

UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingeniería

**Desarrollo de juego de memoria de 3 secuencias para el
smartwatch Apple Watch en Swift 2**

Proyecto Técnico

Francisco Foyain Lara

Ingeniería de Sistemas

Trabajo de titulación presentado como requisito
para la obtención del título de
Ingeniero de Sistemas

Quito, 21 de diciembre de 2015

UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ
COLEGIO DE CIENCIAS E INGENIERIA

**HOJA DE CALIFICACIÓN
DE TRABAJO DE TITULACIÓN**

**Desarrollo de juego de memoria de 3 secuencias para el smartwatch Apple
Watch en Swift 2**

Francisco Foyain

Calificación:

Nombre del profesor, Título académico

Fausto Pasmay , M.Sc.

Firma del profesor

Quito, 21 de diciembre de 2015

Derechos de Autor

Por medio del presente documento certifico que he leído todas las Políticas y Manuales de la Universidad San Francisco de Quito USFQ, incluyendo la Política de Propiedad Intelectual USFQ, y estoy de acuerdo con su contenido, por lo que los derechos de propiedad intelectual del presente trabajo quedan sujetos a lo dispuesto en esas Políticas.

Asimismo, autorizo a la USFQ para que realice la digitalización y publicación de este trabajo en el repositorio virtual, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Firma del estudiante: _____

Nombres y apellidos: Francisco Javier Foyain Lara

Código: 00100955

Cédula de Identidad: 1714439393

Lugar y fecha: Quito, 21 de diciembre de 2015

RESUMEN

La aplicación PanchoMania es un juego desarrollado en lenguaje de programación Swift 2 para el nuevo reloj inteligente de Apple llamado Apple Watch, el juego consta de 3 secuencias de juegos de memoria que cualquier persona puede jugar. El primero es un juego de seguir series de colores, el segundo juego es de encontrar tarjetas pares y por ultimo el tercer juego hay que aplastar números en orden descendente antes de que se acabe el tiempo. Con los juegos se busca ayudar a los niños y adultos mayores para que mejoren su memoria y atención inmediata. Se realizo el juego exitosamente logrando cargar el juego al AppStore de Apple llegando a tener 11 descargas en 4 días. La nueva infraestructura y nuevo lenguaje de programación fue una motivación para lograr el objetivo.

Palabras clave: AppleWatch, Apple, AppStore, Swift2, iOS, iPhone, PanchoMania.

ABSTRACT

The PanchoMania application is a game developed in Swift 2 programming language for the new smart watch of Apple called Apple Watch. The game consists of 3 sequences memory games that anyone can play. The first game is a set of continuing series of colors. The second game is to find pairs of cards and finally the last game the numbers must be array in descendent order before the time runs out. The games seek to help children and elderly to improve memory and immediate attention. Successfully managing the load of the game to the Apple AppStore getting to have eleven downloads in four days took place. The new infrastructure and programming language was a motivation to achieve the goal.

Key words: AppleWatch, Apple, AppStore, Swift2, iOS, iPhone, PanchoMania.

AGRADECIMIENTO

Sin Dios, nada de esto seria posible. Todo tiene un sentido en la vida, el camino esta puesto por el para cada uno.

Agradezco a mis padres: Francisco Foyain E y Cecilia Lara V, les agradezco por nunca perder la esperanza en mi, siempre darme apoyo, aliento y también regañadas. Les agradezco por ponerme en la mejor universidad del país, aunque costosa, me llevo conocimientos tanto académicos como humanos, así como los mejores años de mi vida. Les quiero mucho.

Agradezco a mi hermana, Loquita, como puede ser que de alguien de tan corta edad me pueda enseñar tanto a mi. Te agradezco por cada risa y motivación diaria que me has brindado. Cada abrazo y beso me lleno de energía para seguir adelante. Te quiero mucho.

Agradezco a mi novia, Daniela Veintimilla, gracias por ser el empujón diario durante los últimos 3 años de mi carrera. Nunca perdiste la fe en mi, siempre con tu comentario, eres un crack yo se que tu puedes, me llenaste de motivación. Te agradezco por demostrarme que el cariño puede motivar de cualquier manera. Te agradezco también por tus ayudas en clases de colegio general. Siempre recordare que tu estabas mas estresada que yo en mis pruebas, trabajos y proyectos.

Agradezco a David Villacis, Oscar Cortes, Danny Velasquez, Pablo Jarrin, amigos y compañeros, les agradezco por su amistad, motivación y ayuda en cada momento que lo necesite.

Profesores, personal de mantenimiento, y resto de comunidad USFQ. Gracias por hacer de la USFQ la mejor universidad del país. Gracias por todas las enseñanzas brindadas, así como las regañadas que fortalecieron mas mi carácter, nada en la vida es fácil y como tal ustedes demostraron y ayudaron a formar mi carácter para ser la persona que soy hoy en día.

DEDICATORIA

Este logro va dedicado a mis padres, por ser la estructura básica de lo que soy yo, los valores, conocimientos, risas, lloros, discusiones, regañadas y castigos han hecho lo que hoy es Francisco Foyain Lara. A mi hermana, una niña tan dulce, llena de cariño y valores que me ha dado el cariño necesario para seguir adelante. A mi novia, Daniela Veintimilla, sin la que este logro no podría haberse dado, te lo dedico a ti por el cariño, paciencia y amor que me brindaste, en la universidad nos conocimos, y ya vamos 3 años juntos. Y en especial se la dedico a mi Abuelo, (+) Héctor Lara, abuelito cuanto me hubiera gustado que estés este momento a mi lado, y festejar juntos a tu primer nieto graduado. Se que desde el cielo me acompañaras hoy y cada día de mi vida.

Les quiero mucho a todos. Gracias

Francisco Foyain

TABLA DE CONTENIDO

ÍNDICE DE FIGURAS.....	10
INTRODUCCIÓN.....	11
ANTECEDENTES.....	12
JUSTIFICACIÓN.....	14
OBJETIVOS.....	15
Objetivo General.....	15
Objetivos Específicos.....	15
SOPORTE TEÓRICO.....	17
Swift2.....	17
Digital Crown.....	17
Apple Watch.....	17
WatchKit.....	17
WatchOS2.....	18
iOS 9.....	18
ViewController para iOS.....	18
Page Content Controller iOS.....	19
WKInterfaceController.....	19
Storyboard.....	19
Conexiones IBOutlet y IBAction.....	19
AppleStore.....	20
iTunesConnect.....	20
TestFlight.....	20
DESARROLLO DEL JUEGO PANCHOMANIA PARA WATCHOS Y MANUAL DE USUARIO PARA IOS.....	21
AppleWatch.....	21
Diseño de la aplicación.....	21
MenuInicialController.....	22
OpcionController.....	24
NivelController.....	25
RepeticionesController.....	26
JuegoColoresController.....	27
JuegoTarjetasController.....	32
JuegoNumerosController.....	37
APLICACIÓN IOS PARA IPHONE.....	40
Estructura de la aplicación de iOS.....	40
Esquema de uso.....	41
PageViewController.....	42
PageContentViewController.....	42
ViewController.....	43
Publicación en el AppStore.....	44
Cuenta de desarrollador.....	44
Creación de perfiles, certificados de desarrollo e identificador.....	45
iTunes Connect.....	46
Archivo y carga desde Xcode.....	47
Error de carga de compilación.....	49

Carga exitosa a iTunes Connect.....	50
Aprobación de la aplicación	51
RESULTADOS OBTENIDOS	52
Encuesta a niños.....	52
Encuesta a los adultos mayores	52
Estadísticas de descarga de la aplicación.	52
CONCLUSIONES	55
REFERENCIAS BIBLIOGRAFICAS	57
ANEXO A: MENUINICIALCONTROLLER.....	59
ANEXO B: NIVELCONTROLLER.....	62
ANEXO C: REPETICIONES CONTROLLER.....	65
ANEXO D: OPCIONCONTROLLER.....	68
ANEXO E: JUEGOCOLORES CONTROLLER.....	69
ANEXO F: JUEGOTARJETASCONTROLLER	77
ANEXO G: JUEGONUMEROSCONTROLLER.....	90
ANEXO H: VIEWCONTROLLER	97
ANEXO I: PAGECONTENT VIEWCONTROLLER.....	100
ANEJO J: INTERFACE CONTROLLER	101
ANEKO K: MAINSTORYBOARD	102
ANEXO L: MANUAL DE USUARIO CREADO PARA EL IPHONE.....	105

ÍNDICE DE FIGURAS

Figura 1 Diseño de uso del juego	22
Figura 2 Diseño de la clase MenuInicialController	23
Figura 3 Funcionalidad MenuInicialController	24
Figura 4 Diseño de la clase OpcionController	25
Figura 5 Funcionalidad opcionController	25
Figura 6 Diseño de la clase NivelController	26
Figura 7 Diseño de la clase RepeticionesController	27
Figura 8 Diseño de la clase JuegoColoresController	27
Figura 9 función willActivate	29
Figura 10 función iniciarJuego	30
Figura 11 función JuegoColoresController	32
Figura 12 clase JuegoTarjetasController	33
Figura 13 función JuegoTarjetasController	34
Figura 14 clase JuegoNumerosController	37
Figura 15 Trayectoria juego de números	38
Figura 16 Funcionalidad Aplicación iOS	41
Figura 17 Clase PageContentViewController	42
Figura 18 Clase PageContentViewController	43
Figura 19 Licencia de desarrollo	44
Figura 20 Certificados	45
Figura 21 Perfil de distribución	45
Figura 22 Identificadores	46
Figura 23 Registro en Itunes Connect	46
Figura 24 Registro en Itunes Connect	47
Figura 25 Organizer de Aplicación	48
Figura 26 Carga de Aplicación	48
Figura 27 Carga de Aplicación 2	49
Figura 28 Alpha Channel Remover	50
Figura 29 Compilaciones de iOS	51
Figura 30 Aprobación de PanchoMania.	51
Figura 31 Detalle de estadísticas.	53
Figura 32 Dispositivos que han descargado.	53
Figura 33 Instalación por día.....	54

INTRODUCCIÓN

En el presente trabajo técnico se realizó el desarrollo de un juego de memoria para el reloj inteligente Apple Watch. Las 3 secuencias que contiene el juego son: juego de colores en el cual hay que seguir un orden aleatorio en el que se presentan colores en pantalla, posterior a esto se sigue a un juego de tarjetas, en el que hay que encontrar las tarjetas pares iguales entre 6 tarjetas y por último se pasa a un juego que hay que buscar los números que se presentan de mayor a menor. Se utilizó el lenguaje de programación Swift 2 para el desarrollo de todo el juego. Concluyendo con la carga del juego a la tienda virtual de aplicaciones AppStore, en la que Apple aprobó el juego y actualmente se encuentra a la venta en 12 países del mundo gratuitamente. La programación móvil es un tema innovador y de carácter obligatorio actualmente para cualquier ingeniero de sistemas. Por lo que conocer sobre este nuevo lenguaje de programación e infraestructura ayuda a tener mayor experiencia para el campo laboral en el futuro.

ANTECEDENTES

Actualmente la programación de aplicaciones y juegos para teléfonos móviles, tablets y relojes inteligentes o smartwatch son la nueva tendencia de desarrollo. El Apple Watch es un reloj inteligente de la marca Apple que se lo lanzo mundialmente el día 10 de Abril del 2015, para competir con el reloj preexistente de su rival Android. Este reloj llevo con sistema operativo WatchOS el cual basa su lenguaje de programación como todos los dispositivos de iOS en Swift. Posterior a su lanzada y distribución mundialmente el Apple Watch obtiene una actualización con la llegada del iPhone 6S. Conocido como WatchOS 2. Con este nuevo sistema operativo llega un nuevo lenguaje de programación Swift 2 el cual tiene mejoras para los desarrolladores como son el uso del *Crown* del reloj, el cual es un botón nuevo creado por Apple que cumple la función de seleccionador en varias aplicaciones, permite girar este botón como tornillo con suavidad y de esta manera seleccionar, hacer zoom entre otras cosas además del *Crown* se incluyo el soporte de *Force Touch* el cual es un nuevo sistema que se integro en la pantalla del reloj y desde el iPhone 6S el cual le da una capacidad de botón a la pantalla del dispositivo si se aplasta un poco mas fuerte de lo común sobre el touch del mismo, conjunto con esto se puede realizar vibraciones personalizadas para eventos, entre otros. Lo cual hace que las aplicaciones y juegos que se desarrollen en este dispositivo electrónico tengan mejor alcance y capacidad para el cliente final.

Actualmente en el mercado ecuatoriano el desarrollo de aplicaciones para iOS es algo que no se lo realiza con frecuencia por el costo de la licencia de desarrollo y la complejidad de tener el ambiente de desarrollo, es necesario tener una computadora Mac para poder instalar el IDE propio del lenguaje objetivo C, Swift o Swift 2. Con la llegada de los relojes Apple Watch con WatchOS2 este limitante no se lo ve alterado. Las aplicaciones que se

encuentran hoy en el AppStore para el reloj inteligente son de acceso fácil, sin complejidad de uso, poca carga visual y con brevedad de ejecución. La mayoría de aplicaciones son extensiones de aplicaciones previas en el teléfono que demuestran poca información, los juegos son juegos que muestran solo flujos de conversación o de elección y no una continuidad.

JUSTIFICACIÓN

El uso cotidiano de las nuevas tecnologías han convertido a los dispositivos móviles en una necesidad para cualquier edad de las personas. Desde los mas pequeños hasta los adultos mayores están acostumbrándose a usar la tecnología de diferentes maneras. Para comunicación, difusión, estudio, investigación, ocio son algunas de las formas que la gente usa sus teléfonos celulares o tablets. Ahora con la llegada de los relojes inteligentes esta interacción con la tecnología se extiende hasta la muñeca de la persona. Ya no se necesita sacar el celular del bolsillo para chequear el calendario, recordatorios, emails o incluso un mail. De la misma manera se puede jugar en juegos hechos para estos dispositivos que presentan bajo nivel de dificultad, gran brevedad y facilidad de uso.

Es por esto que se propone desarrollar un juego de memoria para el Apple Watch, en el cual cualquier persona lo puede jugar con sus diferentes niveles de dificultad pero que va enfocado a los niños y adultos mayores. Los primeros, en base al juego se puede contribuir a la formación de la memoria de los niños con un juego inofensivo, con imágenes hecho para todos e incluso a hacer operaciones matemáticas sencillas de buscar el numero mayor. Y en segundo lugar para los adultos mayores. Los cuales necesitan tener mas cantidad de interacción con la memoria instantánea para así de esta manera evitar la aparición de enfermedades como el Alzheimer. El juego posee imágenes grandes y colores vistosos para que una persona de cualquier edad pueda jugar con este. El lenguaje de programación Swift 2 al ser algo nuevo tendrá complejidad y poca documentación lo que tendrá una dificultad extra realizar el juego para un dispositivo nuevo como es el reloj inteligente.

OBJETIVOS

Objetivo General.

Realizar un juego en el lenguaje de programación Swift 2 para el reloj inteligente Apple Watch utilizando recursos propios de UIKit así como recursos propios de hardware con lo que se obtendrá 3 juegos secuenciales: seguir el color, encontrar las tarjetas iguales y aplastar los números descendientemente.

Objetivos Específicos.

- Incluir recursos propios de hardware en el transcurso del juego como son las vibraciones como notificaciones de eventos.
- Incluir recursos de hardware del Crown para elegir el nivel.
- Elegir la dificultad del juego que varia en su acción para cada juego, en el juego 1 será el tiempo que se demore de cambiar de color a color, en el juego de tarjetas el tiempo que se puede visualizar las tarjetas y por ultimo en el juego de números el tiempo que se tiene para aplastar los números.
- Las repeticiones afectaran solo al primer juego, mientras mas repeticiones se elija mas complicado será pasar al siguiente juego pero se podrá obtener mayor puntaje alto.
- El puntaje alto será un valor que se guarda como recurso de memoria del juego. Este valor se cambia solo si se pasa el puntaje alto anterior.
- Utilizar timers para iniciar cada juego y dar mensajes de motivación al jugador.

- Utilizar un timer diferente para el tiempo que le falta al jugador en el ultimo juego.
- Crear 5 interface controllers para el uso del juego del Apple Watch.
- Utilizar 2 view controllers, uno que es page view controller y otro view controller para la aplicación que se presentara en el celular.
- Crear icono propio del juego y "splashscreen" para la aplicación del celular
- Registrar el proceso de subir el juego al AppStore de Apple y todos los detalles que conlleva esto como corrección de errores, requisitos necesarios, creación de perfiles de distribución así como la creación del certificado de firma de la aplicación.

SOPORTE TEÓRICO

Swift2.

Swift es el lenguaje sucesor de objective-c presentado en un keynote de Apple el 2 de Junio del 2014 (Medina, 2014). El cual es un lenguaje de programación orientado a objetos, compilado y basado en enunciados. Un año después, en Abril del 2015 se presenta Swift 2 como sucesor de Swift(Apple, 2014). Con mejor rendimiento, nuevo manejo de errores y soporte primario a primeras clases. Este lenguaje se lo saca como código abierto para los desarrolladores, algo nuevo en el sistema de desarrollo de Apple.

Digital Crown

Con la presentación del Apple Watch presentan un nuevo botón con el que se podrá desplazar en el reloj, hacer zoom o abrir Siri. Apple lo define como : “Digital Crown. A modern twist on a traditional feature” (Apple, Watch, 2015). El uso del digital Crown busca imitar el del botón análogo tradicional de los relojes pero para acciones propias del reloj.

Apple Watch

Reloj inteligente presentado por Apple en Abril del 2015, el cual viene en 3 versiones: sport, watch y edition. El reloj presenta una pantalla táctil de 38 mm o 42 mm. Viene con una pantalla táctil con un el nuevo sistema force touch, el nuevo botón digital Crown, pulseras cambiables, sensor de pulso de la persona, un vibrador para notificaciones, sensores de movimiento y de luz.

WatchKit

Un API introducido por Apple para programar aplicaciones de iPhone para el Apple Watch. El cual consta de dos partes la extensión del celular que usa recursos del celular y la

aplicación propia que se instala en el reloj (Apple, Apple, 2015). Es necesario agregar el sdk al proyecto además del esquema del Apple watch de esta manera se crean automáticamente los dos tipos en nuestro proyecto una extensión y una app de Apple Watch.

WatchOS2

Con la llegada del iOS9 para el iPhone y el iPad llega el WatchOS2, el nuevo sistema operativo del Apple Watch, que permite a los desarrolladores conjunto con el WatchKit tener mas poder sobre los componentes propios de hardware del dispositivo como son el Digital Crown o los sensores de salud. (Apple, WatchOS2, 2015)

iOS 9

Sistema operativo de los iPhone e iPad que presento Apple en el Keynote de Abril del 2015 pero que llego a las personas en septiembre del mismo año, disponible para todos los dispositivos desde el iPhone 4s trayendo mejoras graficas, de rendimiento, batería y en el desarrollo. Este sistema operativo vino en conjunto con WatchOS2. (Apple, iOS9, 2015)

ViewController para iOS

Son la parte primordial de la estructura interna de las aplicaciones. Cada uno de los ViewControllers maneja partes de la aplicación y es el enlace entre la interface de esta parte de la aplicación y los datos que se usan. Existen dos tipos de view controllers: de contenido y contenedores(controladores hijo y padre). (Apple, View Controllers Programming Guide For iOS, 2015)

Page Content Controller iOS

Tipo de ViewController de tipo contenedor ya que abarca el contenido presente de otros ViewControllers en forma de paginas dentro de este. (Apple, Page View Controllers, 2014)

WKInterfaceController

Es la clase principal que se obtiene del paquete WatchKit para desarrollar la interface de las aplicaciones del Apple Watch. Se usa los interface controllers para manejar los elementos gráficos presentes dentro de la aplicación, algo similar a lo que sucede con los Storyboard en iOS. Los interface controllers corren como extensión dentro del celular y son llamados poco a poco por la aplicación en el Apple Watch. (Apple, WatchKit Framework Reference, 2015)

Storyboard

Es una representación grafica de la interfaz de usuario de la aplicación de iOS que muestra las pantallas de contenido y las conexiones entre las pantallas. (Apple, iOS Developer Library, 2013)

Conexiones IBOutlet y IBAction

Son propiedades de un elemento grafico presente dentro de un Storyboard de un objeto referenciando a otro objeto, un ejemplo de IBOutlet se lo encuentra en Labels, es posible cambiar el texto del Label y un ejemplo de IBAction es el evento de touch de un botón. (Apple, iOS Developer Library, 2012)

AppleStore

Tienda en la cual se venden las aplicaciones creadas para iOS, WatchOS, MacOSX o TVOS. La cual es gratuita pero se necesita un AppleID para poder descargar las aplicaciones.

ItunesConnect

Aplicación de Apple encargada del registro, carga y verificación de las aplicaciones creadas por desarrolladores para los diferentes dispositivos Apple.

TestFlight

Capacidad de ItunesConnect que tiene un desarrollador de enviar aplicaciones a diferentes personas para que prueben la aplicación que subieron a iTunes Connect para que la pongan en AppleStore. Con este medio es posible validar bugs y errores antes de que la aplicación llegue al Apple Store.

DESARROLLO DEL JUEGO PANCHOMANIA PARA WATCHOS Y MANUAL DE USUARIO PARA IOS

AppleWatch

Diseño de la aplicación.

La aplicación consta de 2 clases para el desarrollo de la aplicación propia del celular: PageContentController y ViewController. Para el desarrollo del juego en el reloj inteligente se utilizan 7 clases de Swift2: MenuInicialController, NivelController, RepeticionesController, JuegoColoresController, JuegoTarjetasController y JuegoNumerosController. En la aplicación del celular las dos clases se encuentran conectadas programáticamente en base a page view controller. Por otro lado el juego se encuentra dividido en clases de configuración propia del desarrollo del juego que interactúan indirectamente con las clases de los juegos, las clases propias de los juegos son 3, una clase por cada juego. Se evito el desarrollo de clases que se utilicen en varias partes del juego para lograr frecuencia en el desarrollo del juego, además de que Apple establece que el manejo de cada interface controller se lo debe realizar usando una clase de cada tipo de interface controller para lograr la conexión de los outlets y acciones solo con este interface controller. En el grafico 1 se puede observar el detalle de reciprocidad de uso entre las clases del juego de Apple Watch.

