iOS e le strategie di programmazione di questo campo, in oltre la scelta di progettare un gioco ha permesso una trattazione di alcuni aspetti molto importanti dal punto di vista grafico e di come integrare scenari animati all'interno di una applicazione.

Un'altro aspetto molto importante e intrigante è stato quello di creare un avversario che possa comportarsi autonomamente, una specie di intelligenza artificiale. La realizzazione di un ente intelligente di questo genere è un procedimento che cerca di simulare un comportamento razionale in grado di percepire ed arricchire le proprie conoscenze e di agire di conseguenza con lo scopo di raggiungere il suo obbiettivo, ovvero vincere la partita di briscola. Affinché esso possa raggiungere il suo scopo è stata definita una strategia di gioco che esso seguirà per raggiungere il suo scopo .
Un'applicazione di questo tipo lascia molto spazio alla creatività e a tante possibili feature che possono essere aggiunte, in modo particolare esistono due punti molto interessanti sui quali si potrebbe dedicare un lavoro futuro per una estensione del programma :

1. Numero di giocatori : attualmente il gioco è una partita fatta tra due giocatori soltanto, detto ciò il codice è stato comunque predisposto ad essere ampliato con un numero più elevato di partecipanti, questo è stato fatto con l'utilizzo di strutture dati dinamiche che permettono un aggiornamento futuro nel caso si voglia aumentare il numero di giocatori. Va ricordato che avere più giocatori in questo caso richiederebbe al programma di essere compatibile ad una modalità online con più giocatori, data la difficoltà di far giocare più giocatori sulla stessa macchina.
2. Migliorare la strategia di gioco del PC: possiamo perfezzionare la strategia di gioco del PC da molti punti di vista:
   (a) Ricordare lo score: lo scopo principale è vincere, e lo si fa quando si superano i 60 punti di score, vista con la logica di agente intelligente questo vuol dire che esso ha come 'Goal' proprio quello di avere punti > 60, per ottenere un risultato di questo tipo esso deve avere presente lo score in tempo reale della partita, quindi la si può vedere come una abilità extra nel memorizzare il risultato. Attualmente data l'assenza di questa abilità del PC, possiamo trovarci in una situazione nella quale esso preferisca perdere dei punti senza rendersi conto che cosi facendo ha compromesso definitivamente la sua possibilità di vincita (nel caso in cui l'avversario ha superato i 60 punti)
   (b) Previsione futura: il comportamento attuale del PC è locale, ovvero prende in considerazione la situazione attuale e cerca di trarre il massimo vantaggio da essa, viene però completamente ignorato lo stato futuro nel quale andrà a trovarsi una volta fatta una certa mossa. Per esempio in una certa situazione potrebbe essere utilizzata una briscola per vincere una mano con pochi punti, e cosi facendo trovarsi senza alcuna briscola nella mano successiva e in difficoltà con la scelta da fare.
3. Più strategie di gioco per il PC: attualmente il nostro avversario segue una sola ed unica strategia di gioco, sarebbe interessante poter integrare più possibili strategie che esso può applicare in maniera anche casuale, oppure a seconda di come si comporta l'avversario di fronte.

References

1. Start Developing iOS Apps (Swift),
   iOS Developer Library
   `https://developer.apple.com/library/ios/referencelibrary/GettingStarted/DevelopiOSAppsSwift/`

2. Swift Tutorials, Code Samples, References and more.
   `http://www.learnswift.tips/`