VERSIÓN 1.0.0

Desarrollo de una aplicación para dispositivos móviles con Realidad Aumenta para el apoyo a niños para Operaciones con Sumas y Restas

PRESENTADOR: JESUS ANTONIO CHAVEZ SAUCEDO

UNAM

PROYECTO FINAL IOSLAB

PROYECTO

Este proyecto explica el diseño e implementación de una aplicación para dispositivos móviles para el apoyo a niños entre 6 – 10 a través de un videojuego con Realidad Aumentada a para el reforzamiento de operaciones matemáticas simples como lo son la suma y la resta, ya que al reforzar estas las siguientes operaciones básicas les serán mucho más fáciles.

La Aplicación móvil presentara operaciones con sumas 0 restas en el aire con la Realidad Aumentada, en la pantalla aparecerán 2 botones los cuales de forma aleatoria aparecerán las posibles respuestas, al presionar las mismas saldrán disparadas, al hacer contacto se verifica si la respuesta con la que se colisiono es la correcta se eliminara la operación y se generara otra de forma aleatoria al igual las respuestas cambiaran, al ir contestando mas de forma correcta se acumulan los puntos y cada 5 puntos se subirá de nivel por lo que las operaciones poco a poco se irán complicando; todo esto debe ser llevado a cabo en 30 segundos.

OBJETIVOS

- El objetivo del presente proyecto es ayudar a los niños de entre 6 – 10 años a reforzar las operaciones matemáticas como lo son las Sumas y las Restas con la **Realidad Aumentada**

PROYECTO

REALIDAD AUMENTADA

La realidad aumentada es una tecnología que complementa el mundo real con el mundo digital. Superpone imágenes generadas por ordenadores, smartphones, tabletas o visores especiales a lo que sucede en tiempo real, de modo que el usuario tenga una mejor percepción de la realidad.

La realidad aumentada no solo ofrece formas divertidas e interactivas para que el usuario común y corriente aprenda, experimente e imagine cosas nuevas, sino que también tiene aplicación en numerosos campos, como se verá a continuación

Un práctico ejemplo de realidad aumentada es Pokémon GO. Este popular juego utiliza esta tecnología para colocar a las criaturas virtuales y los objetos que permiten capturarlos en lugares reales. Las imágenes digitales de estos elementos virtuales no son más que píxeles en la pantalla del usuario, pero la tecnología de realidad aumentada los hace parecer tangibles.

La realidad aumentada vs. la realidad virtual

Aunque aparentemente son similares, la realidad aumentada y la realidad virtual son tecnologías muy distintas.

La realidad virtual es una tecnología informática que replica artificialmente un entorno real (o imaginario) y le proporciona al usuario (apelando principalmente a su visión y audición) la sensación de estar verdaderamente dentro de él. La realidad virtual puede ser experimentada a través del uso de gafas especiales como el Oculus Rift.

ARKit by APPLE

La primera versión de ARKit sincronizaba el mundo real con el virtual y permitía la gestión de la luz. Y era capaz de detectar planos horizontales (pero no verticales). Además, los planos detectados eran regulares: debían ser planos rectangulares.