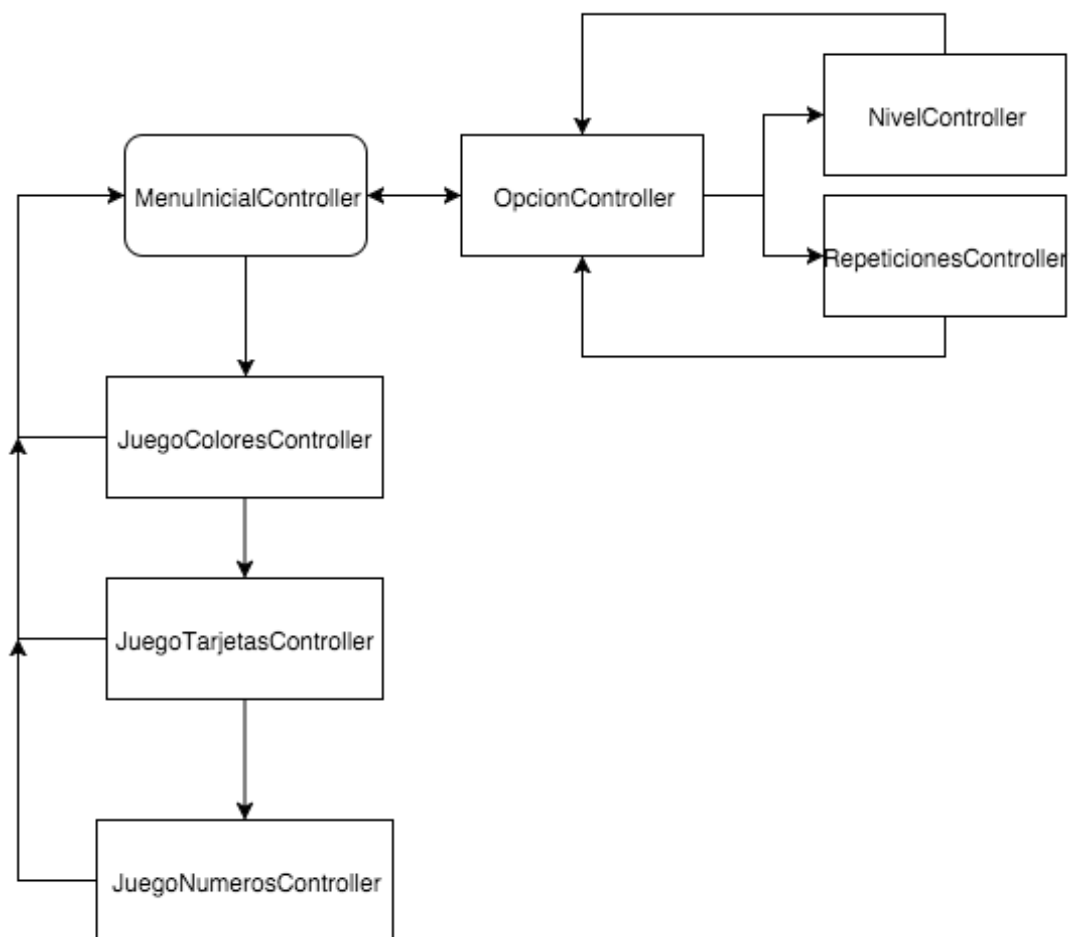


Figura 1 Diseño de uso del juego

MenuInicialController.

Esta clase es la inicial del juego Pancho Manía para el AppleWatch, en esta clase existen outlets de conexión que varían en base al puntaje adquirido durante el juego así como la conexión de eventos de los botones presentes dentro del menú inicial los cuales son empezar el juego y configuración. En el siguiente grafico se puede observar un detalle de lo que se presenta dentro de la clase MenuInicialController.swift

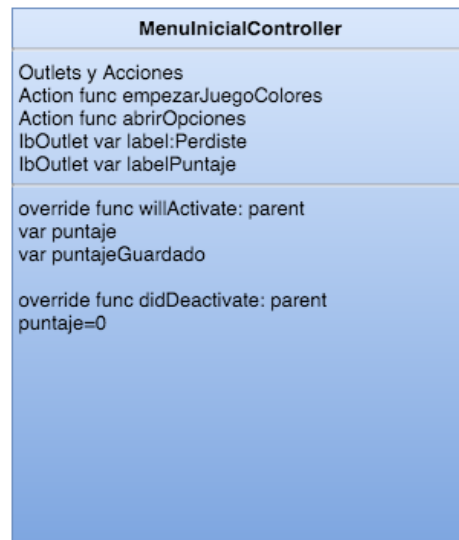


Figura 2 Diseño de la clase **MenuInicialController**

Las acciones de evento abrirán en base de método push los controladores de configuración o el del juego de colores respectivamente.

Los outlets de conexión varían su texto conforme se va jugando, de acuerdo a si se obtiene un puntaje alto o no se presentara con el outlet labelPerdiste el nuevo puntaje alto o el puntaje alto preexistente, con el segundo outlet se presenta solamente el puntaje adquirido si no es un puntaje alto o una notificación si es un puntaje alto.

Dentro de esta clase se realiza la comparación de puntaje alto preexistente con el puntaje adquirido actualmente, ambos puntajes se encuentran presentes en un archivo dentro de la memoria del teléfono, NSUserDefaults de tipo integer los cuales tienen la key: "puntajeJuego" y "puntajeAlto". El primero es el puntaje actual adquirido, el segundo el puntaje alto que se tenía previamente. Todas las acciones se realizan con el método willActivate o didDeactivate. El método willActivate permite realizar acciones cuando el controller se hace visible en la pantalla, es decir cuando el elemento gráfico se presenta en pantalla. Por otro lado didDeactivate realiza acciones cuando el entorno gráfico de este controlador ya no se lo puede ver.

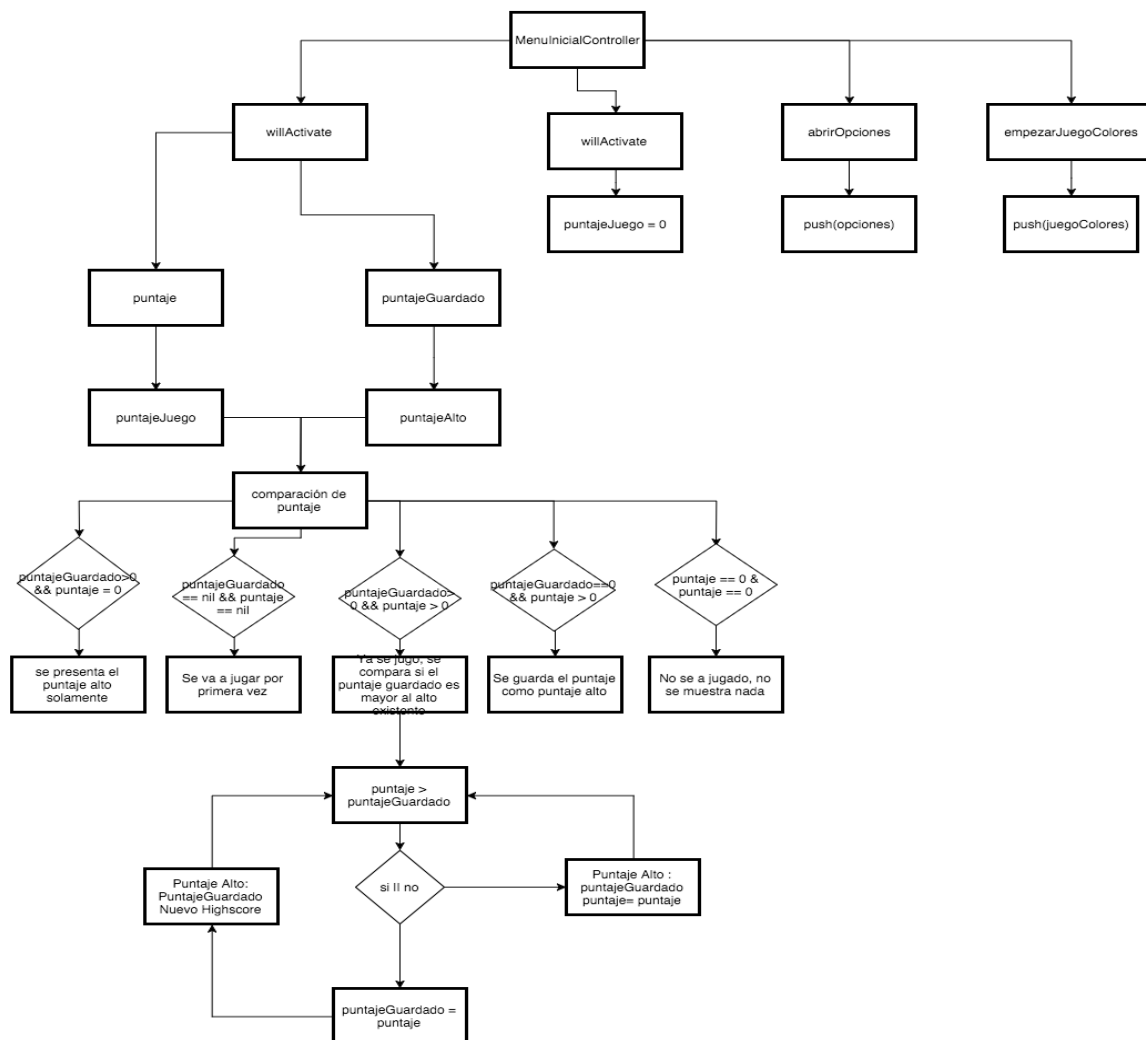


Figura 3 Funcionalidad MenuInicialController

OpcionController.

La clase OpcionController al igual que MenuInicialController se encarga de obtener outlets de conexión y acciones de evento del controlador “opciones”.

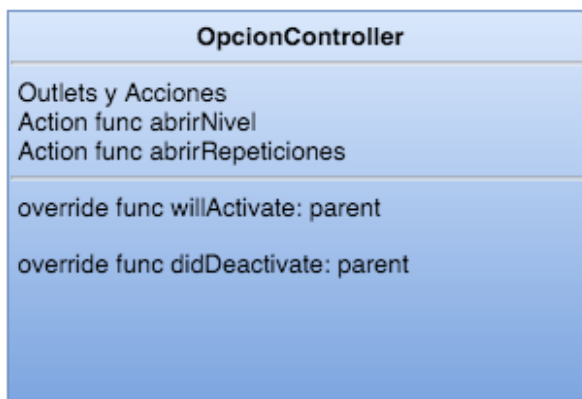


Figura 4 Diseño de la clase OpcionController

Esta clase es el puente de conexión entre el menú inicial y las opciones de configuración de nivel(velocidad) y numero de repeticiones. Estas opciones se abren por método push de controlador. En base de una acción de botón.

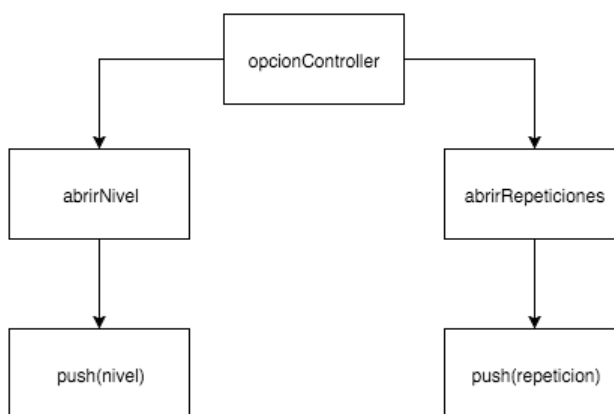


Figura 5 Funcionalidad opcionController

NivelController.

La clase NivelController es la clase encargada de controlar las acciones del interface controller nivel. El cual se encarga de tomar de un slider o del Crown un valor entero que representara el nivel(velocidad). En el grafico 4 se puede ver el detalle del contenido de esta clase:

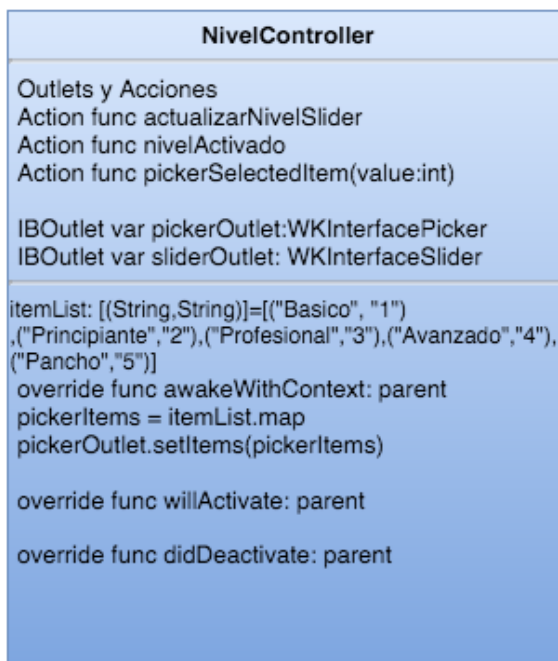


Figura 6 Diseño de la clase NivelController

Dentro de la clase NivelController que es controlador de nivel se encuentra la clase actualizarNivelSlider que es encargada de obtener el valor del slider por acción, y este mismo valor se lo pasa al picker con un valor de -1 ya que los niveles empiezan desde 1 por lo que hay que nivelar estos valores que son diferentes entre el picker y el slider. Lo mismo sucede con el método pickerSelectedItem el cual se encarga de obtener como acción el valor del picker agregado, en base de este entero obtenido como parámetro se cambia el valor del label propio del picker además del valor. El valor de ambos tipos de selección de nivel serán los mismos todo el tiempo y al final con la acción nivel activado se guardara en un UserDefaults el valor del nivel elegido.

RepeticionesController.

Tiene la misma funcionalidad que NivelController, en base a un slider y a un picker que se puede manejar con tacto o con el Digital Crown para cambiar el valor del numero de repeticiones para el primer juego, las repeticiones que se pueden elegir serán 3, 6 o 9. Esto solo afecta el desarrollo del primer juego y sirve para obtener mayor puntaje.

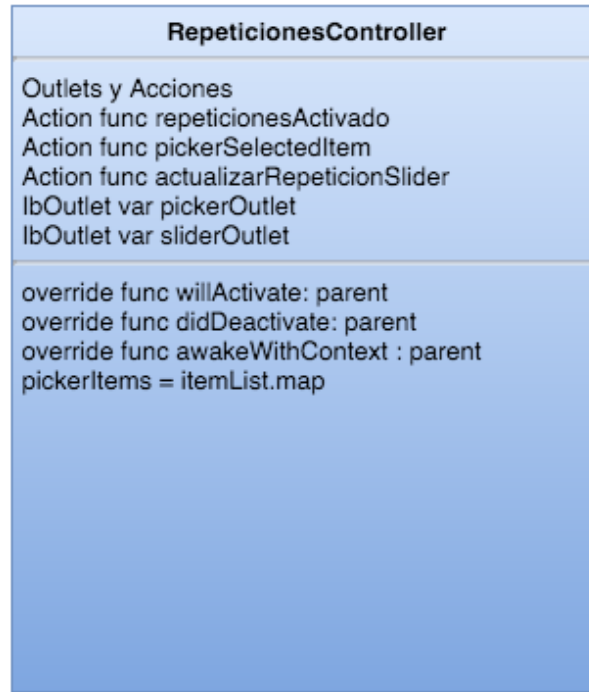


Figura 7 Diseño de la clase RepeticionesController

JuegoColoresController.

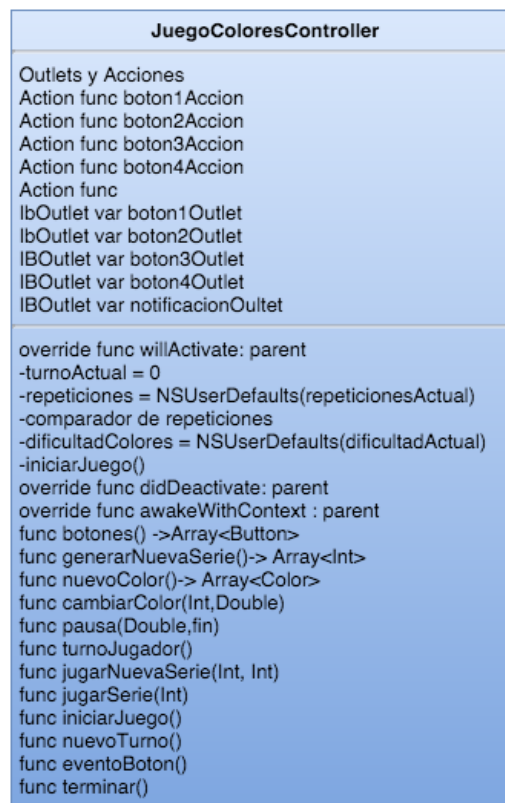
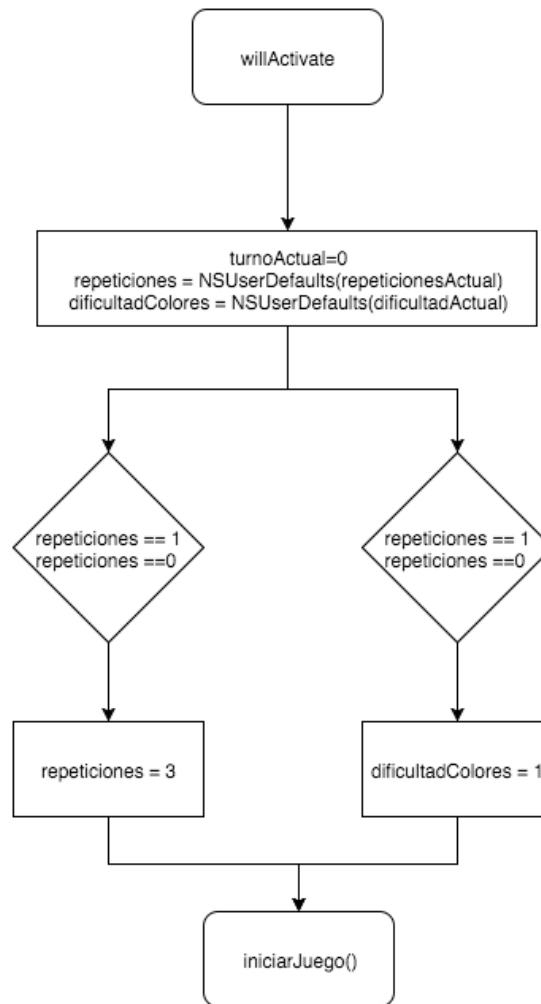


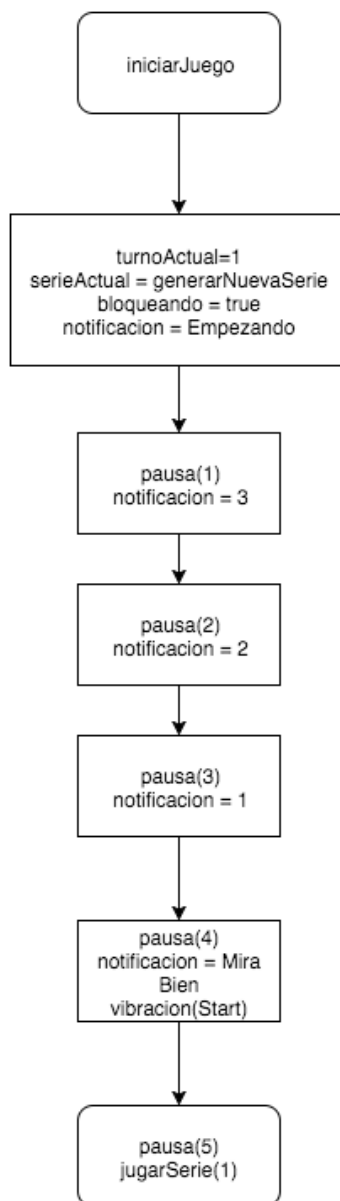
Figura 8 Diseño de la clase JuegoColoresController

Es el controlador del primer juego, este es un juego de colores que hay que seguir la secuencia que se muestra creada aleatoriamente, en esta clase se usa el numero de repeticiones elegidas en la clase RepeticionesController. Si se llega al numero de repeticiones elegida se pasa a la siguiente fase del juego que es el juego de tarjetas. Sino se regresa al menú inicial. Se usa la propiedad propia de notificaciones de vibración del dispositivo para alertar la secuencia de juego generada y para notificar cuando se selecciono uno de los colores. Los 4 colores son 4 botones y en base de outlets y timers se cambia de color del botón por el componente alpha propio del botón.

Existe un label que se usa como mensaje de alerta y de motivación cuando se selecciono correctamente una serie. Asi sea que se perdió y se regresa al menú inicial o que se paso satisfactoriamente las series se guarda en el UserDefaults puntaje el puntaje adquirido. Que posteriormente será el inicio en el juego tarjetas o se usara para comparar como puntaje máximo en el menú inicial.

willActivate.**Figura 9 función willActivate**

La función willActivate que es lo primero que corre cuando se empieza el programa se encarga de inicializar variables e iniciar la función iniciarJuego.

iniciarJuego.**Figura 10 función iniciarJuego**

La función iniciarJuego llama a la función pausa, generarNuevaSerie y jugarSerie. La primera es un delay que se basa en un tiempo de inicio y de fin para poder mostrar en pantalla mensajes como contador de tiempo, o esperar cierto tiempo para cambiar de acción, la segunda función, generarNuevaSerie devuelve un arreglo de 1000 números aleatorios entre 1 y 4. La función jugarSerie por su parte llama a la función jugarNuevaSerie enviando como parámetro el índice de inicio 1.

Series.

En jugarNuevaSerie se inicia con el turno del jugador, llamando a turnoJugador caso contrario se utiliza un algoritmos de control de caso, para variar el tiempo que se demora en cambiar de color a color y el tiempo que se demora un botón en prender y apagar en base a la dificultad escogida, para cambiar de color se llama a la función cambiarColor, posterior a esto se realiza una recursión con jugar nueva serie restando 1 al valor del índice. Para cambiar el color de un botón se utiliza índice del botón así como la duración en base a la dificultad, el color inicial se lo guarda como variable, se cambia en base al color del fondo del botón y llamando a la función nuevoColor, que cambia al color con un componente de alpha de 0.1, es decir mas oscuro, además de esto se realiza una notificación en base a la vibración del reloj del tipo Click. Este mismo criterio se lo utiliza cuando se selecciona un botón, pero usando una notificación del tipo Notification. En el grafico 11 se puede ver con mejor detalle el transcurso del juego de colores.

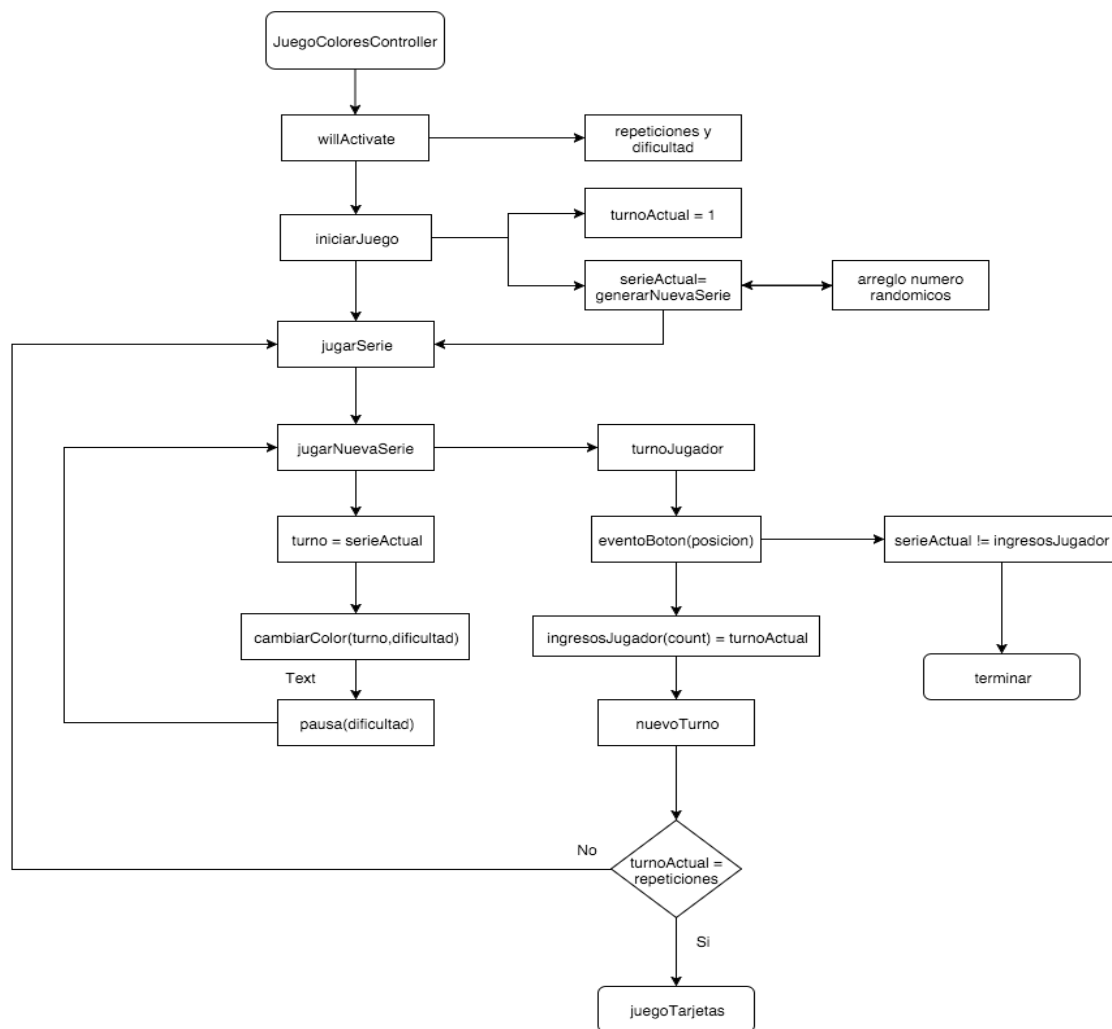


Figura 11 función JuegoColoresController

JuegoTarjetasController.

El juego de tarjetas se basa en la búsqueda de dos tarjetas que tengan la misma imagen, el tiempo que se puede observar las tarjetas varia de acuerdo a la dificultad elegida en un inicio. Si se llega a encontrar los 3 pares de tarjetas se gana y se pasa al siguiente juego.

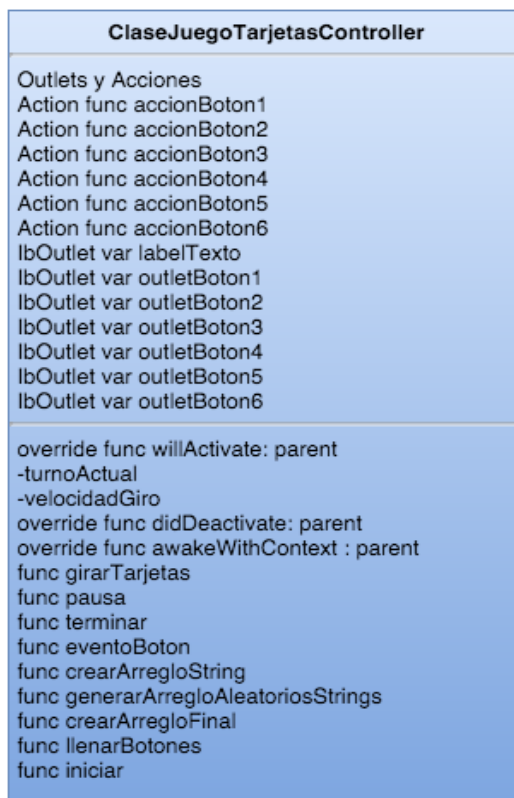


Figura 12 clase JuegoTarjetasController

Al iniciar el juego de tarjetas automáticamente se recibe el puntaje, que es el turno del juego de colores, así como la dificultad que es el tiempo que se demora en darse la vuelta las tarjetas para buscar el par. El juego de tarjetas se basa en creación y comparación de arreglos de datos. Se crea un arreglo de las nombres de las imágenes, posterior a esto se crea un arreglo de números aleatorios de máximo el numero de imágenes que se tiene y posterior a esto un arreglo de 6 números, que va entre 1 a 3 que se va a utilizar para llenar las tarjetas de las imágenes con la función llenar botones.

Con la dificultad se utilizara un tiempo dado para llamar a la función girarTarjetas con la que se vuelve a mostrar la parte trasera de las tarjetas. Desde este punto se espera el evento del botón que mostrara la primera tarjeta, este numero se lo guarda en un nuevo arreglo, se recibe el segundo índice de la segunda tarjeta, si son las mismas se añade 1 punto

al puntaje caso contrario se termina regresando al menú inicial, al tener el contador de ingresos del jugador en 6 se pasara a la siguiente parte del juego, juego de números. En la siguiente figura se puede ver la secuencia de generación de los arreglos y como sirven en el transcurso del juego.

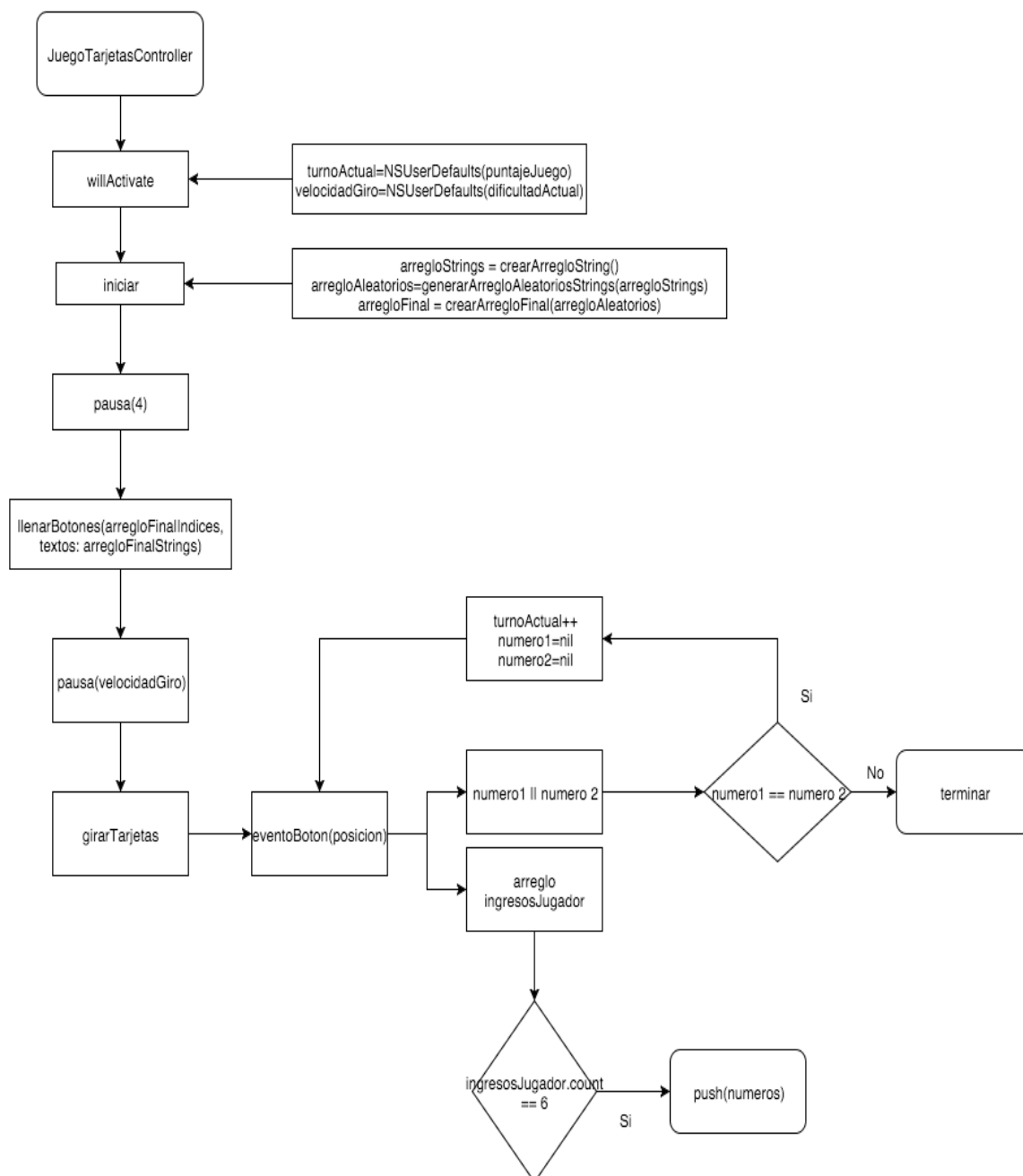


Figura 13 función JuegoTarjetasController

Además de la función pausa que se utiliza en el juego de colores, en el juego de tarjetas se utilizan funciones para la creación de los arreglos aleatorios, para llenar los botones o para el evento de la acción de aplastar la pantalla. Entre las que se encuentran: `crearArregloStrings`, `generarArregloAleatorioStrings`, `crearArregloFinal`, `llenarBotones` y `eventoBoton`.

crearArregloStrings.

Esta función se encarga de crear un arreglo de strings de los nombres de las imágenes que se van a utilizar y retorna el arreglo creado

generarArregloAleatorioStrings.

Recibe como parámetro un arreglo de strings. Se genera 3 números aleatorios hasta el número 13 porque existen 13 imágenes. Al generar un número aleatorio de 1 al 13 se obtiene el texto que se encuentra en la posición dada por el número aleatorio. Este número aleatorio se guarda en un arreglo para evitar que este número se lo vuelva a elegir. Devuelve este arreglo de 3 strings generados.

crearArregloFinal.

Esta función por otro lado recibe un arreglo de strings, con este arreglo de strings de tamaño 3 se lo utiliza como fuente de datos para el arreglo final. Empezando con la fila de arriba, los 3 primeros números, se obtiene 3 números randómicos del 1 al 3, y se valida que no exista ya antes con un arreglo temporal. Después de obtener se realiza lo mismo con los siguientes 3 números, que vendrían a ser la parte de abajo, se obtienen otra vez 3 números randómicos del 1 al 3 sin repetición. Al terminar esto la función devuelve 2 parámetros, el primero es un arreglo de los 6 strings, y el segundo un arreglo de las posiciones que se encuentran los 6 strings.

llenarBotones.

Para llenar los botones se usan los dos arreglos generados por crearArregloFinal para poner cada imagen con el nombre obtenido del primer arreglo de la posición obtenida del segundo arreglo.

eventoBoton.

Esta es la función que se ejecuta al aplastar un botón, en base a la posición de este botón. Con la posición de este botón se obtiene de los arreglos generados en crearArregloFinal, los valores que corresponderían en el índice dado. Esta posición aplastada tendría el índice del arreglo crearArregloFinal. Este numero se lo guarda en numeroUno o numeroDos si ya esta lleno numeroUno. Y se agrega uno a un arreglo temporal que se encarga de contar el numero de ingresos. De esta manera se va comparando numeroUno y numeroDos. Verificando si es el mismo índice, se elimina los valores guardados previamente en numeroUno y en numeroDos, caso contrario se termina la aplicación regresando al menuInicial.

JuegoNumerosController.

ClaseJuegoNumerosController
Outlets y Acciones Action func accionBoton1 Action func accionBoton2 Action func accionBoton3 Action func accionBoton4 Action func accionBoton5 Action func accionBoton6 Action func accionBoton7 Action func accionBoton8 Action func accionBoton9 lbOutlet var outletTexto lbOutlet var outletTimer lbOutlet var outletBoton1 lbOutlet var outletBoton2 lbOutlet var outletBoton3 lbOutlet var outletBoton4 lbOutlet var outletBoton5 lbOutlet var outletBoton6 lbOutlet var outletBoton7 lbOutlet var outletBoton8 lbOutlet var outletBoton9
override func willActivate: parent -turnoActual -velocidadGiro override func didDeactivate: parent override func awakeWithContext : parent func finTimer func iniciar func arreglarArreglo func llenarBotones func crearArregloNumerosAleatorios func terminar func eventoBoton

Figura 14 clase JuegoNumerosController

Este es el tercero y ultimo juego de PanchoMania, en el que hay que seleccionar los números de mayor a menor antes de que se acabe el tiempo del timer, este tiempo varia de acuerdo a la dificultad elegida en el menú inicial.

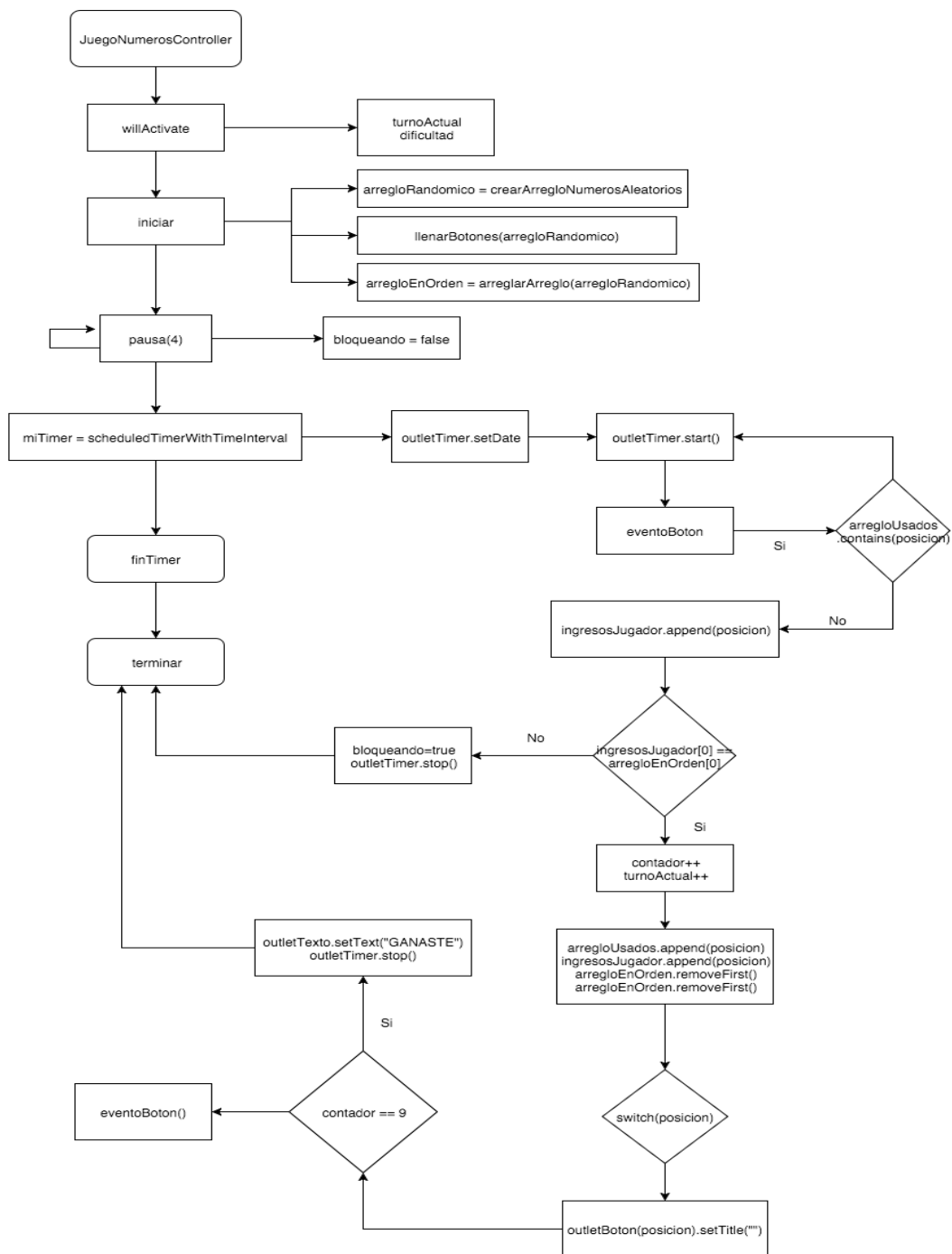


Figura 15 Trayectoria juego de números

Iniciar.

En la función iniciar() se crean los arreglos que van a servir para todo el juego, primero se crea un arreglo randómico que sirve para llenar los 9 cuadrados de botones. Posterior a esto se ordena al arreglo en orden descendente con la función arreglarArreglo(). Después se crea e inicializa el timer con el outlet del atributo timer generado previamente.

crearArregloNumerosAleatorios.

En esta función se crea un arreglo de 9 números aleatorios entre 1 y el 100, validando valores únicos. Y se retorna un arreglo de tipo entero.

llenarBotones.

Con esta función se llenan los 9 botones en base de los outlets generados en el inicio. Se recibe el arreglo de números aleatorios. Y se los pone en el orden que llegan empezando por el botón superior izquierdo.

arreglarArreglo.

Se recibe el arreglo de números aleatorios generado con la función crearArregloNumerosAleatorios. Este arreglo se lo arregla con la función propia de Swift sort(>), el signo ">" especifica que va de mayor a menor.

eventoBoton.

Al igual que en los demás controladores, el evento del botón es el eje principal en el transcurso del juego, la posición de cada botón se la recibe como parámetro. Si es que el booleano bloqueando se encuentra en falso se puede usar el touch sino todo lo que se tope en pantalla no sirve. Si hay como ingresar por teclado se recibe la posición que se ingreso. Esta posición se compara si ya se la ingreso previamente si no se la ingresado se compara con el primer elemento del arreglo ordenado. Si los dos valores son iguales se elimina el valor del arreglo ordenado y de los ingresos del jugador que es el valor usado inicialmente

para la comparación. Se suma el valor de 1 al turno y al contador de ingresos. Si el valor de ingresos llega a ser 9 se gana y se regresa al menú inicial. Caso contrario se sigue jugando mientras el timer no llega a cero. Si el timer llega a cero se pierde y se llama a la función terminar. En la función terminar se guarda en el archivo UserDefaults el valor del turnoActual en puntajeJuego.

APLICACIÓN IOS PARA IPHONE

Cuando se compra o descarga una aplicación para el Apple Watch en el AppStore automáticamente se descarga también una aplicación para el iPhone enlazado con el reloj. No hay que olvidar que cuando se desarrolla una aplicación para el reloj inteligente es necesario crear dos partes de la aplicación, la app en si y una extensión. La extensión toma los datos desde el celular, sin importar que el código programado este hecho específicamente para el reloj. La app propia del reloj tiene los elementos gráficos. La extensión las clases necesarias para su ejecución. Es por esto que se vio necesario el crear una aplicación también para el celular, de otra manera el celular hubiera tenido un icono de una aplicación que no sirve de nada.

Estructura de la aplicación de iOS.

La aplicación mostrara un pequeño manual de usuario de la aplicación del reloj con tomas de pantalla y definiciones por cada pantalla que se puede ver del juego. Consta de 3 controladores gráficos y 2 clases para su funcionamiento. Es un formato de MVC en el que el modelo es ViewController, el controlador es PageContentViewController y la vista es PageViewController.

Esquema de uso.

La aplicación funcionara principalmente con un contenedor de controladores de tipo PageViewController. Este contenedor contiene los outlets de conexión con los elementos gráficos los cuales son una imagen y un label. Al label se le agrego una animación para cuando se cambia de pagina. Por otro lado existe un controlador que tiene en su clase todos los recursos que va a consumir el contenedor. En la clase ViewController existen todos los textos de los labels, nombres de imágenes, etc. Este controlador extiende de la clase UIPageViewController para su funcionamiento como pagina. En el ViewController se mantiene una indexación de las paginas que se van a mostrar, el limite es el tamaño del arreglo de textos.