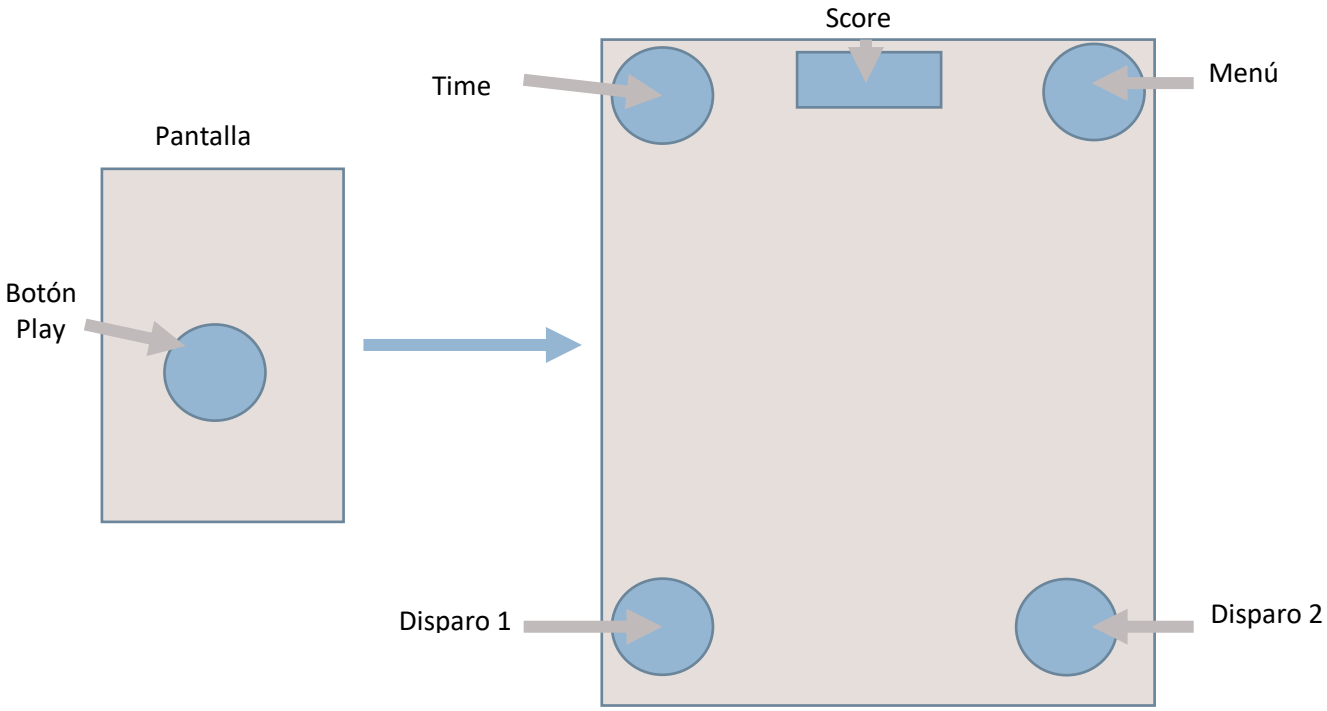
En junio de 2017, Apple sorprendía a los desarrolladores con el anuncio de su propia librería de realidad aumentada y que llegaría en septiembre con iOS 11: ARKit. La forma en que un mundo generado por nuestro dispositivo Apple se colocaba sobre la imagen de la cámara y parecía que estaba allí realmente

En marzo de 2018, Apple lanzó iOS 11.3 e incorporó ARKit 1.5 que permitía detectar planos verticales y además superficies irregulares por lo que ya no era necesario que fueran planos rectangulares E incorporó la detección de imágenes 2D uniendo su tecnología de visión computarizada.

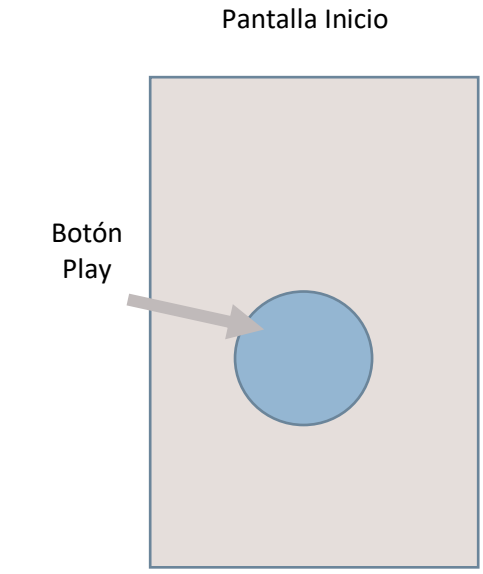
En junio de 2018, Apple ha presentado ARKit 2.0 junto con iOS 12 con nuevas funciones: detección y seguimiento de imágenes 2D, experiencias compartidas, grabación de mapas de referencia y reflectividad. Además de mejorar significativamente los algoritmos para que la precisión en la sincronía entre mundo real y virtual sea aún mejor.