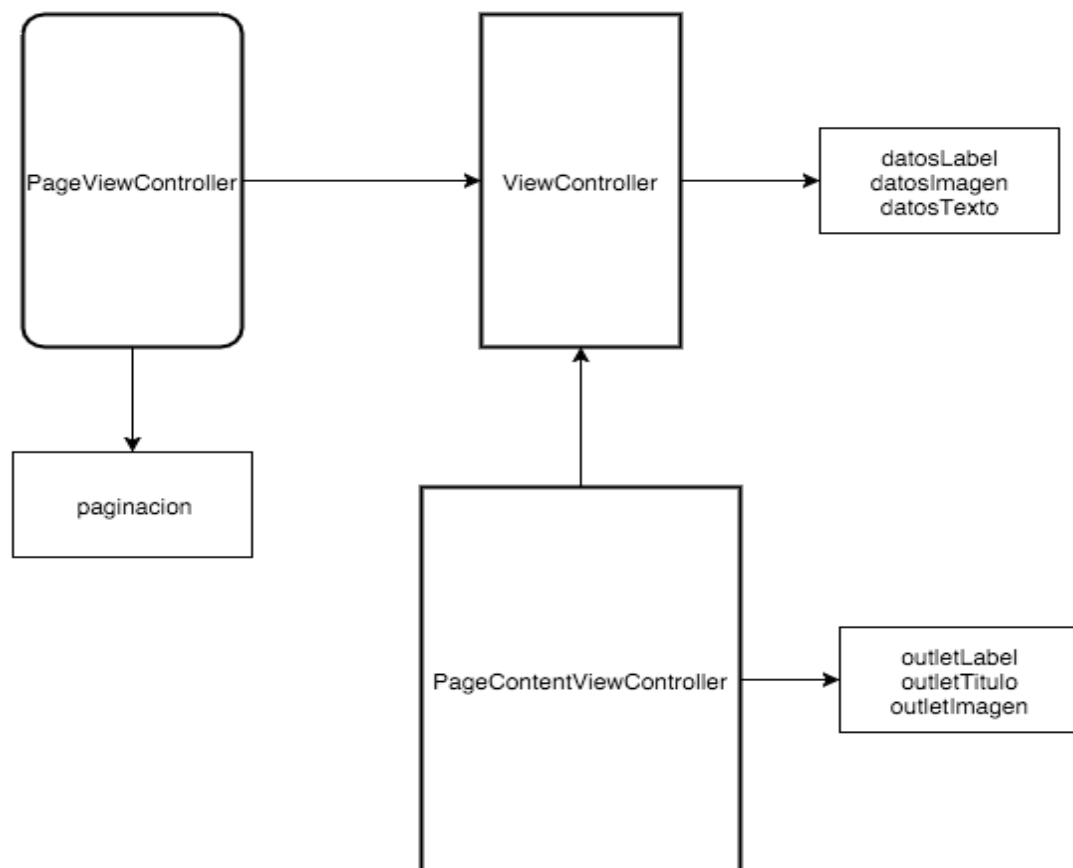


Figura 16 Funcionalidad Aplicación iOS

PageViewController.

No tiene una clase controlador propio ya que esta es controlador madre de los view controllers, esta da la forma de paginación dentro de la aplicación.

PageContentViewController.

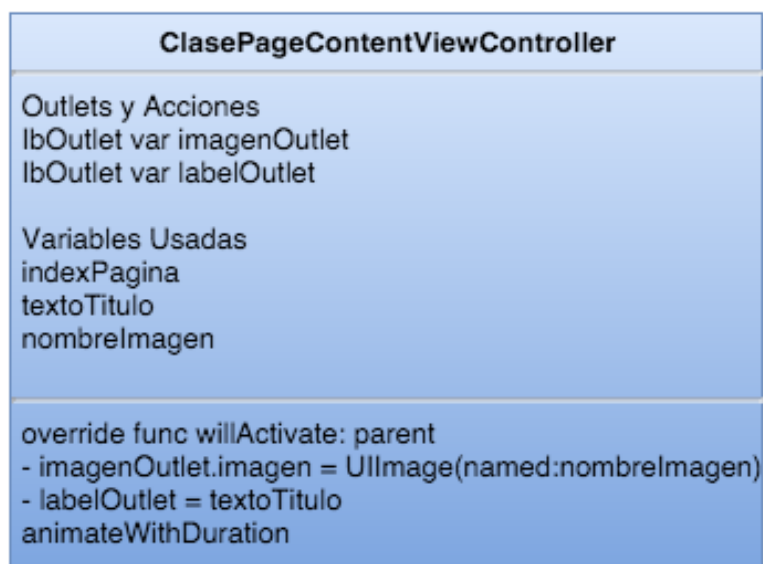


Figura 17 Clase PageContentViewController

Para no tener varios ViewControllers con formato similar, se creo un contenedor, el cual tiene la estructura básica que tendrá el View y los outlets de conexión, este obtendrá los datos de ViewController. Por lo tanto esta clase solo tienen los outlets de conexión, y los setters de los atributos del texto y de la imagen que se presentaran dentro del ViewController.

ViewController.

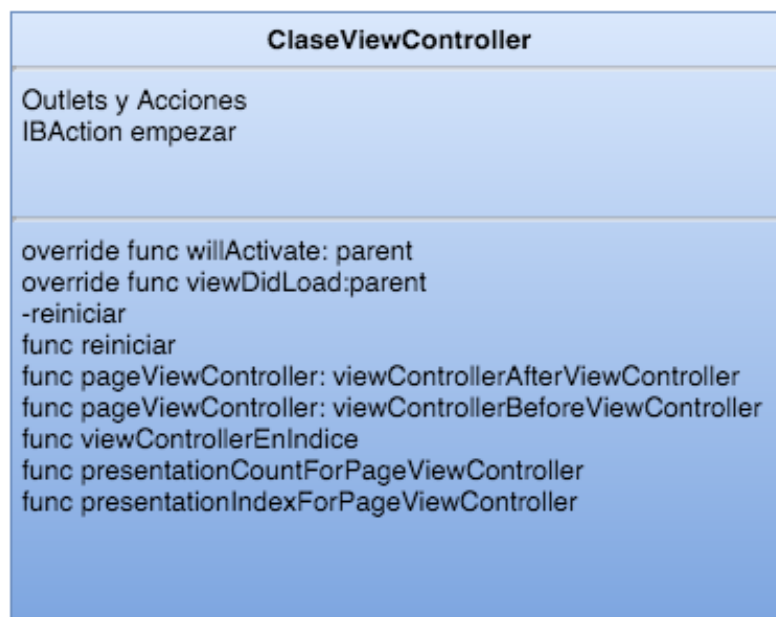


Figura 18 Clase PageContentViewController

La clase ViewController en el formato MVC vendría a ser el modelo, ya que tiene la estructura de los datos que se van a enviar al controlador para que los procese la vista. Dentro de esta clase se establecen las imágenes que se van a mostrar en pantalla, así como los textos de cada imagen. Existen funciones para poder indexar y manejar correctamente las vistas. La función viewControllerEnIndice se encarga de establecer que texto y que imagen van en cada página a presentarse que se la establece con un índice. La función que usa el método viewControllerAfterViewController se encarga de manejar el paso a la siguiente página sumando el índice. La clase viewControllerBeforeViewController se encarga de regresar a la anterior página y restar el índice. Ambas son funciones propias del tipo pageViewController que es el manejador de nuestra vista. Las funciones presentationCountForPageViewController y presentationIndexForPageViewController también son métodos propios de Swift, se encargan de manipular el contador en los círculos

que se presentan como medidor en cada pagina. La función empezar es un evento en el botón “Empezar de nuevo”. La cual se encarga de indexar en la pagina cero por lo que solo se podrá realizar un movimiento “AfterViewControlller”.

PUBLICACIÓN EN EL APPSTORE


Cuenta de desarrollador

Después de crear la aplicación exitosamente, el siguiente paso es la carga de la aplicación al AppStore de Apple en donde se podrá descargar la aplicación en cualquier dispositivo. El primer paso es la suscripción como desarrollador de Apple. Cualquier persona puede hacerlo. Existen dos tipos de suscripciones, individual en la que es una sola persona la que tiene el derecho o empresarial, en la que se tienen diferentes ventajas como la instalación de la aplicación sin necesidad de tener el AppStore para descargarse, se lo puede hacer por medio de una descarga desde un link en un servidor (OTA). La licencia de desarrollador tiene un costo de \$99 dólares anuales.


Items

[Back to Apple Developer](#)

We will email you when your products are activated.

	Apple Developer Program – Membership for one year Membership Product	\$99.00
---	--	----------------

Payment

Billing Contact	Francisco Foyain Lara	Payment Method	
Billing Address			

Bag Subtotal	\$99.00
Order Total	\$99.00

Please note that your order is governed by Apple's [Sales and Refund Policy](#).

Figura 19 Licencia de desarrollo

Creación de perfiles, certificados de desarrollo e identificador

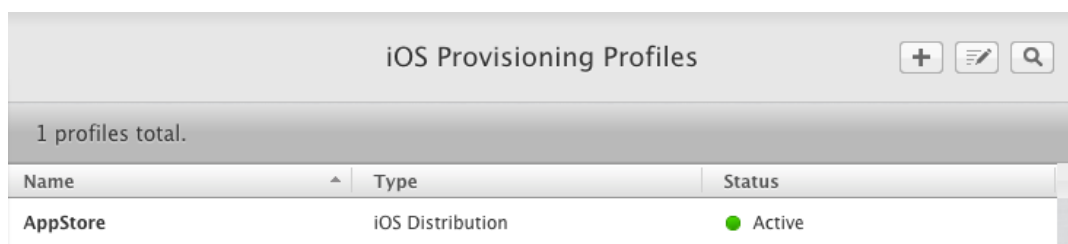
Los perfiles de desarrollo son indispensables para la prueba de la aplicación en un dispositivo propio, además de que hay que instalarlo en la maquina para verificar la cuenta de desarrollador. Por otro lado un certificado de desarrollo es la firma electrónica de la aplicación que se lo hace con la computadora con la que se compila la aplicación para la carga. Es decir es un certificado de autenticidad de la compilación de la aplicación del desarrollador. Este certificado sirve mientras se esta desarrollando la aplicación, para cargar la aplicación al AppStore es necesario crear un certificado de distribución. Que tiene el mismo fin que el certificado de desarrollo. El identificador de la aplicación será el id de la aplicación.



The screenshot shows the 'iOS Certificates' window with a search bar and a plus icon. Below the title bar, it indicates '2 Certificates Total'. A table lists the certificates:

Name	Type	Expires
Francisco Foyain Lara	iOS Development	Nov 03, 2016
Francisco Foyain Lara	iOS Distribution	Nov 22, 2016

Figura 20 Certificados



The screenshot shows the 'iOS Provisioning Profiles' window with a search bar, a plus icon, and an edit icon. Below the title bar, it indicates '1 profiles total.'. A table lists the profiles:

Name	Type	Status
AppStore	iOS Distribution	● Active

Figura 21 Perfil de distribución

iOS App IDs	
2 App IDs Total	
Name	ID
Xcode iOS Wildcard App ID	*
com-ffoyain-PanchoMania	com.foyain.PanchoMania

Figura 22 Identificadores

Itunes Connect

Cuando se tienen listos los certificados, perfiles e identificadores el siguiente paso es crear una entrada de aplicación dentro de iTunes Connect para que Apple puede verificar la aplicación y aceptarla o rechazarla.

The screenshot shows the iTunes Connect interface for the app 'PanchoMania'. The page is titled 'Información de la app' and includes a 'Guardar' button. The main content area is divided into sections: 'Información que se puede traducir' and 'Información general'. The 'Información que se puede traducir' section contains a 'Nombre' field with the value 'PanchoMania' and a 'URL de la política de privacidad' field with the value 'http://example.com (opcional)'. The 'Información general' section contains an 'ID de pack' dropdown menu with the value 'com-ffoyain-PanchoMania - com.foyain.PanchoMania', a 'SKU' field with the value 'com.foyain.PanchoMania', and an 'ID de Apple' field with the value '1061845557'. The 'Idioma principal' is set to 'Español (España)'. The 'Categoría' section has three dropdown menus: 'Juegos', 'Puzzle', and 'Cartas', with a 'Secundario (opcional)' dropdown menu below them.

Figura 23 Registro en Itunes Connect

En iTunes Connect hay que llenar todos los formularios que se pide como son información de la aplicación así como precio y disponibilidad. Si se requiere agregar

prestaciones extras como GameCenter o respaldo en iCloud en la pestaña prestaciones es posible agregarlo y editarlo. Si la aplicación tuviera costo es necesario llenar un contrato con Apple para verificar el porcentaje de ganancia de Apple y el del desarrollador. La aplicación PanchoMania no tiene costo alguno. Es necesario cargar los iconos de la aplicación así como las capturas de pantalla o video que se presentara en el AppStore cuando se busca la aplicación. En la figura 23 se puede ver un ejemplo de esto.

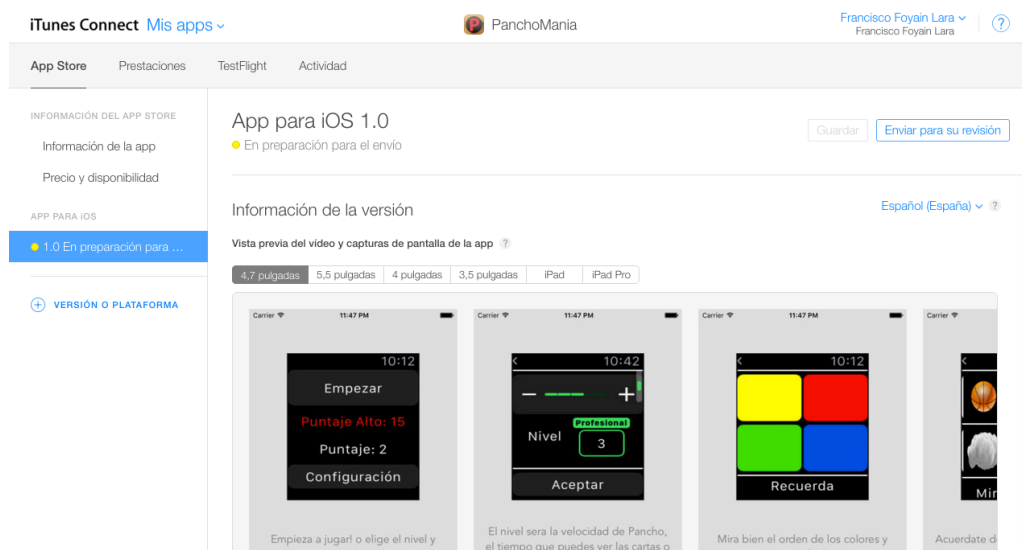


Figura 24 Registro en Itunes Connect

Archivo y carga desde Xcode

Al tener listo el registro en iTunes Connect se puede archivar la aplicación y cargarla. Para esto es necesario cambiar el certificado de desarrollador al de distribución. En XCode se archiva la aplicación poniendo como esquema un dispositivo genérico. Posterior a esto desde XCode abrirá la aplicación Organizer. En donde se podrá ver el programa Archivo y poder subirlo al AppStore como se puede ver en el grafico 24.

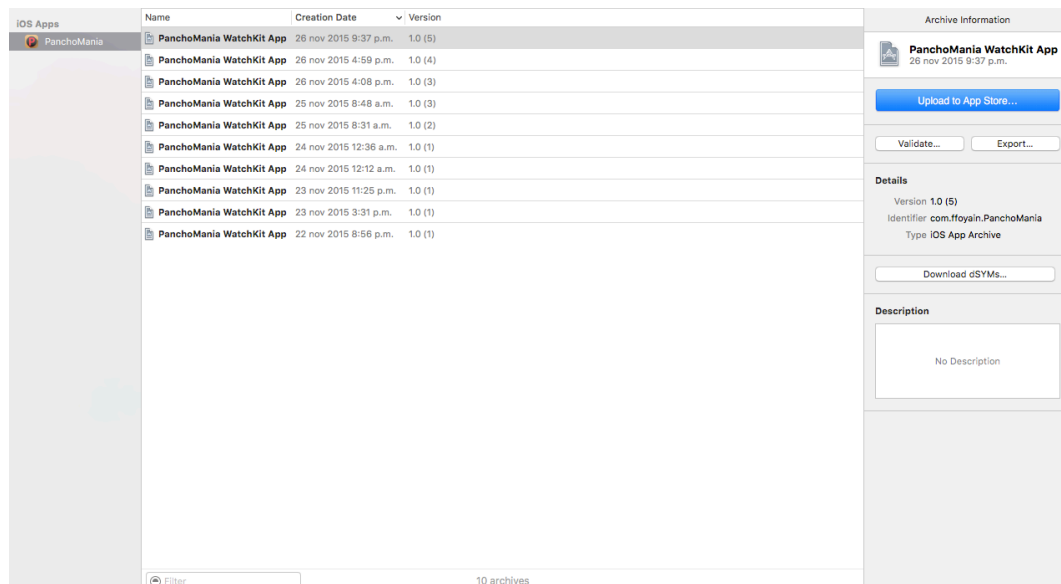


Figura 25 Organizer de Aplicación

Cuando se selecciona en “Upload to AppStore” se conectara con el servidor de Apple en donde se procederá a verificar ciertos atributos básicos de la aplicación. Como se puede ver en los siguientes gráficos:

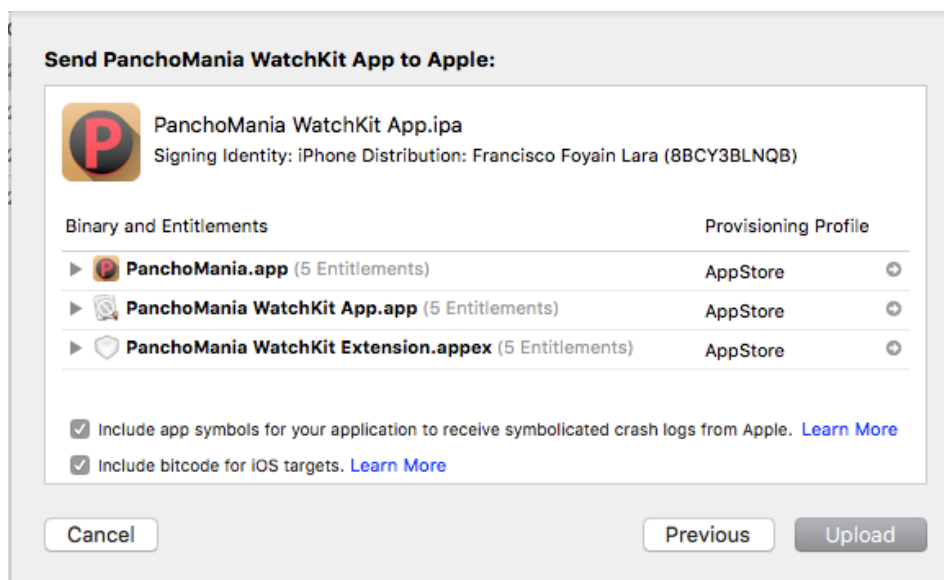


Figura 26 Carga de Aplicación

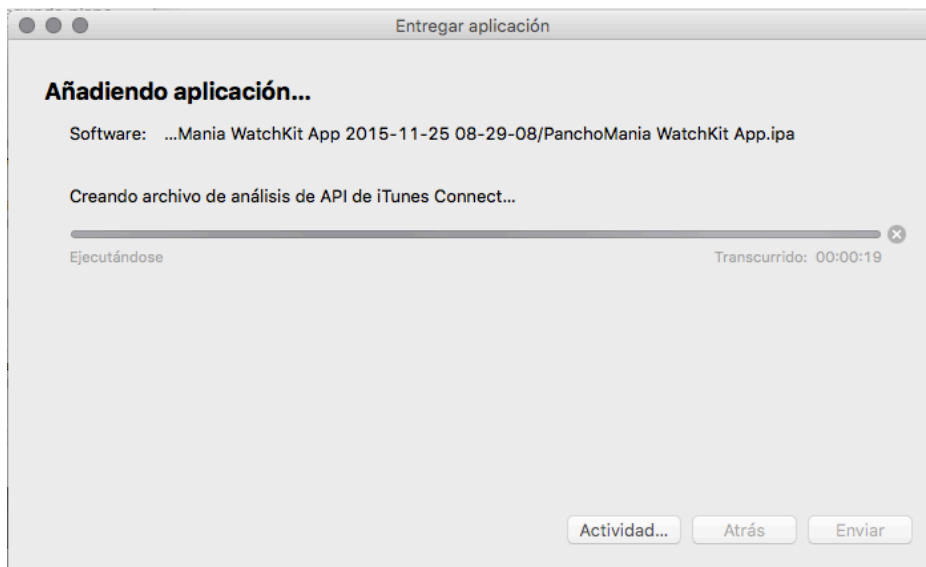


Figura 27 Carga de Aplicación 2

Error de carga de compilación

Con la carga con el Xcode, Apple realiza una verificación preliminar de la aplicación de lo que es imágenes, tamaño de archivos, capturas de pantalla, el splashscreen, entre otras cosas. Al intentar cargar la aplicación a iTunes Connect la aplicación se subía exitosamente pero en iTunes Connect se quedaba en un estado de carga por lo que no se podía pasar a la siguiente fase de verificación. Apple envió un email con el error del archivo, las imágenes que se estaban cargando no podían estar en con Alpha Channel. Así que con la aplicación freeware alpha channel remover se corrigió este error de todas las imágenes para el intento de carga número 5.

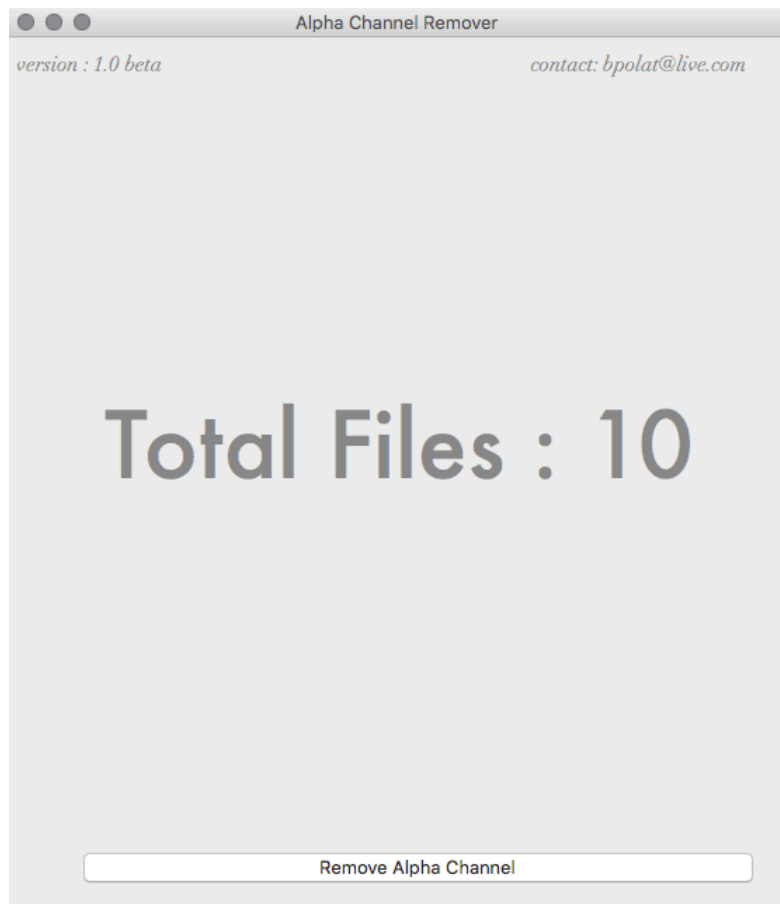


Figura 28 Alpha Channel Remover

Carga exitosa a iTunes Connect

El día 26 de Noviembre del 2015 se pudo cargar exitosamente el archivo de la aplicación al AppStore para que Apple verifique el juego PanchoMania. Demoro 3 días en la corrección de errores y verificación de cambios para que la versión 5 se procese finalmente en iTunes Connect.

Compilaciones de iOS

Todas las compilaciones que se han enviado para iOS. Los números de la versión son los números de la versión Xcode.

✓ Versión 1.0










Compilación	Fecha de carga	Estado de las pruebas externas	Estado en el App Store
 5 (En proceso)	El 26 de nov. de 2015 a las 21:46		
 4 	El 26 de nov. de 2015 a las 17:04		
 3 	El 26 de nov. de 2015 a las 16:15		
 2 	El 24 de nov. de 2015 a las 0:46		
 1 	El 23 de nov. de 2015 a las 23:35		

Figura 29 Compilaciones de iOS

Aprobación de la aplicación

El día 5 de diciembre de 2015 se recibió un email de aprobación de la aplicación por parte de Apple, tomo en total 9 días para que Apple apruebe la aplicación y empiece la distribución en los diferentes mercados de AppStore.

Dear Francisco Foyain Lara,

The following app has been approved and the app status has changed to Ready for Sale:

App Name: PanchoMania
 App Version Number: 1.0
 App Type: iOS
 App SKU: com.ffoyain.PanchoMania
 App Apple ID:1061845557

If your contracts are not in effect at this time, your app status will be Pending Contract. You may track the progress of your contracts in the [Agreements, Tax, and Banking](#) module in iTunes Connect.

To make changes to this app, sign in to iTunes Connect and open the [Manage Your Applications](#) module.

It can take up to 24 hours before your app is available on the App Store. This delay is dependent on any app availability issues.

Before you market your app, read the [App Store Marketing and Advertising Guidelines](#) for Developers. The guidelines include information on using the App Store badge, best practices to market apps on the App Store, and details on the use of Apple product images.

If you have any questions regarding your app, use the [Contact Us](#) module on iTunes Connect.

Regards,
 The App Store team

Figura 30 Aprobación de PanchoMania.

RESULTADOS OBTENIDOS

Encuesta a niños

La aplicación PanchoMania tiene como principal objetivo que los niños la puedan jugar por lo que se hizo una encuesta niños desde los 5 años hasta los 13 años para ver su nivel de satisfacción. En general el único problema que tenían los niños de esta edad fue la desconcentración con el reloj en si, querían jugar con el reloj hasta que se den cuenta que hay un juego en la pantalla. El juego de colores y el de las tarjetas fue jugado exitosamente por todos los niños, el juego de números los mas pequeños tuvieron dificultad.

Encuesta a los adultos mayores

El segundo objetivo del juego PanchoMania son los adultos mayores. Por lo que se hizo que 4 adultos mayores interactúen con el juego para ver sus reacciones y facilidad de uso. El único problema que se encontró fue el de falta de manejo de dispositivos electrónicos de ellos, por lo que después de una breve explicación del manejo del reloj todos jugaron exitosamente el juego, sin ninguna dificultad del tamaño de los gráficos o pantalla.

Estadísticas de descarga de la aplicación.

Estas estadísticas se las tomo el día 10 de diciembre de 2015, 5 días después de que la aplicación PanchoMania se encuentre en el AppStore se lo realizo con iTunes Connect que tiene un plug-in propio para mostrar estos resultados. Los resultados van desde el día 5 de diciembre al 9 de diciembre de 2015. En los 4 días se obtiene 116 visitas en el AppStore y un total de 11 descargas de la aplicación, teniendo al Ecuador como mayor territorio de descarga con 7 descargas. Tomando en cuenta que no todos tienen un reloj AppleWatch el

crecimiento de descargas del juego tiene una tendencia de crecimiento, como se puede ver en las siguientes graficas:

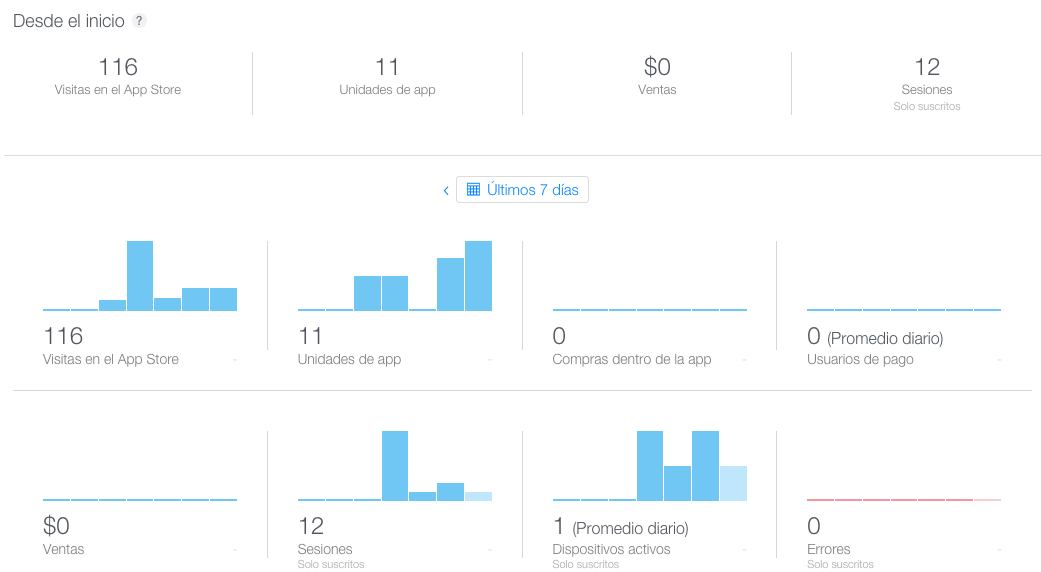


Figura 31 Detalle de estadísticas.

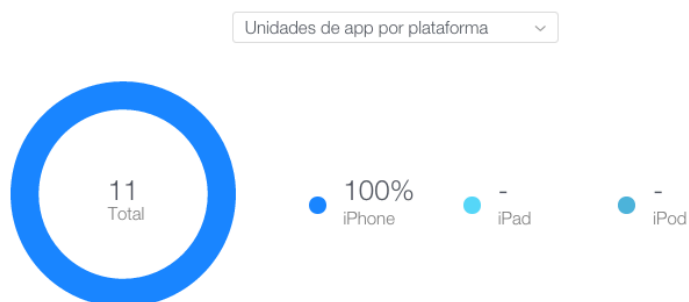


Figura 32 Dispositivos que han descargado.

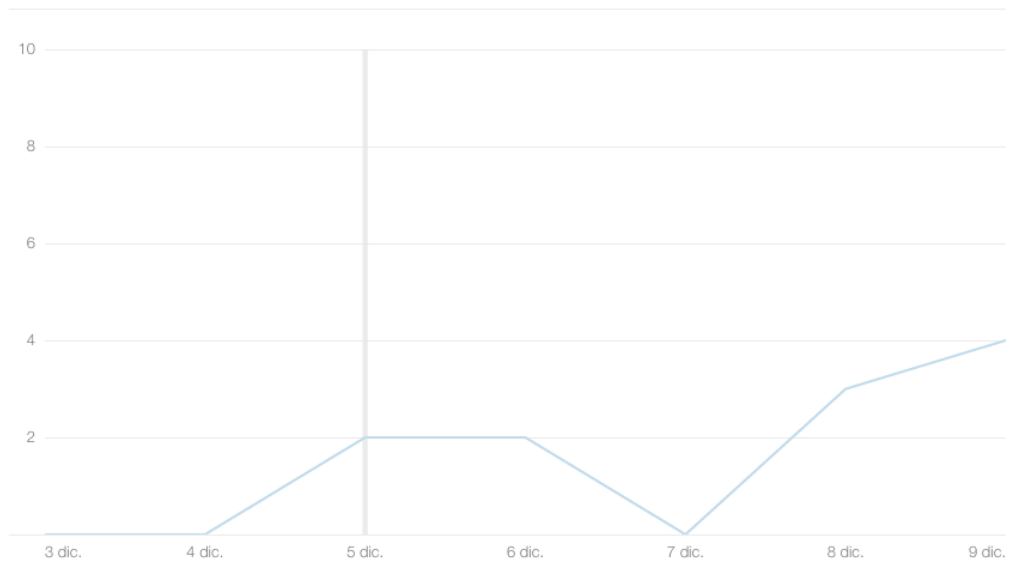


Figura 33 Instalación por día.

CONCLUSIONES

El desarrollo del juego PanchoMania para el reloj inteligente Apple Watch se lo logro eficazmente, logrando concretar todas las fases del juego con las prestaciones esperadas. No fue necesario utilizar bases de datos porque no se utilizaban gran cantidad de datos, el uso de los UserDefaults fue primordial para guardar el puntaje alto alcanzado así como el nivel y numero de repeticiones para el transcurso del juego. El uso de Swift 2 fue un limitante ya que existía falta de recursos de documentación para la programación, en este caso lo que se pudo utilizar fue la documentación propia de Apple sobre los métodos y controladores usados. Se logro crear 7 interface controllers para el manejo de la aplicación del reloj, empezando por el menú, opciones, nivel, repeticiones, el juego de colores, el juego de tarjetas y el juego de números. En el caso de los niveles y repeticiones se logro implementar recursos de hardware propios como el Digital Crown del reloj para elegir el nivel de dificultad o el numero de repeticiones. En los juegos en si se utilizo los UserDefaults para el manejo del turno actual como el del nivel y repeticiones elegido. Se agrego notificaciones de vibración durante el juego para que sea fácil de sentir y presenciar el hecho de un acto como el touch de pantalla. Esto favoreció a que los adultos mayores sepan que han aplastado ya la pantalla. Las imágenes utilizadas en el juego son sin derecho de autor. La carga de la aplicación al AppStore demoro en total 12 días desde el primer intento hasta que se encontró aprobada. Se crearon los perfiles, identificadores y perfiles exitosamente. El único problema que se encontró fue una falta de indicaciones sobre lo que hay como usar y lo que no hay como usar. Dando como resultado el error en el formato de archivo de las imágenes utilizadas. Después de editarlas y cargar el archivo de la aplicación el 26 de noviembre de 2015, Apple aprobó el juego el 5 de diciembre de 2015.

Llegando en 4 días a alcanzar 11 descargas y 116 visualizaciones. El juego se lo puede encontrar en el AppStore como PanchoMania sin costo en 12 regiones del mundo. Se espera agregar ciertas funciones como GameCenter y la creación del juego para el teléfono inteligente iPhone para la próxima versión del juego. Recomiendo tener un margen de 10 días para tener la aplicación cargada en el AppStore.

REFERENCIAS BIBLIOGRAFICAS

- Medina, F. (12 de Junio de 2014). *Introduccion a Swift-Hola Programadores #SwiftEspañol*. Recuperado el 05 de Diciembre de 2015, de Programadores-iOS: <http://programadores-ios.net/introduccion-swift-un-primer-vistazo-hola-programadores/>
- Apple. (8 de Junio de 2014). *Swift2*. Recuperado el 5 de Diciembre de 2015, de Developer Apple: <https://developer.apple.com/swift/blog/?id=29>
- Apple. (2015). *Watch*. Recuperado el 5 de Diciembre de 2015, de Apple: <http://www.apple.com/watch/watch-reimagined/>
- Apple. (2015). *Apple*. Recuperado el 5 de Diciembre de 2015, de Developer WatchKit: <https://developer.apple.com/watchkit/>
- Apple. (2015). *WatchOS2*. Recuperado el 5 de Diciembre de 2015, de WatchOS2: <https://developer.apple.com/watchos/>
- Apple. (s.f.). *iOS9*. Obtenido de iOS9: <http://www.apple.com/ios/whats-new/>
- Apple. (2015). *iOS9*. Recuperado el 2015 de Diciembre de 2015, de iOS9: <http://www.apple.com/ios/whats-new/>
- Apple. (21 de Octubre de 2015). *View Controllers Programming Guide For iOS*. Recuperado el 6 de Diciembre de 2015, de iOS Developer library: <https://developer.apple.com/library/prerelease/ios/featuredarticles/ViewControllerPGforiPhoneOS/>
- Apple. (12 de Noviembre de 2014). *Page View Controllers*. Recuperado el 6 de Diciembre de 2015, de View Controller Catalog for iOS: <https://developer.apple.com/library/ios/documentation/WindowsViews/Conceptual/ViewControllerCatalog/Chapters/PageViewControllers.html>
- Apple. (21 de Marzo de 2015). *WatchKit Framework Reference*. Recuperado el 6 de Diciembre de 2015, de iOS Developer Library: https://developer.apple.com/library/ios/documentation/WatchKit/Reference/WKInterfaceController_class/
- Apple. (13 de Septiembre de 2013). *iOS Developer Library*. Recuperado el 6 de Diciembre de 2015, de Cocoa Application Competencies for iOS: <https://developer.apple.com/library/ios/documentation/General/Conceptual/Devpedia-CocoaApp/Storyboard.html>
- Apple. (9 de Enero de 2012). *iOS Developer Library*. Recuperado el 6 de Dic de 2015, de Concepts in Objective-C Programming:

<https://developer.apple.com/library/ios/documentation/General/Conceptual/CocoaEncyclopedia/Outlets/Outlets.html>

ANEXO A: MENUINICIALCONTROLLER

```

//
// MenuInicialController.swift
// PanchoMania
//
// Created by Francisco Foyain Lara on 7/10/15.
// Copyright © 2015 FranciscoFoyain. All rights reserved.
//

import WatchKit
import Foundation

class MenuInicialController: WKInterfaceController {

    // outlet de accion del boton empezar

    @IBAction func empezarJuegoColores() {
        self.pushControllerWithName("juegoColores", context: self);
    }

    //outlet de accion del boton opciones

    @IBAction func abrirOpciones() {
        self.pushControllerWithName("opciones", context: self);
    }

    //outlets de conexion con los labels.

    @IBOutlet var labelPerdiste: WKInterfaceLabel!

    @IBOutlet var labelPuntaje: WKInterfaceLabel!

    override func awakeWithContext(context: AnyObject?) {
        super.awakeWithContext(context)

    }

    override func willActivate() {
        // This method is called when watch view controller is about
        to be visible to user
        super.willActivate()
        // se recibe los valores de los keys puntajeJuego y
        puntajeAlto
        var puntaje =
NSUserDefaults.standardUserDefaults().integerForKey("puntajeJuego")
as Int!
        var puntajeGuardado =

```

```

NSUserDefaults.standardUserDefaults().integerForKey("puntajeAlto")
as Int!

    //NSLog("Puntaje \ \(puntaje)")
    //NSLog("ALTO \ \(puntajeGuardado)")

    // Se valida que exista alto previamente
    if (puntajeGuardado > 0 && puntaje == 0){
        labelPerdiste.setText("Puntaje Alto: \ \(puntajeGuardado)"
)
        labelPerdiste.setHidden(false)
        labelPuntaje.setHidden(true)
    }
    // Se valida si no existe ni puntaje alto ni actual para no
    presentar ningun
    if (puntajeGuardado == nil && puntaje == nil){
        puntajeGuardado = 0
        puntaje = 0
        labelPerdiste.setHidden(true)
        labelPuntaje.setHidden(true)
    }
    //Se ocultan los labels si no existe puntaje alto o puntaje
    actual

    if(puntajeGuardado == 0 && puntaje == 0){
        labelPerdiste.setHidden(true)
        labelPuntaje.setHidden(true)
    }

    //Si existe puntaje alto o actual se validara si existe un
    nuevo puntaje y se muestran los labels
    if(puntajeGuardado > 0 && puntaje > 0){
        // se verifica si el puntaje actual es mayor al puntaje
        alto

        if(puntaje > puntajeGuardado){
            puntajeGuardado = puntaje
        }

        NSUserDefaults.standardUserDefaults().setInteger(puntajeGuardado,
        forKey: "puntajeAlto")
        labelPerdiste.setText("Puntaje Alto:
        \ \(puntajeGuardado)" )
        labelPuntaje.setText("Nuevo puntaje alto" )
        labelPerdiste.setHidden(false)
        labelPuntaje.setHidden(false)
        //caso contrario no existe puntaje alto
    }else{
        labelPerdiste.setText("Puntaje Alto:
        \ \(puntajeGuardado)" )
        labelPuntaje.setText("Puntaje: \ \(puntaje)" )
        labelPerdiste.setHidden(false)
        labelPuntaje.setHidden(false)
    }
}
}

```

```

        // Se valida si no existe un puntaje alto previamente, el
        puntaje nuevo es el mayor
        if(puntajeGuardado == 0 && puntaje > 0){
            if(puntaje > puntajeGuardado){
                puntajeGuardado = puntaje

NSUserDefaults.standardUserDefaults().setInteger(puntajeGuardado,
forKey: "puntajeAlto")
                labelPerdiste.setText("Puntaje Alto:
\\(puntajeGuardado)" )
                labelPuntaje.setText("Puntaje mas alto" )
                labelPerdiste.setHidden(false)
                labelPuntaje.setHidden(false)
            }else{
                labelPerdiste.setText("Puntaje Alto:
\\(puntajeGuardado)" )
                labelPuntaje.setText("Puntaje: \\(puntaje)" )
                labelPerdiste.setHidden(false)
                labelPuntaje.setHidden(false)
            }
        }
    }

    override func didDeactivate() {
        // This method is called when watch view controller is no
        longer visible
        super.didDeactivate()
        NSUserDefaults.standardUserDefaults().setInteger(0, forKey:
        "puntajeJuego")
    }
}
}

```

ANEXO B: NIVELCONTROLLER

```

//
// NivelController.swift
// PanchoMania
//
// Created by Francisco Foyain Lara on 8/10/15.
// Copyright © 2015 FranciscoFoyain. All rights reserved.
//

import WatchKit
import Foundation

class NivelController: WKInterfaceController {

    var numeroNivel=1
    var nivelLetras:String?
    //Elementos que se mostraran en el picker
    var itemList : [(String,String)] = [
        ("Basico", "1"),
        ("Principiante", "2"),
        ("Profesional", "3"),
        ("Avanzado", "4"),
        ("Pancho", "5")

    ]

    //Se crean outlets de conexion del picker y slider

    @IBOutlet var pickerOutlet: WKInterfacePicker!
    @IBOutlet var sliderOutlet: WKInterfaceSlider!

    //Funcion que cambia el valor del slider, se resta uno para
    mantener estructura de valor de 1 a 5.
    @IBAction func actualizarNivelSlider(value: Float) {
        //NSLog("\(value)")

        numeroNivel = Int(value)
        //NSLog("\(numeroNivel)")

        pickerOutlet.setSelectedItemIndex(numeroNivel-1)
    }

    //Al aceptar se guarda el valor seleccionado en el UserDefaults
    con key dificultad actual, si no se selecciona nada se supone que es

```

```

valor 1.

    @IBAction func nivelActivado() {

        if
NSUserDefaults.standardUserDefaults().integerForKey("dificultadActua
l") == 0 {
            numeroNivel = 1

NSUserDefaults.standardUserDefaults().setInteger(numeroNivel,
forKey: "dificultadActual")
            NSUserDefaults.standardUserDefaults().synchronize()

        }else{

NSUserDefaults.standardUserDefaults().setInteger(numeroNivel,
forKey: "dificultadActual")
            NSUserDefaults.standardUserDefaults().synchronize()
        }

        self.popController()

    }
    //se cambia el valor del picker del mapa .1 que es el valor
    numerico y no del titulo

    @IBAction func pickerSelectedItem(value: Int) {

        nivelLetras = itemList[value].1
        numeroNivel = Int(nivelLetras!)
        sliderOutlet.setValue(Float(numeroNivel))

    }
    // se inicia el picker con los elementos creados al comienzo de
    la clase
    override func awakeWithContext(context: AnyObject?) {
        super.awakeWithContext(context)

        let pickerItems: [WKPickerItem] = itemList.map {
            let pickerItem = WKPickerItem()
            pickerItem.caption = $0.0
            pickerItem.title = $0.1

            return pickerItem
        }
        pickerOutlet.setItems(pickerItems)

        // Configure interface objects here.