COMO SE DESARROLLO

Diagrama general de la aplicación.

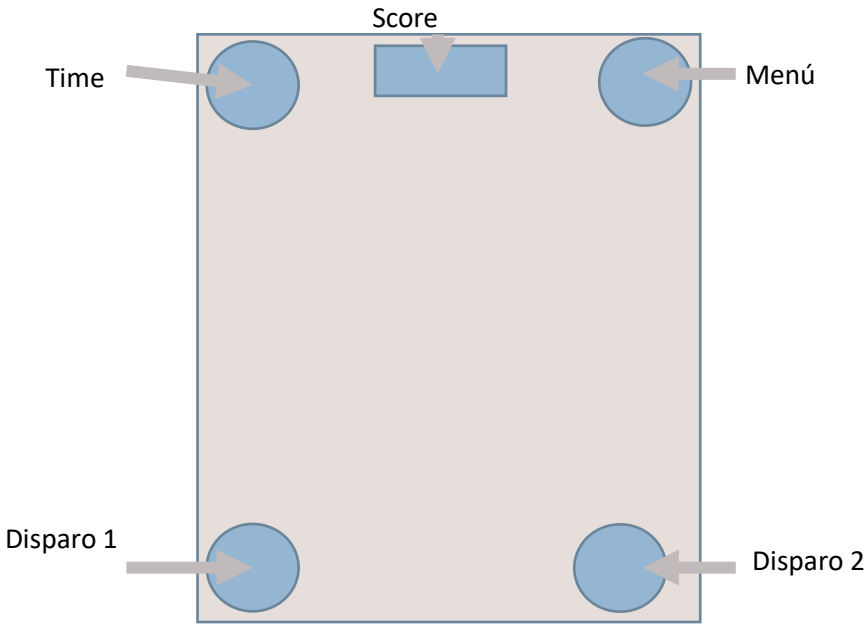


Pantalla de Inicio



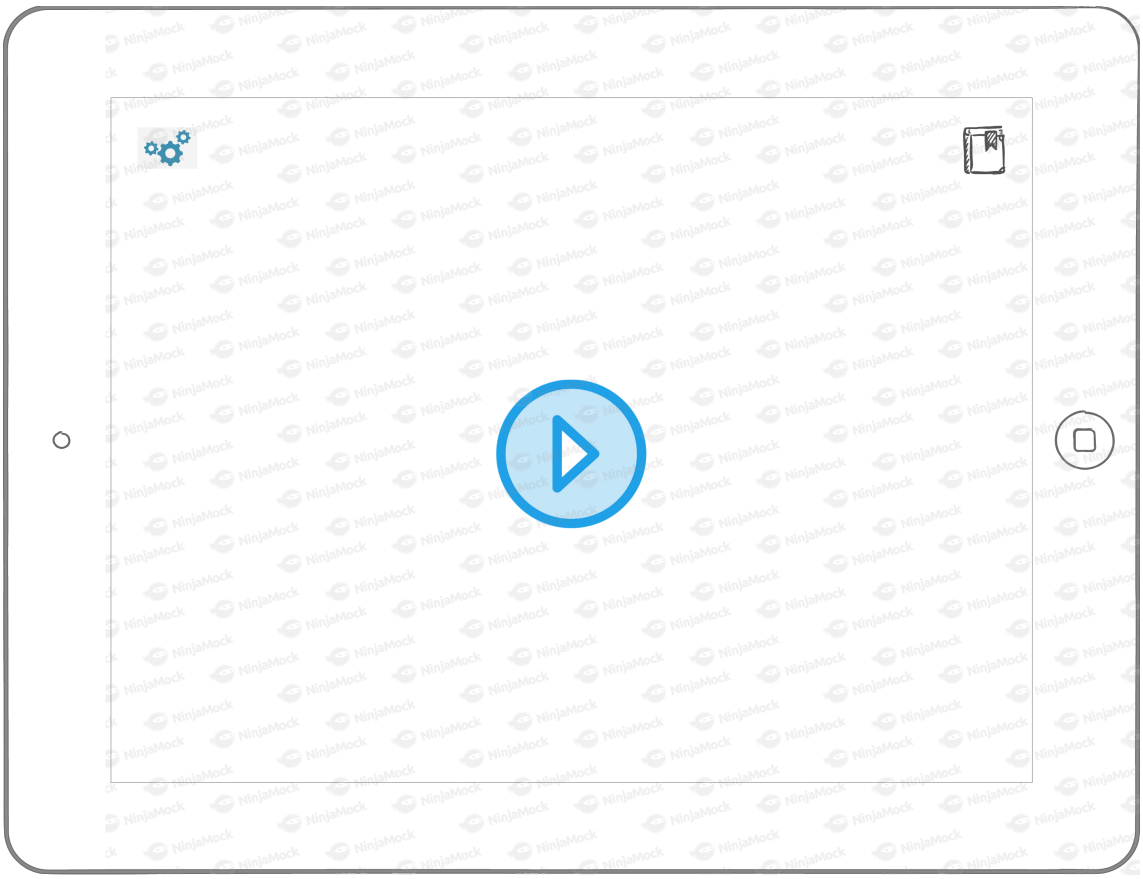
Consiste una pantalla donde aparecerá un botón de Play al inicio y el titulo del Juego, si el usuario regresa después de haber jugado el botón cambia a reintentar, mostrará el mejor y el ultimo score logrado.