```

```
}  
  
    override func willActivate() {  
        // This method is called when watch view controller is about  
to be visible to user  
        super.willActivate()  
  
    }  
  
    override func didDeactivate() {  
        // This method is called when watch view controller is no  
longer visible  
        super.didDeactivate()  
    }  
}
```


ANEXO C: REPETICIONES CONTROLLER

```

//
// RepeticionesController.swift
// PanchoMania
//
// Created by ffoyain on 11/11/15.
// Copyright © 2015 FranciscoFoyain. All rights reserved.
//

import WatchKit
import Foundation

class RepeticionesController: WKInterfaceController {

    var repeticiones = 1
    var indexSlider = 3
    var repeticionesLetras:String?
    //se inicializan los valores que tendra el picker.Con 2
    //atributos 0.0 titulos 0.1 valor
    var itemList : [(String,String)] = [
        ("Pocas", "3"),
        ("Varias", "6"),
        ("Muchas", "9")
    ]

    //se crean los outlets de conexion

    @IBOutlet var pickerOutlet: WKInterfacePicker!

    @IBOutlet var sliderOutlet: WKInterfaceSlider!

    // Al aplasta aceptar se verifican las repeticiones elegidas, si no
    // se selecciono nada se supone que es 3. Y se regresa al menu
    // anterior.
    @IBAction func repeticionesActivado() {
        if
        UserDefaults.standardUserDefaults().integerForKey("repeticionesAct
        ual") == 0 || repeticiones == 1 {
            repeticiones = 3

        UserDefaults.standardUserDefaults().setInteger(repeticiones,
        forKey: "repeticionesActual")
            UserDefaults.standardUserDefaults().synchronize()

        }else{

        UserDefaults.standardUserDefaults().setInteger(repeticiones,

```

```

forKey: "repeticionesActual")
    NSUserDefaults.standardUserDefaults().synchronize()
}

self.popController()

}
//Se usa la opcion de picker que viene conectada con el digital
crown. El valor es el que se encuentra en el mapa en posicion .1
@IBAction func pickerSelectedItem(value: Int) {

    repeticionesLetras = itemList[value].1
    repeticiones = Int(repeticionesLetras!)

    sliderOutlet.setValue(Float(repeticiones))
    //NSLog("Repeticiones picker:\(repeticiones)")

}
// Se selecciona el valor en el slider. Se resta uno para
mantener la estructura. Se divide para 3 para hacer el formato 3-6-
9
@IBAction func actualizarRepeticionSlider(value: Float) {

    //NSLog("\(value)")

    repeticiones = Int(value)
    //NSLog("Repeticiones slider:\(repeticiones)")

    indexSlider = repeticiones/3
    pickerOutlet.setSelectedItemIndex(indexSlider-1)

}

// Se agregan al picker los elementos creados en el mapa
inicial.
override func awakeWithContext(context: AnyObject?) {
    super.awakeWithContext(context)

    let pickerItems: [WKPickerItem] = itemList.map {
        let pickerItem = WKPickerItem()
        pickerItem.caption = $0.0
        pickerItem.title = $0.1

        return pickerItem
    }
    pickerOutlet.setItems(pickerItems)
}

```

```
    // Configure interface objects here.
}

override func willActivate() {
    // This method is called when watch view controller is about
to be visible to user
    super.willActivate()
}

override func didDeactivate() {
    // This method is called when watch view controller is no
longer visible
    super.didDeactivate()

}

}
```

ANEXO D: OPCIONCONTROLLER

```

//
// OpcionController.swift
// PanchoMania
//
// Created by ffoyain on 12/11/15.
// Copyright © 2015 FranciscoFoyain. All rights reserved.
//

import WatchKit
import Foundation

class OpcionController: WKInterfaceController {

    //Accion de aplastar en nivel se abre nivel
    @IBAction func abrirNivel() {
        self.pushControllerWithName("nivel", context: self);
    }
    //Accion de aplastar en repeticiones se abre repeticiones
    @IBAction func abrirRepeticiones() {
        self.pushControllerWithName("repeticion", context: self);
    }

    override func awakeWithContext(context: AnyObject?) {
        super.awakeWithContext(context)

        // Configure interface objects here.
    }

    override func willActivate() {
        // This method is called when watch view controller is about
        // to be visible to user
        super.willActivate()
    }

    override func didDeactivate() {
        // This method is called when watch view controller is no
        // longer visible
        super.didDeactivate()
    }
}

```

ANEXO E: JUEGOCOLORES CONTROLLER

```

//
// JuegoColoresController.swift
// PanchoMania
//
// Created by Francisco Foyain Lara on 7/10/15.
// Copyright © 2015 FranciscoFoyain. All rights reserved.
//

import WatchKit
import Foundation

class JuegoColoresController: WKInterfaceController {

    var repeticiones: Int?
    var dificultadColores : Int?
    var tiempoDuracion : Double?
    var tiempoPausa : Double?
    var valorRepeticiones : Int?
    var puntajeJuegoMarcha: Int?

    //outlets de conexion
    @IBOutlet var boton1Outlet: WKInterfaceButton!

    @IBOutlet var boton2Outlet: WKInterfaceButton!

    @IBOutlet var boton3Outlet: WKInterfaceButton!

    @IBOutlet var boton4Outlet: WKInterfaceButton!

    @IBOutlet var notificacionOutlet: WKInterfaceLabel!

    //acciones de botones se ejecuta la funcion evento con la
    posicion de cada boton
    @IBAction func boton1Accion() {
        WKInterfaceDevice.currentDevice().playHaptic(.Notification)

        self.eventoBoton(0)
    }

    @IBAction func boton2Accion() {
        WKInterfaceDevice.currentDevice().playHaptic(.Notification)

        self.eventoBoton(1)
    }
}

```

```

@IBAction func boton3Accion() {
    WKInterfaceDevice.currentDevice().playHaptic(.Notification)

    self.eventoBoton(2)
}

@IBAction func boton4Accion() {

    WKInterfaceDevice.currentDevice().playHaptic(.Notification)

    self.eventoBoton(3)

}

var bloqueando: Bool = true

var ingresosJugador: Array<Int> = []
var serieActual: Array<Int>!
var turnoActual: Int = 0

override func awakeWithContext(context: AnyObject?) {

    super.awakeWithContext(context)

    // Configure interface objects here.
}

override func willActivate() {
    // This method is called when watch view controller is about
to be visible to user
    super.willActivate()
    // se carga desde el inicio el valor del nivel y de
repeticiones, el turno se empieza desde cero.
    turnoActual = 0;
    repeticiones =
NSUserDefaults.standardUserDefaults().integerForKey("repeticionesAct
ual")

    if(repeticiones == 1 || repeticiones == 0){
        repeticiones = 3
    }

    dificultadColores =
NSUserDefaults.standardUserDefaults().integerForKey("dificultadActua
l")
    if(dificultadColores == 0 || dificultadColores == nil){

```

```

        dificultadColores = 1
    }

    iniciarJuego()
}

override func didDeactivate() {
    // This method is called when watch view controller is no
    longer visible
    super.didDeactivate()
}

// se crea un arreglo de colores, son los colores que se
presentan en el juego.
let colores =
[UIColor.yellowColor(),UIColor.redColor(),UIColor.greenColor(),UICol
or.blueColor()];

// devuelve un arreglo con los botones en base de un indice

func botones() -> Array<WKInterfaceButton>{
    return [boton1outlet, boton2outlet, boton3outlet,
boton4outlet]
}

//Se crea el generador de numero randomicos del 1 al 3 de hasta
1000 numeros.

let numeroRandomico = 1000;
func generarNuevaSerie() -> Array<Int>{
    var arregloSerie = [Int]()
    for var contador = 0; contador < numeroRandomico; contador++
{
        let numero = Int(arc4random_uniform(4))
        arregloSerie.append(numero)
    }
    return arregloSerie
}

//Se crea arreglo con los valores obtenidos del color inicial
usando alpha 0.1

func nuevoColor() -> Array<UIColor>{
    var coloresFinales:Array<UIColor> = []
    for colorInicial in self.colores{
coloresFinales.append(colorInicial.colorWithAlphaComponent(0.1))
    }
    return coloresFinales
}

```

//Cambia el color en el indice dado(obtenido randomicamente) conjunto con la funcion nuevo color y se regresa al color inicial obtenido del arreglo colores

```
func cambiarColor(indice: Int!, duracion: Double!){
    let posicion = botones()[indice]
    let colorFinal = nuevoColor()[indice]

    let colorInicial = colores[indice]
    posicion.setBackgroundColor(colorFinal)
    WKInterfaceDevice.currentDevice().playHaptic(.Click)

    pausa(duracion){
        posicion.setBackgroundColor(colorInicial)
    }
}
```

//Funcion que dara el delay de espera en varias funciones, entre el paso de usuario a reloj, espera que debe tener el boton aclarado, etc.

```
func pausa(tiempoPausa:Double,fin:()->()){
    dispatch_after(dispatch_time(DISPATCH_TIME_NOW,
    Int64(tiempoPausa * Double(NSEC_PER_SEC))
    ),
    dispatch_get_main_queue(), fin)
}
//funcion que recibe los ingresos del jugador

func turnoJugador(){
    bloqueando = false
    notificacionOutlet.setText("Recuerda")
    ingresosJugador = []
}
//crea una serie en base a la serieActual generada, se usa la
funcion cambiar color y se demora el tiempo en base de la dificultad
elegida
func jugarNuevaSerie(indice: Int, turnoFinal: Int){
    if(indice <= 0){
        turnoJugador()
    }else{

        let turno = serieActual[turnoFinal - indice]

        switch dificultadColores as Int!{
        case 1:
            tiempoDuracion = 0.5
```



```

        case 2:
            tiempoDuracion = 0.40

        case 3:
            tiempoDuracion = 0.35

        case 4:
            tiempoDuracion = 0.30

        case 5:
            tiempoDuracion = 0.20

        default:
            tiempoDuracion = 0.5
    }

    cambiarColor(turno, duracion: tiempoDuracion!)

    switch dificultadColores as Int!{
    case 1:
        tiempoPausa = 1.0

    case 2:
        tiempoPausa = 0.9

    case 3:
        tiempoPausa = 0.8

    case 4:
        tiempoPausa = 0.7

    case 5:
        tiempoPausa = 0.6

    default:
        tiempoPausa = 0.5
    }

    pausa(tiempoPausa!){
        self.jugarNuevaSerie(indice - 1, turnoFinal:
turnoFinal)
    }
}

}

// es la funcion que empieza una nueva serie tomando el indice
actual del juego desde iniciarjuego
func jugarSerie(indice: Int){
    jugarNuevaSerie(indice, turnoFinal: indice)
}

```

```

}
// se empieza el juego con jugar serie 1.
func iniciarJuego(){

    turnoActual = 1
    serieActual = generarNuevaSerie()
    bloqueando = true
    notificacionOutlet.setText("Empezando")
    pausa(1){
        self.notificacionOutlet.setText("3")
    }
    pausa(2){
        self.notificacionOutlet.setText("2")
    }
    pausa(3){
        self.notificacionOutlet.setText("1")
    }
    pausa(4){
        self.notificacionOutlet.setText("Mira Bien!")
        WKInterfaceDevice.currentDevice().playHaptic(.Start)
    }
    pausa(5){
        self.jugarSerie(1)
    }
}

//Turno del reloj. Crea la serie en base al indice actual, que
es el turno actual.

func nuevoTurno(){

    switch repeticiones as Int!{

    case 3:
        valorRepeticiones = 3

    case 6:
        valorRepeticiones = 6

    case 9:
        valorRepeticiones = 9

    default:
        valorRepeticiones = 3

    }

    if(turnoActual == valorRepeticiones){
        pausa(1){

```

```

    }
    pausa(2){
        WKInterfaceDevice.currentDevice().playHaptic(.Success)
    }

NSUserDefaults.standardUserDefaults().setInteger(self.turnoActual,
forKey: "puntajeJuego")
        self.pushControllerWithName("juegoTarjetas", context:
nil);

    }else{

        turnoActual+=1
        bloqueando = true
        notificacionOutlet.setText("Sigue Asi")
        pausa(2){

            self.jugarSerie(self.turnoActual)
            self.notificacionOutlet.setText("Tu Turno")
        }

    }
}
// esta es la funcion principal, en esta se recibe los ingresos
del jugador se los guarda en un arreglo y se compara con la
serieActual.
func eventoBoton(posicion: Int){
    if(bloqueando){
        return
    }
    ingresosJugador.append(posicion)

    for(var contador = 0; contador <
ingresosJugador.count;contador++){
        if(serieActual[contador] !=
self.ingresosJugador[contador]){
            terminar()

        }
    }
    if(ingresosJugador.count == turnoActual){
        nuevoTurno()
    }
}
// si se pierde se termina y se regresa al menu inicial,
grabando en NSUserDefaults el valor del turno actual que sera el
puntaje actual.
func terminar(){

    notificacionOutlet.setText("Mal")

    pausa(2){

```

```
        WKInterfaceDevice.currentDevice().playHaptic(.Failure)
NSUserDefaults.standardUserDefaults().setInteger(self.turnoActual,
forKey: "puntajeJuego")
        //NSUserDefaults.standardUserDefaults().setInteger(1,
forKey: "dificultadActual")
        //NSUserDefaults.standardUserDefaults().setInteger(3,
forKey: "repeticionesActual")
        self.notificacionOutlet.setText("PERDISTE")

        //self.pushControllerWithName("menu", context: self);
    }

    pausa(3){
        self.notificacionOutlet.setText("PERDISTE")
        self.popToRootController()

    }

}

}
```

ANEXO F: JUEGOTARJETASCONTROLLER

```

//
// JuegoTarjetasController.swift
// PanchoMania
//
// Created by Francisco Foyain Lara on 21/11/15.
// Copyright © 2015 FranciscoFoyain. All rights reserved.
//

import WatchKit
import Foundation

class JuegoTarjetasController: WKInterfaceController {

    var arregloFinalIndices:Array<Int> = []
    var arregloFinalStrings:Array<String> = []
    var bloqueando: Bool = true
    var ingresosJugador: Array<Int> = []
    var numeroUno:Int?
    var numeroDos:Int?
    var turnoActual:Int?
    var velocidadGiro: Int?

    //se crea los outlets de conexion de botones y labels

    @IBOutlet var outletBoton1: WKInterfaceButton!

    @IBOutlet var outletBoton2: WKInterfaceButton!
    @IBOutlet var outletBoton3: WKInterfaceButton!
    @IBOutlet var outletBoton4: WKInterfaceButton!
    @IBOutlet var outletBoton5: WKInterfaceButton!
    @IBOutlet var outletBoton6: WKInterfaceButton!

    @IBOutlet var labelTexto: WKInterfaceLabel!

    //se crean las acciones de los botones usando la funcion evento
    boton con la posicion de cada boton.

    @IBAction func accionBoton1() {
        WKInterfaceDevice.currentDevice().playHaptic(.Click)

        self.eventoBoton(0)
    }

    @IBAction func accionBoton2() {
        WKInterfaceDevice.currentDevice().playHaptic(.Click)

        self.eventoBoton(1)
    }
}

```

```

@IBAction func accionBoton3() {
    WKInterfaceDevice.currentDevice().playHaptic(.Click)

    self.eventoBoton(2)
}

@IBAction func accionBoton4() {
    WKInterfaceDevice.currentDevice().playHaptic(.Click)

    self.eventoBoton(3)
}

@IBAction func accionBoton5() {
    WKInterfaceDevice.currentDevice().playHaptic(.Click)

    self.eventoBoton(4)
}

@IBAction func accionBoton6() {
    WKInterfaceDevice.currentDevice().playHaptic(.Click)

    self.eventoBoton(5)
}

// esta funcion sirve para cambiar las imagenes de los botones a
la parte de atras de la tarjeta.

func girarTarjetas(){
    outletBoton1.setBackgroundImageNamed("carta")
    outletBoton2.setBackgroundImageNamed("carta")
    outletBoton3.setBackgroundImageNamed("carta")
    outletBoton4.setBackgroundImageNamed("carta")
    outletBoton5.setBackgroundImageNamed("carta")
    outletBoton6.setBackgroundImageNamed("carta")
}

// funcion que da delay de un tiempo dado.

func pausa(tiempoPausa:Double,fin:()->()){

    dispatch_after(dispatch_time(DISPATCH_TIME_NOW,
Int64(tiempoPausa * Double(NSEC_PER_SEC))
    ),
        dispatch_get_main_queue(), fin)
}

//si se pierde se guarda el turno actual en puntajejuego y se
regresa al menu inicial(root)
func terminar(){
NSUserDefaults.standardUserDefaults().setInteger(turnoActual!,

```

```

forKey: "puntajeJuego")
    popToRootController()
}

//la funcion principal del juego. Espera una accion de
cualquiera de los botones y recibe su posicion como parametro.

func eventoBoton(posicion: Int){

    if(bloqueando == true){
        return
    }else{
        /* para cada posicion realiza la misma funcion en base
de la posicion recibida:

        *Se gira la tarjeta que se selecciono
        *se guarda la posicion en un arreglo, si esta posicion
ya existe no se hace nada y se retorna
        * se guarda la posicion en numero uno si numero uno y
numero dos estan vacios
        * caso contrario se guarda en numero 2
        * si numero uno y numero dos son iguales se suma uno al
turno caso contrario se pierde y se ejecuta terminar.

        */
        if (posicion == 0){

outletBoton1.setBackgroundImageNamed(arregloFinalStrings[arregloFina
lIndices[0]])
            ingresosJugador.append(posicion)
            ///NSLog("index guardado
\\(arregloFinalIndices[0])")

            if(numeroUno == nil && numeroDos == nil){
                numeroUno = arregloFinalIndices[0]
                labelTexto.setText("Encuentra el par")

            }else if(numeroUno >= 0 && numeroDos == nil){
                numeroDos = arregloFinalIndices[0]
            }

            if(numeroUno >= 0 && numeroDos >= 0){

                if (numeroUno == numeroDos){
                    turnoActual = turnoActual!+1
                    labelTexto.setText("Bien Sigue Asi")
                    ///NSLog("Numero1 \\(numeroUno)")
                    ///NSLog("Numero2 \\(numeroDos)")
                    //NSLog("TURNO: \\(turnoActual)")
                    numeroUno = nil

```

```

        numeroDos = nil
    }else{
        numeroUno = nil
        numeroDos = nil
        labelTexto.setText("Perdiste")
        bloqueando = true

        pausa(2){

            self.terminar()
            //NSLog("TERMINADO")
        }
    }
}

if (posicion == 1){

outletBoton2.setBackgroundImageNamed(arregloFinalStrings[arregloFinalIndices[1]])
    ingresosJugador.append(posicion)
    ///NSLog("index guardado
\\(arregloFinalIndices[1])")

    if(numeroUno == nil && numeroDos == nil){
        numeroUno = arregloFinalIndices[1]
        labelTexto.setText("Encuentra el par")

    }else if(numeroUno >= 0 && numeroDos == nil){
        numeroDos = arregloFinalIndices[1]
    }

    if(numeroUno >= 0 && numeroDos >= 0){

        if (numeroUno == numeroDos){
            turnoActual = turnoActual!+1
            labelTexto.setText("Bien Sigue Asi")
            ///NSLog("Numero1 \\(numeroUno)")
            ///NSLog("Numero2 \\(numeroDos)")
            //NSLog("TURN0: \\(turnoActual)")
            numeroUno = nil
            numeroDos = nil
        }else{
            numeroUno = nil
            numeroDos = nil
            labelTexto.setText("Perdiste")
            bloqueando = true

            pausa(2){

                self.terminar()

```



```

        //NSLog("TERMINADO")
    }
}

if (posicion == 2){

outletBoton3.setBackgroundImageNamed(arregloFinalStrings[arregloFinalIndices[2]])
    ingresosJugador.append(posicion)
    ///NSLog("index guardado
\\(arregloFinalIndices[2])")

    if(numeroUno == nil && numeroDos == nil){
        numeroUno = arregloFinalIndices[2]
        labelTexto.setText("Encuentra el par")

    }else if(numeroUno >= 0 && numeroDos == nil){
        numeroDos = arregloFinalIndices[2]
    }

    if(numeroUno >= 0 && numeroDos >= 0){

        if (numeroUno == numeroDos){
            turnoActual = turnoActual!+1
            labelTexto.setText("Bien Sigue Asi")
            ///NSLog("Numero1 \\(numeroUno)")
            ///NSLog("Numero2 \\(numeroDos)")
            //NSLog("TURNO: \\(turnoActual)")
            numeroUno = nil
            numeroDos = nil
        }else{
            numeroUno = nil
            numeroDos = nil
            labelTexto.setText("Perdiste")
            bloqueando = true
            pausa(2){

                self.terminar()
                //NSLog("TERMINADO")
            }
        }
    }

}

if (posicion == 3){

outletBoton4.setBackgroundImageNamed(arregloFinalStrings[arregloFinalIndices[3]])

```

```

        ingresosJugador.append(posicion)
        ///NSLog("index guardado
        \((arregloFinalIndices[3]))")

        if(numeroUno == nil && numeroDos == nil){
            numeroUno = arregloFinalIndices[3]
            labelTexto.setText("Encuentra el par")

        }else if(numeroUno >= 0 && numeroDos == nil){
            numeroDos = arregloFinalIndices[3]
        }

        if(numeroUno >= 0 && numeroDos >= 0){

            if (numeroUno == numeroDos){
                turnoActual = turnoActual!+1
                labelTexto.setText("Bien Sigue Asi")
                ///NSLog("Numero1 \((numeroUno)")
                ///NSLog("Numero2 \((numeroDos)")
                //NSLog("TURNO: \((turnoActual)")
                numeroUno = nil
                numeroDos = nil
            }else{
                numeroUno = nil
                numeroDos = nil
                labelTexto.setText("Perdiste")
                bloqueando = true
            }

            WKInterfaceDevice.currentDevice().playHaptic(.Failure)

            pausa(2){

                self.terminar()
                ///NSLog("TERMINADO")
            }
        }
    }

    if (posicion == 4){

        outletBoton5.setBackgroundImageNamed(arregloFinalStrings[arregloFinalIndices[4]])

        ingresosJugador.append(posicion)
        ///NSLog("index guardado
        \((arregloFinalIndices[4]))")

        if(numeroUno == nil && numeroDos == nil){
            numeroUno = arregloFinalIndices[4]
            labelTexto.setText("Encuentra el par")
        }
    }
}

```

```

}else if(numeroUno >= 0 && numeroDos == nil){
    numeroDos = arregloFinalIndices[4]
}

if(numeroUno >= 0 && numeroDos >= 0){

    if (numeroUno == numeroDos){
        turnoActual = turnoActual!+1
        labelTexto.setText("Bien Sigue Asi")
        ///NSLog("Numero1 \ (numeroUno)")
        ///NSLog("Numero2 \ (numeroDos)")
        //NSLog("TURNO: \ (turnoActual)")
        numeroUno = nil
        numeroDos = nil
    }else{
        numeroUno = nil
        numeroDos = nil
        labelTexto.setText("Perdiste")
        bloqueando = true
        pausa(2){

            self.terminar()
            //NSLog("TERMINADO")
        }
    }
}

}

if (posicion == 5){

outletBoton6.setBackgroundImageNamed(arregloFinalStrings[arregloFinalIndices[5]])
    ingresosJugador.append(posicion)
    // //NSLog("index guardado
    \ (arregloFinalIndices[5])")

    if(numeroUno == nil && numeroDos == nil){
        numeroUno = arregloFinalIndices[5]
        labelTexto.setText("Encuentra el par")

    }else if(numeroUno >= 0 && numeroDos == nil){
        numeroDos = arregloFinalIndices[5]
    }

    if(numeroUno >= 0 && numeroDos >= 0){

        if (numeroUno == numeroDos){
            turnoActual = turnoActual!+1
            labelTexto.setText("Bien Sigue Asi")
            ///NSLog("Numero1 \ (numeroUno)")

```

```

        ///NSLog("Numero2 \((numeroDos)")
        //NSLog("TURNO: \((turnoActual)")
        numeroUno = nil
        numeroDos = nil
    }else{
        numeroUno = nil
        numeroDos = nil
        labelTexto.setText("Perdiste")

        pausa(2){
            self.terminar()
            //NSLog("TERMINADO")
        }
    }
}

if ingresosJugador.count == 6{
    labelTexto.setText("Lo lograste")

NSUserDefaults.standardUserDefaults().setInteger(turnoActual!,
forKey: "puntajeJuego")

    pausa(2){
        self.pushControllerWithName("numeros", context:
nil);
        //NSLog("TERMINADO TARJETAS")
    }

}

}

//se crea un arreglo de strings de los nombres de las 13
imagenes a utilizar. Y se devuelve este arreglo

func crearArregloString() -> Array<String>{

    var arregloStringImágenes: Array<String> = []
    arregloStringImágenes.append("corazon")
    arregloStringImágenes.append("fball")
    arregloStringImágenes.append("bball")
    arregloStringImágenes.append("bug")
    arregloStringImágenes.append("telefono")
    arregloStringImágenes.append("cartas")
    arregloStringImágenes.append("reloj")
    arregloStringImágenes.append("foco")
    arregloStringImágenes.append("globos")
    arregloStringImágenes.append("hielo")
    arregloStringImágenes.append("nube")
}

```

```

    arregloStringImagenes.append("huevo")
    arregloStringImagenes.append("manzana")

    return arregloStringImagenes
}

// se genera un arreglo de los strings en orden aleatorio. Se
// genera un total de 3 strings.

func generarArregloAleatorioStrings(arreglo: Array<String>) ->
Array<String>{

    var arregloNumerosAleatoriosUsados:Array<Int> = []
    var arregloStrings:Array<String> = []
    for (var contador = 0 ; contador < 3 ; contador++) {

        var numero = Int(arc4random_uniform(13))

        while (
arregloNumerosAleatoriosUsados.contains(numero)){
            numero = Int(arc4random_uniform(13))
        }
        ///NSLog("Numero arreglo de 3 \(numero)")
        arregloNumerosAleatoriosUsados.append(numero)
        arregloStrings.append(arreglo[numero])
    }

    return arregloStrings
}

//Esta funcion recibe el arreglo de 3 strings, y genera de nuevo
//aleatoriamente un arreglo aleatorio de estos datos para arriba y
//abajo excluyendo los ya usados, el arreglo final se devuelve tanto
//de posiciones(indices) como de strings.

func crearArregloFinal(arreglo: Array<String>) -> (strings:
Array<String>, enteros: Array<Int>){
    var arregloFilaArriba: Array<Int> = []
    var arregloFilaAbajo: Array<Int> = []
    var arregloNumeros: Array<Int> = []
    var arregloStrings: Array<String> = []

    for var contador = 0; contador < 3; contador++ {
        var numeroArriba = Int(arc4random_uniform(3))
        while ( arregloFilaArriba.contains(numeroArriba)){
            numeroArriba = Int(arc4random_uniform(3))
        }
        ///NSLog("NumeroArriba \(numeroArriba)")

        arregloFilaArriba.append(numeroArriba)
        arregloNumeros.append(numeroArriba)
    }
    for var contador = 0; contador < 3; contador++ {

```

```

        var numeroAbajo = Int(arc4random_uniform(3))
        while ( arregloFilaAbajo.contains(numeroAbajo)){
            numeroAbajo = Int(arc4random_uniform(3))
        }
        // NSLog("NumeroAbajo \(numeroAbajo)")
        arregloFilaAbajo.append(numeroAbajo)
        arregloNumeros.append(numeroAbajo)
    }

    for var i = 0; i < arregloNumeros.count; i++ {
        arregloStrings.append(arreglo[arregloNumeros[i]])
        ///NSLog("Texto Arreglo \(arreglo[arregloNumeros[i]])")
    }

    return (arregloStrings,arregloNumeros)
}

// funcion encargada de llenar los botones(tarjetas) con un
arreglo recibido, arreglo de crearArregloFinal

func llenarBotones(enteros: Array<Int>, textos: Array<String>){

    let botones = [outletBoton1, outletBoton2, outletBoton3,
outletBoton4, outletBoton5, outletBoton6]
    var i:Int = 0
    for boton in botones {
        boton.setBackgroundImageNamed(textos[enteros[i]])
        ++i
    }
}

//se inicia el juego.
func iniciar(){

    let arregloStrings:Array<String> = crearArregloString()
    let arregloAleatorios:Array<String> =
generarArregloAleatorioStrings(arregloStrings)
    let arregloFinal = crearArregloFinal(arregloAleatorios)
    arregloFinalStrings = arregloFinal.strings
    arregloFinalIndices = arregloFinal.enteros

    self.outletBoton1.setBackgroundImageNamed("carta")
    self.outletBoton2.setBackgroundImageNamed("carta")
    self.outletBoton3.setBackgroundImageNamed("carta")
    self.outletBoton4.setBackgroundImageNamed("carta")
    self.outletBoton5.setBackgroundImageNamed("carta")
    self.outletBoton6.setBackgroundImageNamed("carta")
}

```

```
bloqueando = true

labelTexto.setText(" ")

pausa(1){
    self.labelTexto.setText("3")
}
pausa(2){
    self.labelTexto.setText("2")
}
pausa(3){
    self.labelTexto.setText("1")
}
pausa(4){
    self.labelTexto.setText("Mira las cartas!")
    self.llenarBotones(self.arregloFinalIndices, textos:
self.arregloFinalStrings)
}

if(velocidadGiro == nil || velocidadGiro == 0){
    velocidadGiro = 1
}

if(velocidadGiro == 1){
    pausa(8){
        self.girarTarjetas()

        self.bloqueando = false
    }
}

if(velocidadGiro == 2){
    pausa(7){

        self.girarTarjetas()
        self.bloqueando = false
    }
}

if(velocidadGiro == 3){
    pausa(7){
```

```

        self.girarTarjetas()
        self.bloqueando = false
    }
}

if(velocidadGiro == 4){
    pausa(6){
        self.girarTarjetas()
        self.bloqueando = false
    }
}

if(velocidadGiro == 5){
    pausa(6){
        self.girarTarjetas()
        self.bloqueando = false
    }
}

///NSLog("ContarStrings \(arregloFinalStrings.count)")
///NSLog("ContarEnteros \(arregloFinalIndices.count)")
}

override func awakeWithContext(context: AnyObject?) {
    super.awakeWithContext(context)

    // Configure interface objects here.
}

override func willActivate() {
    // This method is called when watch view controller is about
    // to be visible to user
    // Se guardan los datos del turno actual del anterior juego
    // y de la dificultad para los delays.
    super.willActivate()
    turnoActual =
    UserDefaults.standardUserDefaults().integerForKey("puntajeJuego")
    velocidadGiro =
    UserDefaults.standardUserDefaults().integerForKey("dificultadActua
l")
    ///NSLog("TURNO \(turnoActual)")
    ///NSLog("DIFICULTAD \(velocidadGiro)")
}

```



```
        iniciar()
    }

    override func didDeactivate() {
        // This method is called when watch view controller is no
        longer visible
        super.didDeactivate()
    }
}
```

ANEXO G: JUEGONUMEROSCONTROLLER

```

//
// JuegoNumerosController.swift
// PanchoMania
//
// Created by Francisco Foyain Lara on 20/11/15.
// Copyright © 2015 FranciscoFoyain. All rights reserved.
//

import WatchKit
import Foundation

class JuegoNumerosController: WKInterfaceController {

    var ingresosJugador: Array<Int> = []
    var arregloRandomico:Array<Int> = []
    var arregloUsados:Array<Int> = []
    var arregloEnOrden:Array<Int> = []
    var bloqueando: Bool = true
    var turnoActual: Int?
    var numeroRand = 9
    var contador:Int = 0
    var dificultad:Int = 0
    var miTimer: NSTimer?
    var tiempo:NSTimeInterval?

    //se crean outlets del timer, texto y botones

    @IBOutlet var outletTimer: WKInterfaceTimer!
    @IBOutlet var outletBoton1: WKInterfaceButton!
    @IBOutlet var outletBoton2: WKInterfaceButton!
    @IBOutlet var outletBoton3: WKInterfaceButton!
    @IBOutlet var outletBoton4: WKInterfaceButton!
    @IBOutlet var outletBoton5: WKInterfaceButton!
    @IBOutlet var outletBoton6: WKInterfaceButton!
    @IBOutlet var outletBoton7: WKInterfaceButton!
    @IBOutlet var outletBoton8: WKInterfaceButton!
    @IBOutlet var outletBoton9: WKInterfaceButton!
    @IBOutlet var outletTexto: WKInterfaceLabel!

    // se generan las conexiones de los acciones de los diferentes
    botones.

    @IBAction func accionBoton1() {
    WKInterfaceDevice.currentDevice().playHaptic(.Click)
        eventoBoton(0)}

    @IBAction func accionBoton2() {
    WKInterfaceDevice.currentDevice().playHaptic(.Click)

```

```

eventoBoton(1)}

@IBAction func accionBoton3() {
WKInterfaceDevice.currentDevice().playHaptic(.Click)
eventoBoton(2)}

@IBAction func accionBoton4() {
WKInterfaceDevice.currentDevice().playHaptic(.Click)
eventoBoton(3)}

@IBAction func accionBoton5() {
WKInterfaceDevice.currentDevice().playHaptic(.Click)
eventoBoton(4)}

@IBAction func accionBoton6() {
WKInterfaceDevice.currentDevice().playHaptic(.Click)
eventoBoton(5)}

@IBAction func accionBoton7() {
WKInterfaceDevice.currentDevice().playHaptic(.Click)
eventoBoton(6)}

@IBAction func accionBoton8() {
WKInterfaceDevice.currentDevice().playHaptic(.Click)
eventoBoton(7)}

@IBAction func accionBoton9() {
WKInterfaceDevice.currentDevice().playHaptic(.Click)
eventoBoton(8)}

// Funcion principal encargada de recibir el ingreso del usuario
de los botones si no esta activado el booleano bloqueando.

func eventoBoton(posicion: Int){

    if(arregloUsados.contains(posicion)){
        return
    }
    if(bloqueando){
        return
    }
    // se agrega al arreglo ingresosJugador la posicion
    ingresada tomada de la posicion del arreglo randomico.
    ingresosJugador.append(arregloRandomico[posicion])
    //si el ingreso del jugador y el arreglo ordenado
    descendente es el mismo se suma uno, y se eliminan los valores en
    posicion cero de los arreglos ingresosJugador y arregloEnOrden. Se
    suma 1 al turno que es el puntaje.
    if (ingresosJugador[0] == arregloEnOrden[0]){
        contador=contador+1

        outletTexto.setText("Vas Bien")
        arregloUsados.append(posicion)

```

```

    ingresosJugador.removeFirst()
    arregloEnOrden.removeFirst()
    turnoActual = turnoActual! + 1
    //NSLog("TURNO \ (turnoActual)")

    switch (posicion){
    case 0:
        outletBoton1.setTitle(" ")
    case 1:
        outletBoton2.setTitle(" ")
    case 2:
        outletBoton3.setTitle(" ")
    case 3:
        outletBoton4.setTitle(" ")
    case 4:
        outletBoton5.setTitle(" ")
    case 5:
        outletBoton6.setTitle(" ")
    case 6:
        outletBoton7.setTitle(" ")
    case 7:
        outletBoton8.setTitle(" ")
    case 8:
        outletBoton9.setTitle(" ")
    default:
        return
    }

}

}else{
    // si no son los mismos se pierde.
    bloqueando = true
    outletTimer.stop()
    self.outletTexto.setText("Perdiste")

WKInterfaceDevice.currentDevice().playHaptic(.Failure)

    pausa(2){
        self.outletTexto.setText("Perdiste")
        self.terminar()
    }
    pausa(3){
        self.terminar()
    }

}

}

//NSLog("jugador \ (contador)")
// si el contador es 9, se gana el juego completo y se
ejecuta terminar.
if contador == 9{

    outletTexto.setText("BIEN")
    outletTimer.stop()

```

```

        WKInterfaceDevice.currentDevice().playHaptic(.Success)

        pausa(1){
            self.outletTexto.setText("GANASTE")
        }
        pausa(2){
            self.outletTexto.setText("GANASTE")
            self.terminar()
        }
    }

}
// se guarda el turno actual en puntajeJuego
func terminar(){

NSUserDefaults.standardUserDefaults().setInteger(turnoActual!,
forKey: "puntajeJuego")
//NSLog("TERMINADO")

    popToRootController()
}
// se crea un arreglo de 9 numeros aleatorios entre 1 y 100 sin
utilizar los ya usados.

func crearArregloNumerosAleatorios() -> Array<Int> {

    var arregloNumerosAleatoriosUsados: Array <Int> = []

    for var contador = 0; contador < numeroRand; contador++ {
        var numero = Int(arc4random_uniform(100))

        while (arregloNumerosAleatoriosUsados.contains(numero)){
            numero = Int(arc4random_uniform(100))
        }
        //NSLog("IMPRIMIR numero \(numero)")

        arregloNumerosAleatoriosUsados.append(numero)
    }

    return arregloNumerosAleatoriosUsados
}

//se llenan los botones con los numeros creados del arreglo
aleatorio
func llenarBotones(arreglo: Array<Int>){

    let botones = [outletBoton1, outletBoton2, outletBoton3,
outletBoton4, outletBoton5, outletBoton6,
outletBoton7, outletBoton8, outletBoton9]
    var i: Int = 0

```

```

    for boton in botones {
        boton.setTitle("\(arreglo[i]")
        ++i
    }
}

//se ordena el arreglo descendientemente
func arreglarArreglo(arreglo: Array<Int>) -> Array<Int>{
    let arreglado:Array<Int> = arreglo.sort(>)
    return arreglado
}

//funcion para dar delay.
func pausa(tiempoPausa:Double,fin:()->()){

    dispatch_after(dispatch_time(DISPATCH_TIME_NOW,
Int64(tiempoPausa * Double(NSEC_PER_SEC))
    ),
        dispatch_get_main_queue(), fin)
}

// se inicia el juego.
func iniciar(){

    bloqueando = true

    arregloRandomico = self.crearArregloNumerosAleatorios()
    self.llenarBotones(arregloRandomico)
    arregloEnOrden = arreglarArreglo(arregloRandomico)

    pausa(0){
        self.bloqueando = true

        self.outletTexto.setText("ARREGLA")
    }
    pausa(1){
        self.bloqueando = true

        self.outletTexto.setText("DE")
    }

    pausa(2){
        self.bloqueando = true

        self.outletTexto.setText("MAYOR")
    }

    pausa(3){
        self.outletTexto.setText("A MENOR")
    }

    pausa(4){
        self.outletTexto.setText("AHORA")
        self.bloqueando = false
    }
}

```

```

        self.miTimer =
NSTimer.scheduledTimerWithTimeInterval(self.tiempo!, target: self,
selector: Selector("finTimer"), userInfo: nil, repeats: false)
        self.outletTimer.setDate(NSDate(timeIntervalSinceNow:
self.tiempo!))
        self.outletTimer.start()
    }

}

// el timer para su tiempo y se termina el juego.
func finTimer(){

    outletTexto.setText("Timpo acabado")
    pausa(1){
        self.outletTexto.setText("Timpo acabado")
    }
    terminar()
}

override func awakeWithContext(context: AnyObject?) {
    super.awakeWithContext(context)

    // Configure interface objects here.
}

override func willActivate() {
    // This method is called when watch view controller is about
to be visible to user
    super.willActivate()
    bloqueando = true
    dificultad =
NSUserDefaults.standardUserDefaults().integerForKey("dificultadActua
l")
    //NSLog("\(dificultad)")
    turnoActual =
NSUserDefaults.standardUserDefaults().integerForKey("puntajeJuego")

    if (dificultad == 0 || dificultad == 1){
        tiempo = 26
    }

    if (dificultad == 2){
        tiempo = 21
    }

    if (dificultad == 3){
        tiempo = 16
    }
}

```

```
    }  
  
    if (dificultad == 4){  
        tiempo = 16  
    }  
  
    if (dificultad == 5){  
        tiempo = 11  
    }  
  
    //NSLog("TURNO ACTUAL\%(turnoActual)")  
  
    iniciar()  
  
}  
  
    override func didDeactivate() {  
        // This method is called when watch view controller is no  
        longer visible  
        super.didDeactivate()  
    }  
}
```


ANEXO H: VIEWCONTROLLER

```

//
// ViewController.swift
// PanchoMania
//
// Created by Francisco Foyain Lara on 7/10/15.
// Copyright © 2015 FranciscoFoyain. All rights reserved.
//

import UIKit

class ViewController:
    UIViewController,UIPageViewControllerDataSource,
    UIPageViewControllerDelegate {
    //arreglo de los textos que se van a usar
    let labels = ["Empieza a jugar! o elige el nivel y
repeticiones!", "Puedes configurar la dificultad del juego o el
numero de repeticiones de sigue a Pancho","El nivel sera la
velocidad de Pancho, el tiempo que puedes ver las cartas o el tiempo
que tienes para los numeros","Si quieres hacer un mayor puntaje
elige mas repeticiones del primer juego e intenta ganar el juego en
nivel Pancho","Mira bien el orden de los colores y
repitelos!","Acuerdate donde estaban las tarjetas y encuentralas!",
"Aplasta los numeros de mayor a menor antes de que se acabe el
tiempo. Ahora abre el juego en tu Apple Watch"]
    //arreglo de los nombres de las imagenes que se van a usar.
    var imagenes =
["menu","configuracion","nivel","serie","colores","tarjetas","numero
s"]

    var pageViewController : UIPageViewController!
    //se reinicia el index y se empieza de nuevo.
    @IBAction func empezar(sender: AnyObject) {
        let pageContentViewController =
self.viewControllerEnIndice(0)

self.pageViewController.setViewControllers([pageContentViewControll
er!], direction: UIPageViewControllerNavigationDirection.Forward,
animated: true, completion: nil)

    }
    // esta funcion se ejecuta cuando se abre por primera vez la
aplicacion. Generando el viewcontroller.

    func reiniciar() {
        pageViewController =
self.storyboard?.instantiateViewControllerWithIdentifier("PageViewCo
ntroller") as! UIPageViewController
        self.pageViewController.dataSource = self
    }

```

```

        let pageContentViewController =
self.viewControllerEnIndice(0)

self.pageViewController.setViewControllers([pageContentViewController!], direction: UIPageViewControllerNavigationDirection.Forward,
animated: true, completion: nil)

        self.pageViewController.view.frame = CGRectMake(0, 0,
self.view.frame.width, self.view.frame.height - 40)
        self.addChildViewController(pageViewController)
        self.view.addSubview(pageViewController.view)
        self.pageViewController.didMoveToParentViewController(self)
    }

    override func viewDidLoad() {
        super.viewDidLoad()

        reiniciar()
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

    //funcion encargada de generar el siguiente view controller
    func pageViewController(pageViewController:
UIPageViewController, viewControllerAfterViewController:
UIViewController) -> UIViewController? {

        var index = (viewController as!
PageContentViewController).indexPagina!
        index++
        if(index >= self.imagenes.count){
            return nil
        }
        return self.viewControllerEnIndice(index)
    }

    //funcion encargada de generar el anterior view controller
    func pageViewController(pageViewController:
UIPageViewController, viewControllerBeforeViewController:
UIViewController) -> UIViewController? {

        var index = (viewController as!
PageContentViewController).indexPagina!
        if(index <= 0){
            return nil
        }
        index--
        return self.viewControllerEnIndice(index)
    }

```

```

}

// funcion encargada de ver en que indice se encuentra.

func viewControllerEnIndice(index : Int) -> UIViewController? {
if((self.labels.count == 0) || (index >= self.labels.count)) {
return nil
}

let pageContentViewController =
self.storyboard?.instantiateViewControllerWithIdentifier("PageContent
tViewController") as! PageContentViewController

pageContentViewController.nombreImagen = self.imagenes[index]
pageContentViewController.textoTitulo = self.labels[index]
pageContentViewController.indexPagina = index
return pageContentViewController
}
// contador para los botones de abajo
func presentationCountForPageViewController(pageViewController:
UIPageViewController) -> Int {
return labels.count
}
//contador para iniciar desde el boton 0
func presentationIndexForPageViewController(pageViewController:
UIPageViewController) -> Int {
return 0
}

}

}