Pantalla Principal del Videojuego.



En esta pantalla se muestra le videojuego.

MOCKUPS App



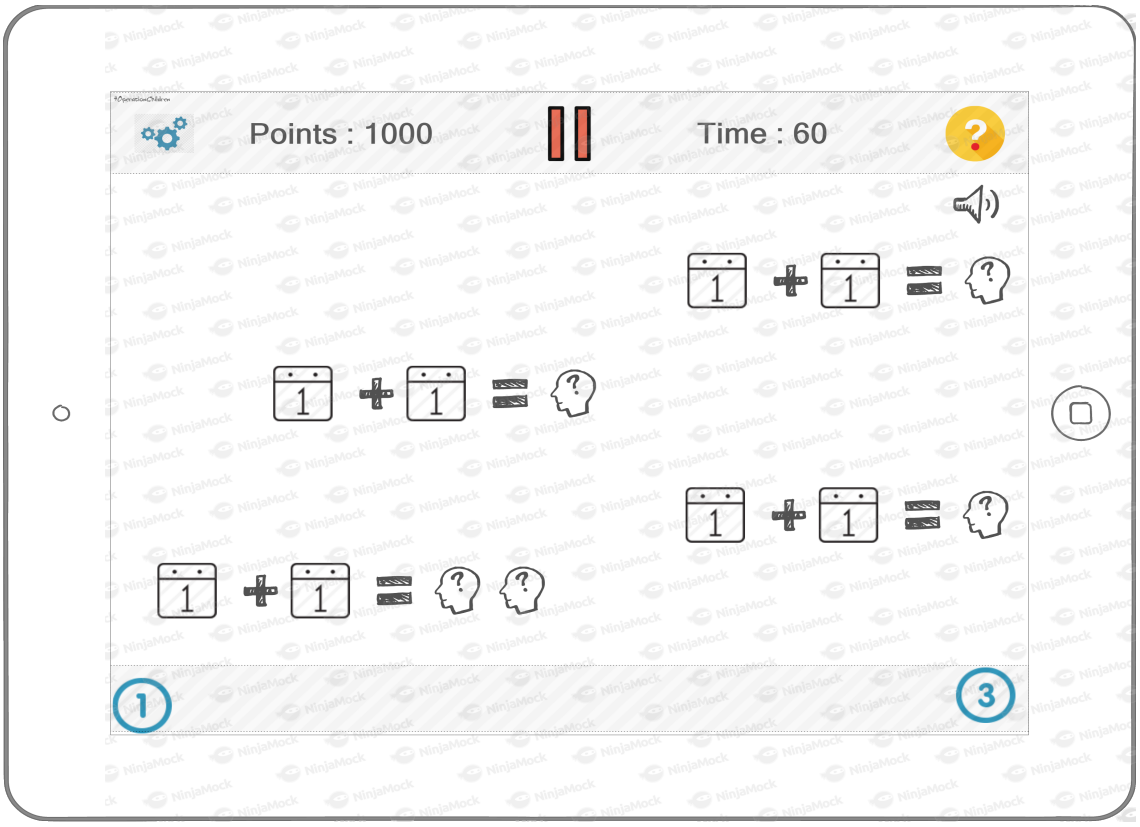
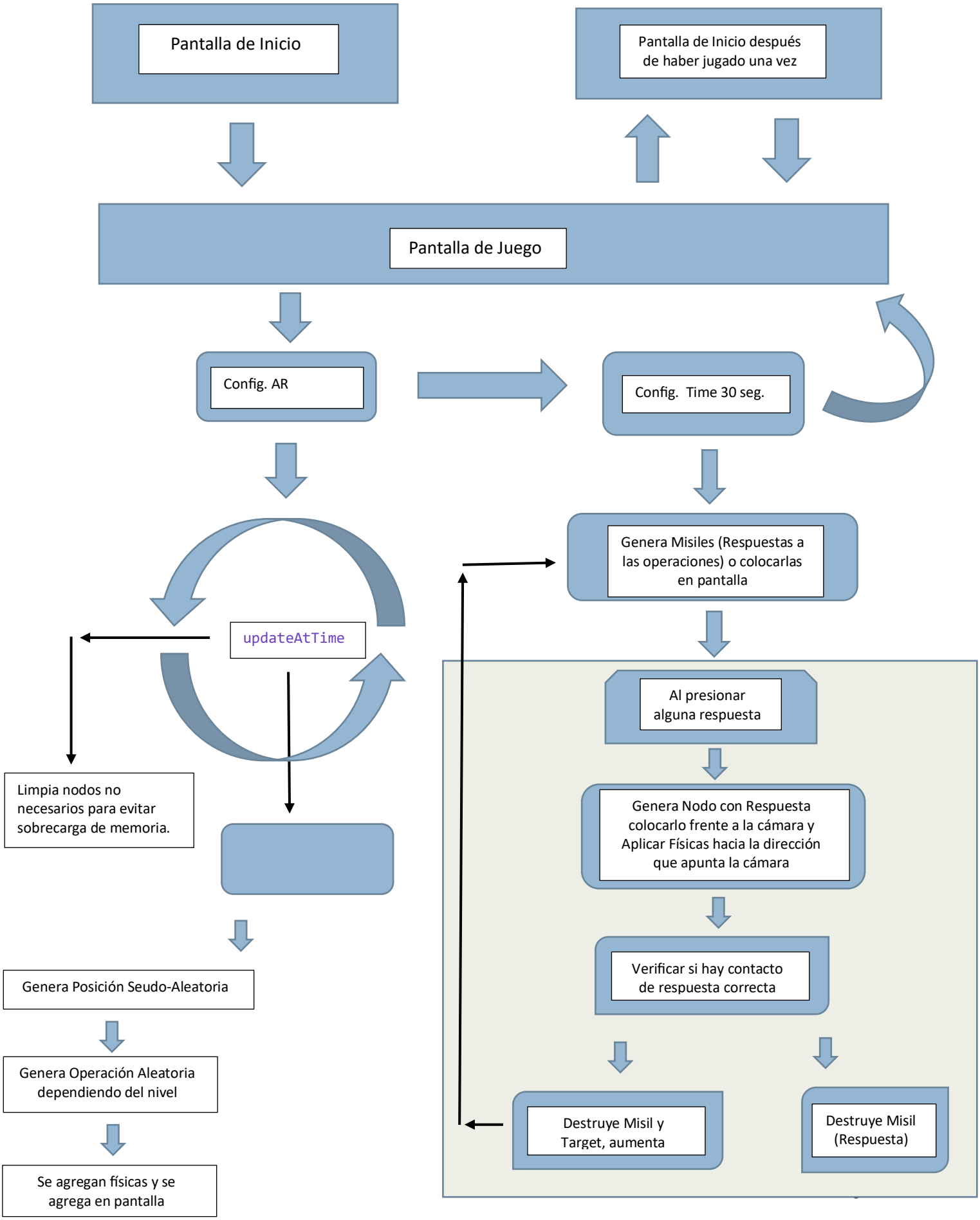


Diagrama General



Como funciona el videojuego

Toda la vista se basa en la clase `ARSCNView` que es la que sirve para mostrar las escenas 3D con `SceneKit`.

Al controlador le agregamos el delegado `ARSCNViewDelegate` , `SCNSceneRendererDelegate` y `SCNPhysicsContactDelegate`, que nos permitirán trabajar mas a detalle con ARKit y las físicas, al controlador le indicamos que él será el delegado de la Escena y las físicas.

```
self.sceneView.delegate = self
self.sceneView.scene.physicsWorld.contactDelegate = self
```

Para comenzar la RA se crea una configuración de lo que vamos a realizar, que para nuestro caso es el tracking del mundo 3D que para ello se crea el objeto `ARWorldTrackingConfiguration` y se pasa como parámetro a la sesión de la vista 3D (también existe configuración para detectar planos)

Con esto se declara que vamos a hacer tracking del mundo y arrancamos la realidad aumentada.

```
        let configuration = ARWorldTrackingConfiguration()
        self.sceneView.session.run(configuration)
```

-Targets (Operaciones a resolver)

Al iniciar la Realidad Aumentada, entra en juego las funciones de los protocolos.

El que en este momento nos interesa es

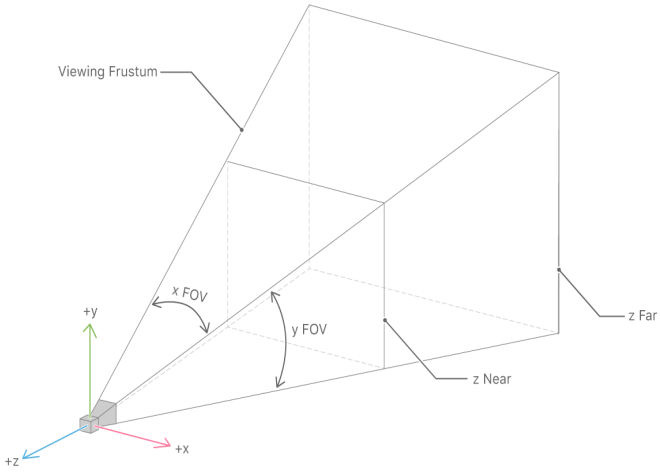
```
func renderer(_ renderer: SCNSceneRenderer, updateAtTime time: TimeInterval)
```

que se encarga de ir capturando lo que sucede cada cierto tiempo, en esta función agregaremos el Spawn de los Targets(Objetivos u Operaciones a Resolver en el aire), limpiamos la escena de los nodos que ya no son útiles.

```
func renderer(_ renderer: SCNSceneRenderer, updateAtTime time: TimeInterval) {
        if gameManager.state == .Playing {
                if time > gameManager.spawnTime {
                        if gameManager.countSpawner < gameManager.countSpawnerMax{
                                spawnTarget()
                                gameManager.spawnTime = time + TimeInterval(Constant.spawnTime)
                        }
                }
                cleanScene()
        }
}
```

Para agregar las operaciones (Targets) es necesario tomar varias cosas en cuenta

Es muy importante conocer la dirección de los ejes en 3D.