```

ANEXO I: PAGECONTENT VIEWCONTROLLER

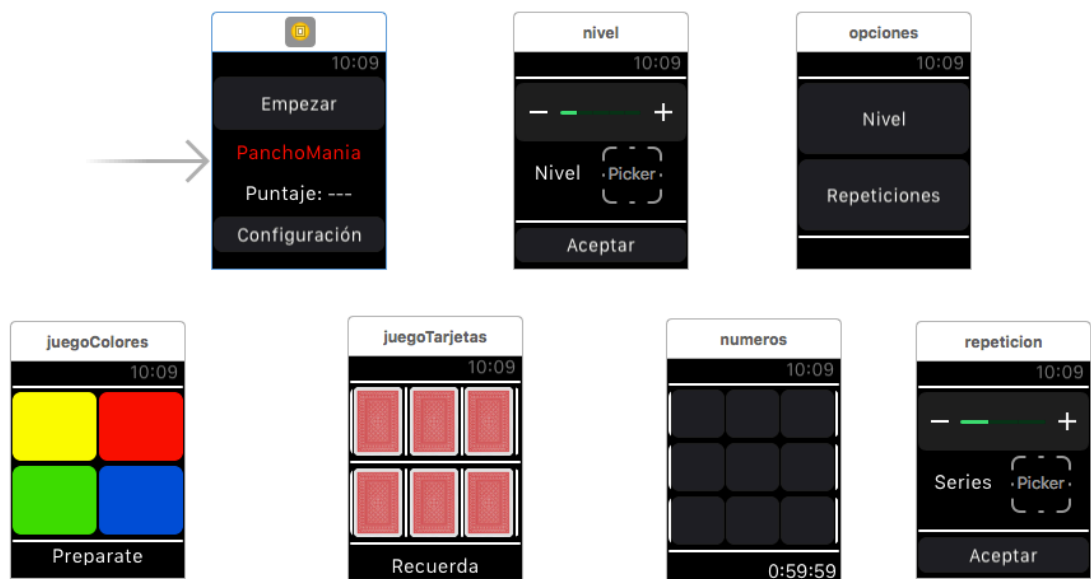
```
//
// PageContentViewController.swift
// PanchoMania
//
// Created by Francisco Foyain Lara on 11/11/15.
// Copyright © 2015 FranciscoFoyain. All rights reserved.
//

import UIKit

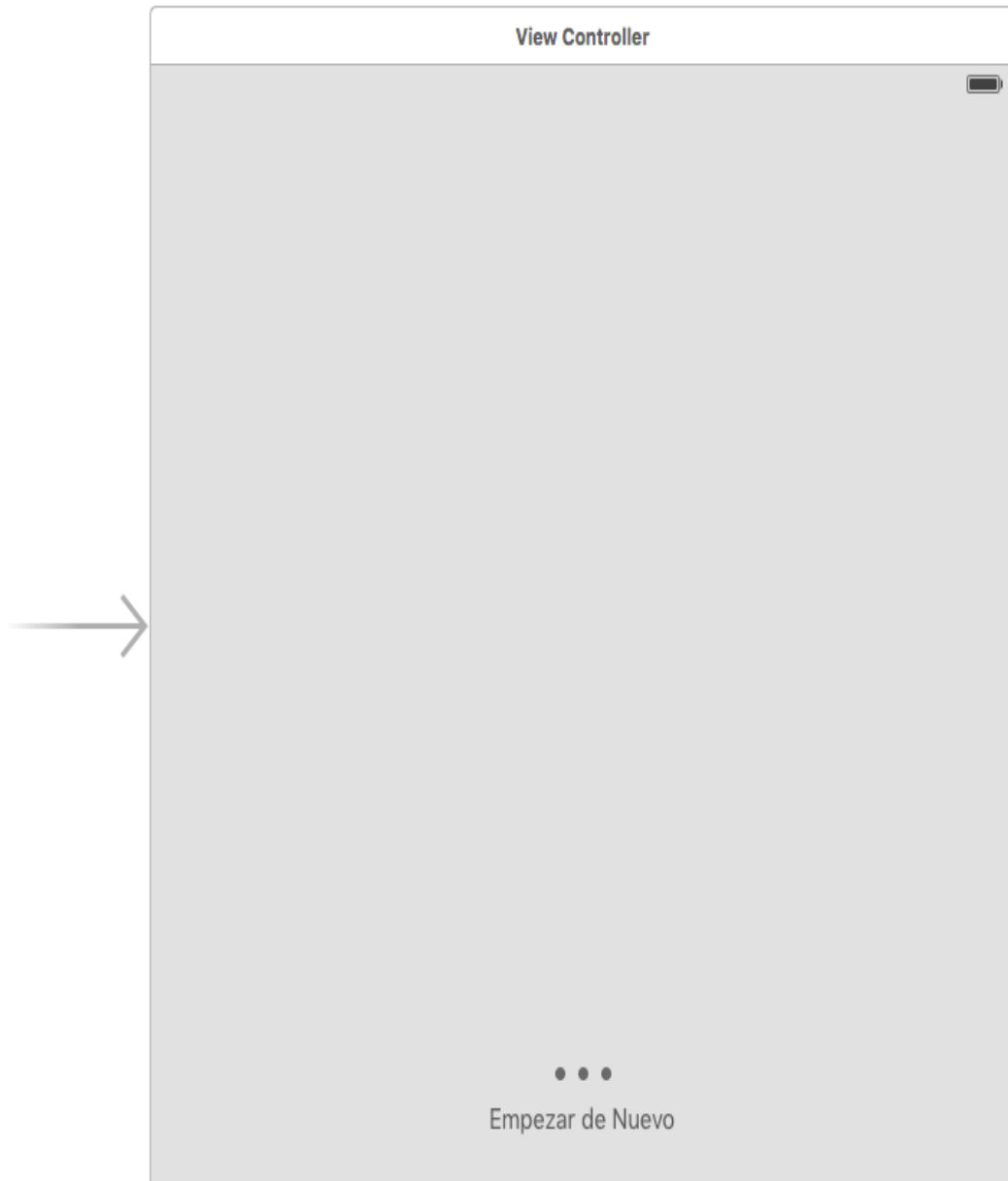
class PageContentViewController: UIViewController {
    //se crean las conexiones de los outlets
    @IBOutlet weak var imagenOutlet: UIImageView!
    @IBOutlet weak var labelOutlet: UILabel!

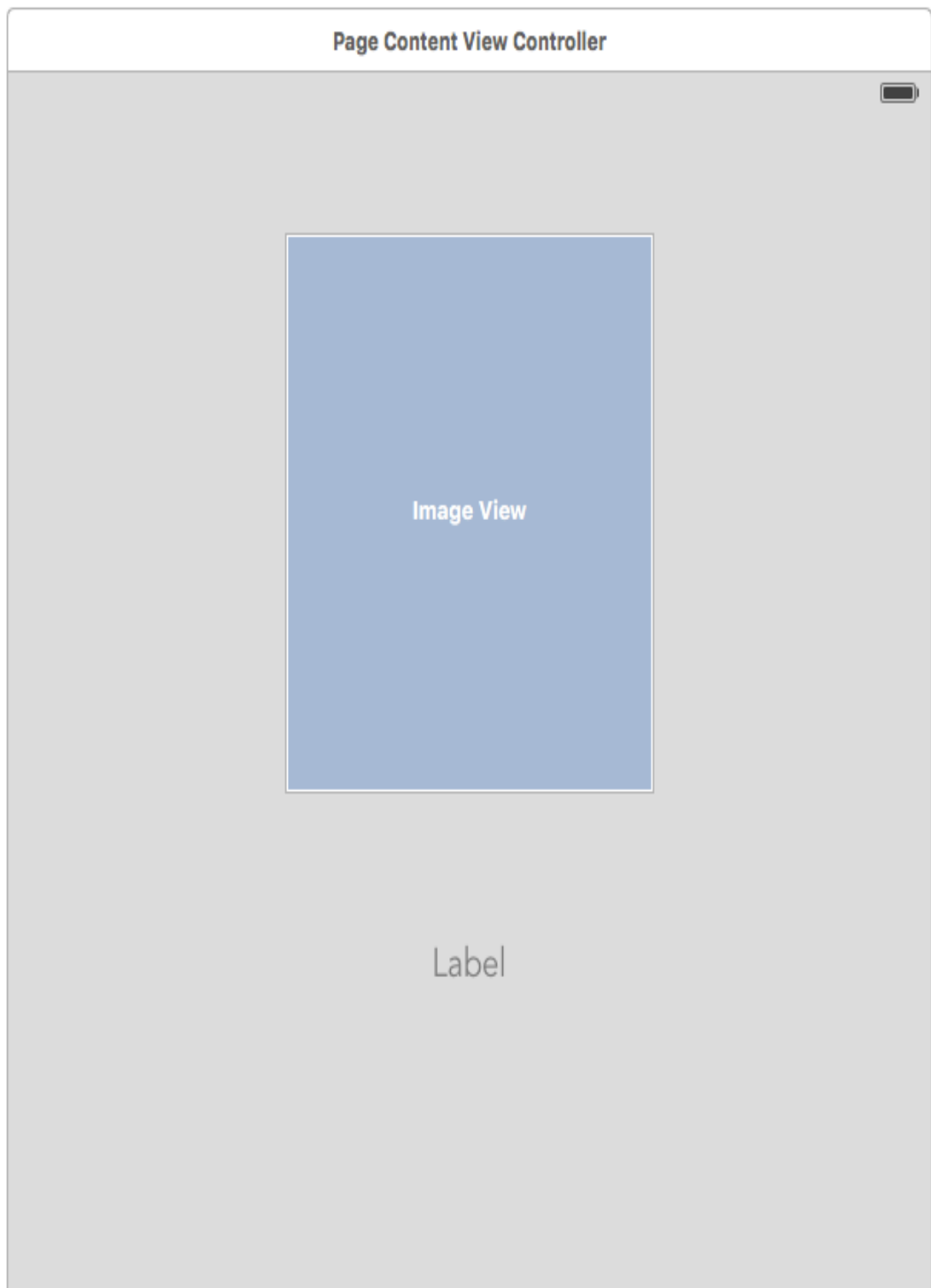
    var indexPagina: Int?
    var textoTitulo : String!
    var nombreImagen : String!
    //se setean los outlets con los valores creados en el el view
    controller
    override func viewDidLoad() {
        super.viewDidLoad()
        self.imagenOutlet.image = UIImage(named: nombreImagen)
        self.labelOutlet.text = self.textoTitulo
        self.labelOutlet.alpha = 0.1
        UIView.animateWithDuration(1.0) { () -> Void in
            self.labelOutlet.alpha=1.0
        }
    }
}
```

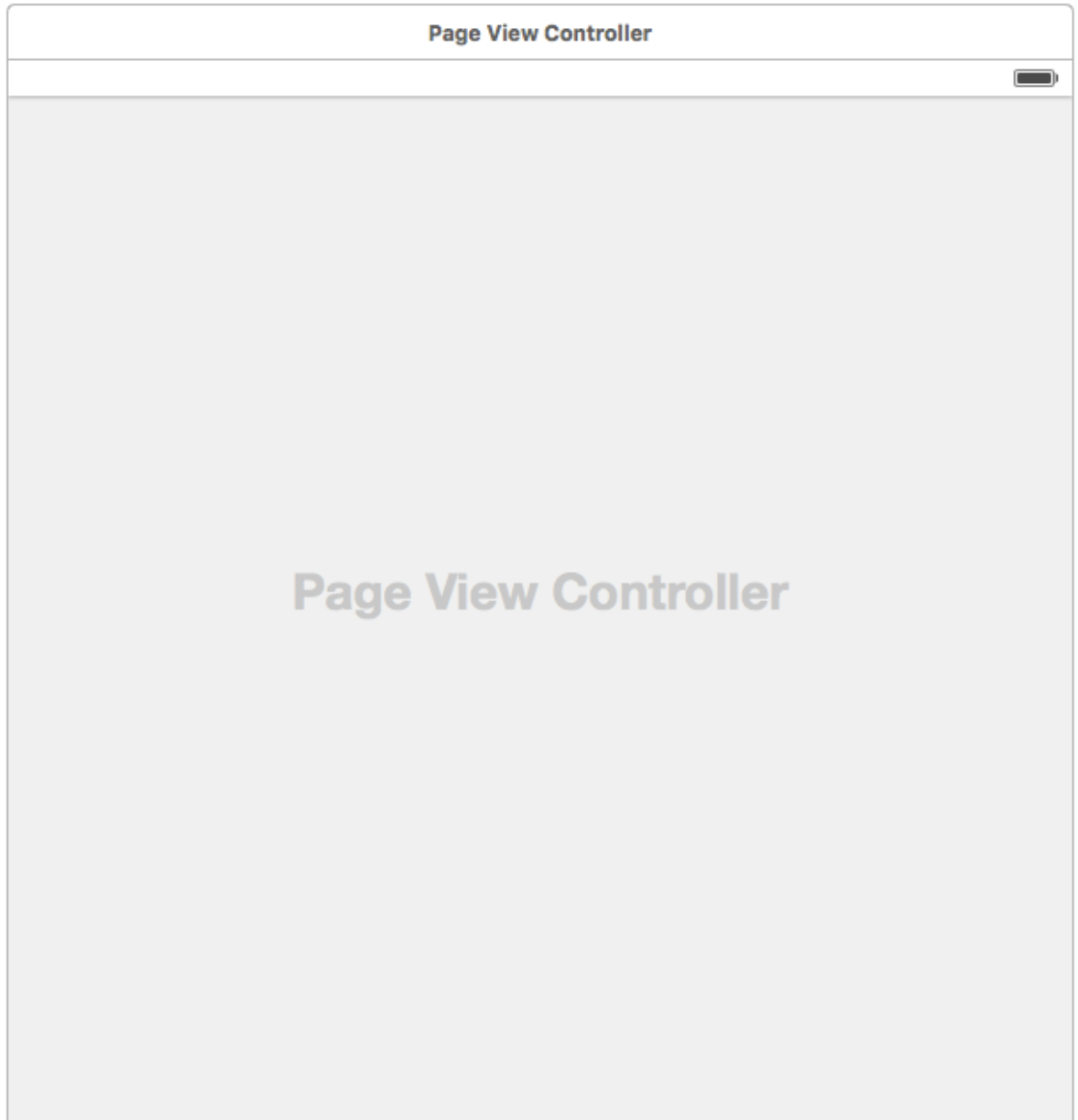
ANEJO J: INTERFACE CONTROLLER

*ANEXO J*

ANEKO K: MAINSTORYBOARD

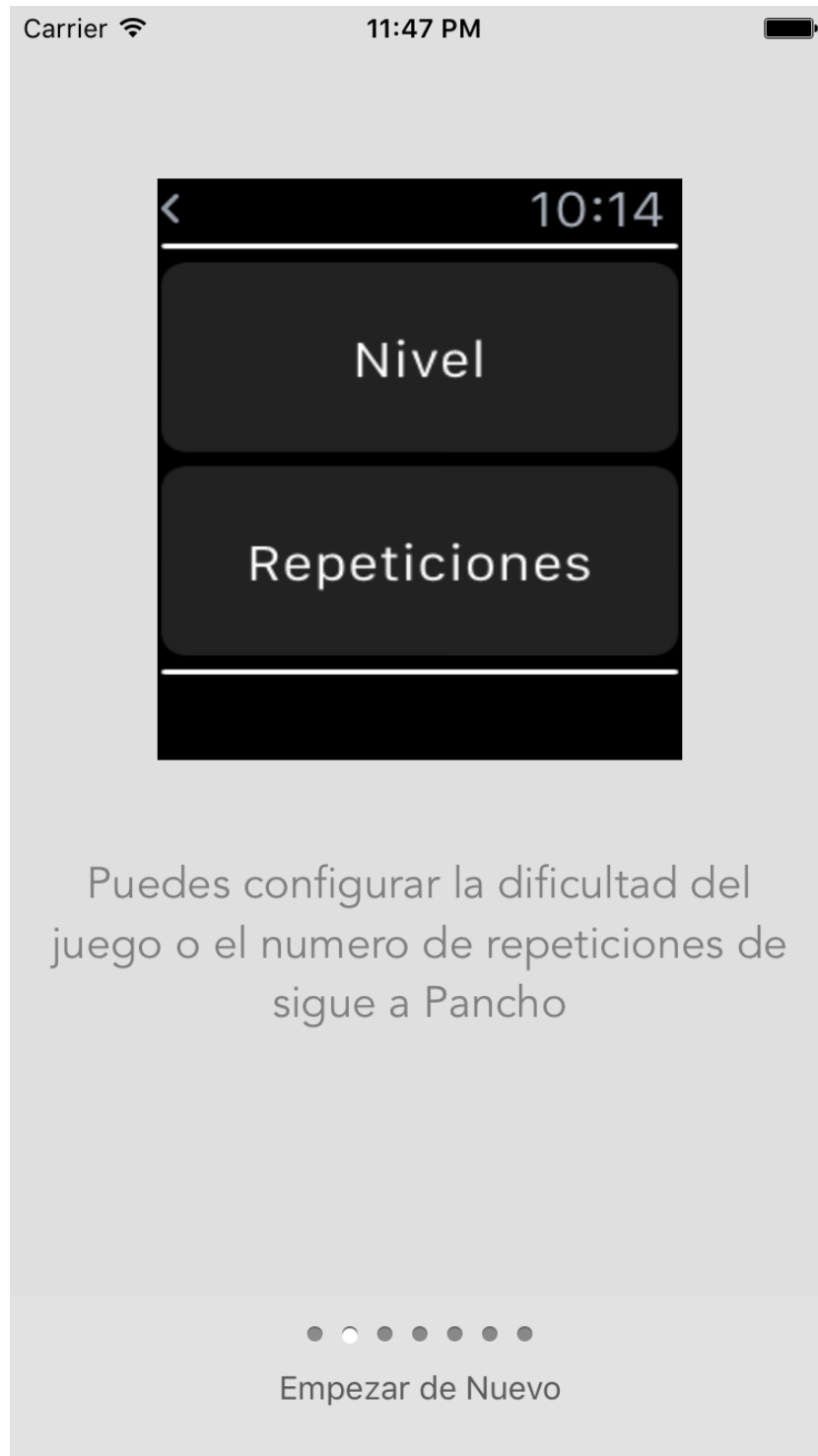


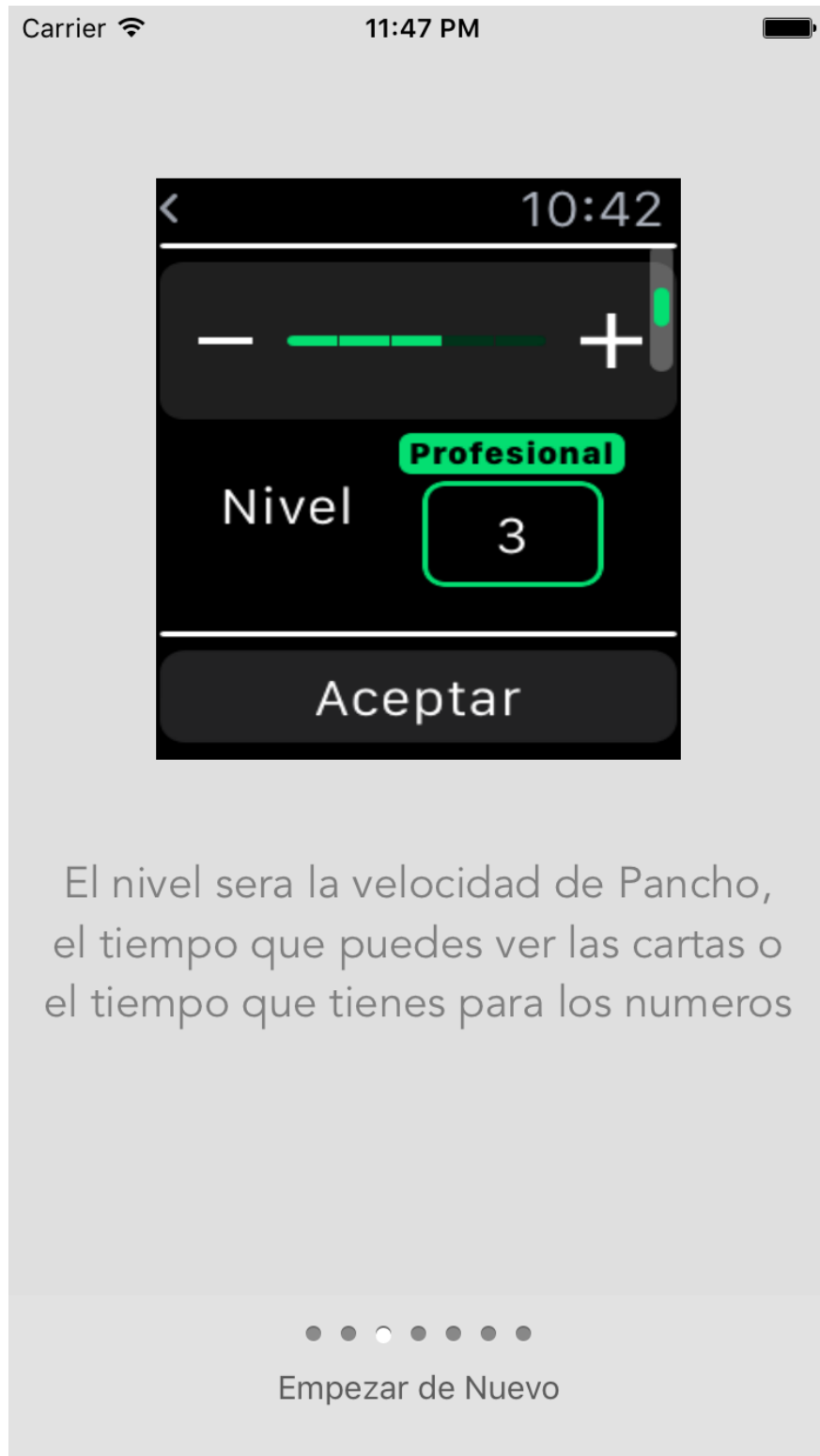




ANEXO L: MANUAL DE USUARIO CREADO PARA EL IPHONE







El nivel sera la velocidad de Pancho,
el tiempo que puedes ver las cartas o
el tiempo que tienes para los numeros





Carrier 📶 11:47 PM 🔋

< 10:14

20	9	78
77	11	38
73	52	89

AHORA 0:00:23

Aplasta los numeros de mayor a menor antes de que se acabe el tiempo. Ahora abre el juego en tu Apple Watch

● ● ● ● ● ● ●

Empezar de Nuevo