Para colocar todo de la forma correcta se debe conocer el como se organiza los objetos dentro del ARKit, los objetos que se organizaran en jerarquía a través de objetos `SCNNode`, se parte del `rootNode` que es el principal que se encuentra en la escena y de ahí se van agregando como hijos en jerarquía.

```
sceneView.scene.rootNode.addChildNode
```

El esquema en general de la jerarquía del juego quedaría así:

sceneView.scene.rootNode

rootNode

SCNode - Target

| |

| | -> SCNPhysicsBody

|

| -> SCNode [**Numero 1**]

| |

| | -> SCNPhysicsBody

| -> SCNode [**Tipo Operación**]

| |

| | -> SCNPhysicsBody

| -> SCNode [**Numero 2**]

| |

| | -> SCNPhysicsBody

| -> SCNode [**Símbolo "="**]

| |

| | -> SCNPhysicsBody

| -> SCNode [**Símbolo "?"**]

| |

| | -> SCNPhysicsBody

```
|
|
| -> SCNode [ Disparo 1 ]
        |
        | -> SCNPhysicsBody
```

Al generar el Target se le agrega físicas y se coloca en pantalla de forma pseudoaleatoria.

-Ciclo (Tiempo - Disparo)

Una vez que se inicio la Realidad Aumentada, comienza el tiempo de juego que son 30 segundos, en ese momento se generan las posibles respuestas a disparar desde 2 botones,

Al presionar alguno d ellos botones se obtiene la dirección y posición de la cámara, esa información la obtenemos de la siguiente forma:

```
func cameraVector() -> (SCNVector3, SCNVector3) { //(direction, position)
        if let frame = self.sceneView.session.currentFrame {
                let mat = SCNMatrix4(frame.camera.transform)
                let dir = SCNVector3(-1 * mat.m31, -1 * mat.m32, -1 * mat.m33)
                let pos = SCNVector3(mat.m41, mat.m42, mat.m43)
                return (dir, pos)
        }
        return (SCNVector3(0, 0, -1), SCNVector3(0, 0, -0.2))
}
```

ARKit nos proporciona toda la información de la cámara dentro del frame que está en la sesión, pero no la da en un formato 4x4 por lo que hay que extraer la info.

```
SCNMatrix4(frame.camera.transform)
```

Los valores que nos interesan son los que se encuentra en la columna 3 y las filas 1,2 y 3 según la **documentación de Apple** es donde nos proporciona el X, Y y Z que necesitamos, con esto obtenemos la orientación de la cámara en el mundo 3D

```
SCNVector3(-1 * mat.m31, -1 * mat.m32, -1 * mat.m33)
```

La localización de la cámara en el mundo 3D se obtiene de la matriz 4X4 en la columna 4 y las filas 1,2,3

```
SCNVector3(mat.m41, mat.m42, mat.m43)
```

Con esto tendríamos el desde y hacia donde se va a disparar.

Se genera el Nodo con la respuesta, se coloca frente a la cámara y se le aplica una fuerza hacia la dirección donde apunta la cámara.

Si la colisión es correcta, se verifica si la respuesta es correcta a la operación, si es así ambos se destruyen el target, el misil y se aumenta el Score; de lo contrario se destruye solo el misil.

Después genera nuevas respuestas para disparar.

Esto se debe llevar acabo en un lapso de 30 segundos.

==========PROBLEMAS DURATE EL PROYECTO==========

El más grande problema fue la forma en cómo se generaban los misiles.

Para generar los objetos que estarán dentro de los SCNode's se tenían 2 opciones.

1a) Se generaban modelos 3D de los objetos que se iban a necesitar (Números, Signos) lo cual por falta de conocimiento en modelado 3D y el tiempo restante se descartó.

2a) ARKit nos proporciona un objeto SCNText que sirve para mostrar textos lo cual quedaba perfecto para lo que se necesitaba.

Las primeras versiones se hacían a través de SCNText por la facilidad de uso.

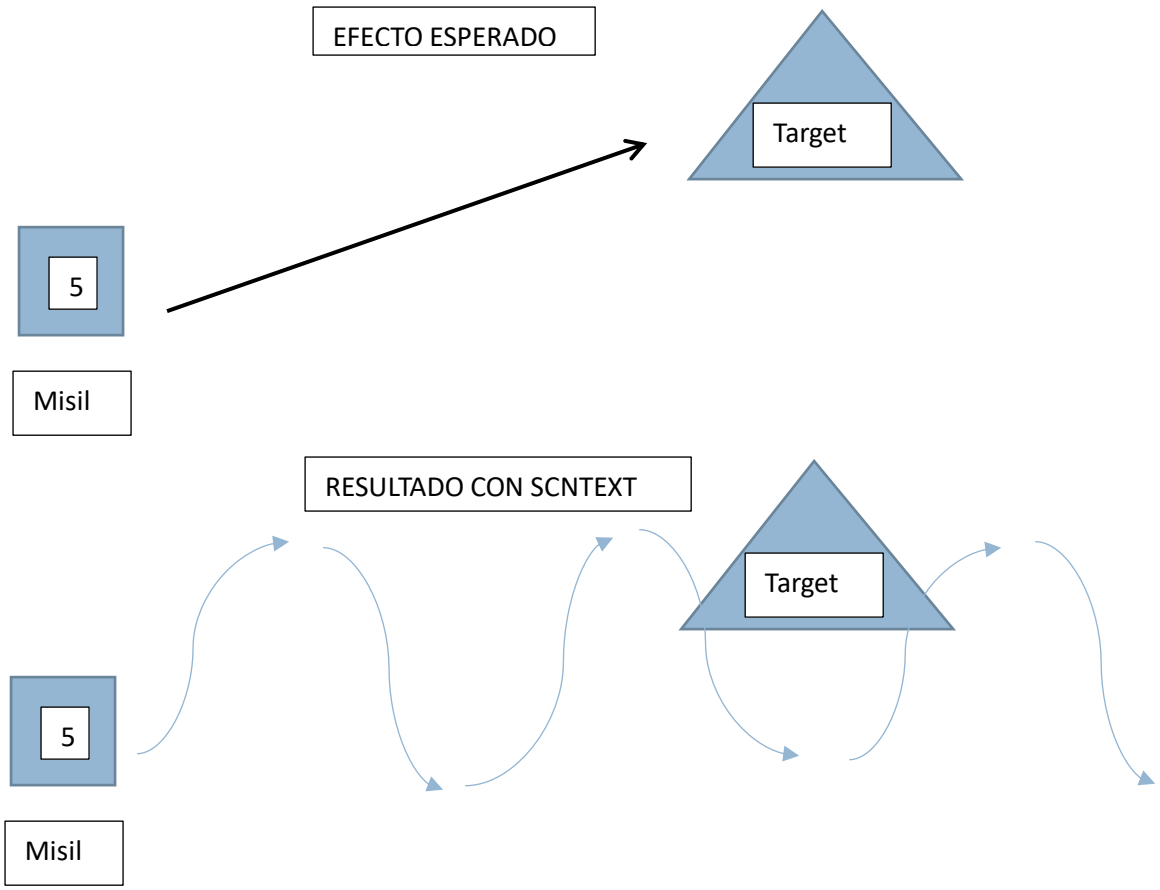
Se genera el objeto SCNText se configura la fuente, el material y se agrega aun objeto SCNNode, se configura la física y listo.

```
let geoText = SCNText(string: number.raw, extrusionDepth: 0.1)
geoText.font = UIFont(name: "Arial", size: 0.6)
geoText.firstMaterial!.diffuse.contents = UIColor.red
let node = SCNNode(geometry: geoText)
```

El problema es, al momento de utilizar para un misil, se le aplica una fuerza hacia una dirección, como se permite en un objeto 3d.

```
node.physicsBody?.applyForce(direction,  asImpulse: true)
```

Lo que sucede es que al aplicar esta fuerza en lugar de ir el objeto de forma recta comienza a hacer efectos diferentes a lo esperado, por lo que se dificulta el disparo.



Este efecto no permitía el disparo correcto, por lo que se tuvo que realizar el cambio a generar los modelos 3D y con este cambio el resultado ya fue el esperado.

También al haber creado los modelos 3D los objetos del target al igual se cambiaron por estética.

==========DISPOSITIVOS EN LOS QUE SE PROBO==========

Iphone 7

Ipad 6ª Generación.