UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Análisis y comparación de las herramientas de interfaz de usuario en Unity

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Rafael Faulí Soria

Tutor: Alberto Bonastre Pina
Rubén Picó Ruiz

2014/2015

Resumen

Este trabajo fin de grado presenta un estudio comparativo de dos herramientas de interfaz gráficas disponibles para el motor UNITY. Por un lado, la experimentada Next-Gen UI (NGUI) frente a la recientemente integrada interfaz nativa de Unity (versión 4.6 y siguientes). Se plantea un estudio detallado de ambas herramientas para a continuación proponer un conjunto de parámetros que permitan esta comparativa. La evaluación de estos parámetros permiten poner de manifiesto los puntos fuertes y las carencias de ambas opciones. Este trabajo se enmarca en el desarrollo de una aplicación real diseñada por la empresa BraveZebra para la Liga de Fútbol Profesional (LFP), denominada LaLiga Fantasy.

Palabras clave: Unity, Interfaz Gráfica de Usuario, Aplicaciones para Dispositivos Móviles, NGUI, Estudio comparativo.

Abstract

In this final grade work a comparative study of two UI (User Interface) tools available for UNITY is presented. Experimented Next-gen UI (NGUI) is compared with the new UI integrated in Unity from version 4.6. A detailed study of both tools is presented. Later, a set of significant parameters is stablished as a base of the comparison. The evaluation of these parameters highlights the advantages and disadvantages of both options. This work has been performed in a start-up called BraveZebra, in the development of a mobile application called LaLiga Fantasy, for the Liga de Fútbol Profesional (LFP).

Key words: Unity, Graphical User Interface, Applications for Mobile Devices, NGUI, Comparative study.

Tabla de contenidos

Contenido

1. Introducción 5
1.1. Motivación 5
1.2. Objetivos 7
2. El motor gráfico Unity 8
2.1. Componentes incorporados 15
2.2. Técnica 9-slice y Sprite Editor 19
2.3. Atlas y Sprite Packer 21
3. El *plugin* NGUI 22
3.1. Componentes incorporados 23
3.2. Atlas Maker y Font Maker 28
4. Comparativa de herramientas 29
4.1. Parámetros de comparación 29
4.2. Estudio 29
4.3. Conclusiones 36
5. Descripción del proyecto 37
5.1. Arquitectura 37
5.2. Método de desarrollo 37
5.3. Plataforma objetivo 38
5.4. Lenguajes 39
5.5. Herramientas empleadas 39
6. Ampliaciones de Unity 41
6.1. Análisis 41
6.2. Diseño 41
6.3. Implementación 42
7. Conclusiones y trabajo futuro 43
8. Términos y vocabulario 44
9. Bibliografía 45

1. Introducción

El proyecto que da lugar a éste trabajo de final de grado se desarrolla en una *StartUp* llamada BraveZebra situada en el edificio 9B de la Universidad Politécnica de Valencia. Una empresa dedicada al *outsourcing* de videojuegos. Una emprendedora con dos años de crecimiento tanto económico como en tamaño habiendo conseguido contratos con una gran variedad de empresarios tanto internacionales como nacionales. Múltiples nacionalidades han trabajado con BraveZebra, desde empresarios americanos hasta chinos, pasando por holandeses, canadienses, y también españoles.

El cliente para el cual se desarrolla el proyecto es un cliente considerado de gran importancia dentro de la empresa, la Liga de Fútbol Profesional. El cual, no es el primer proyecto que confía a la empresa, sino que éste es el segundo que contrata con la empresa emprendedora.

La petición de La Liga (como se hace llamar actualmente la Liga de Fútbol Profesional) ha sido un juego en dos dimensiones disponible para plataformas móviles, como *Android* e *iOS,* y navegadores web, en el que puedes tomar un rol de manager de fútbol y gestionar tu equipo pudiendo hacer ofertas por jugadores en el mercado o poner a la venta tus propio jugadores. El nombre de la aplicación final es *La liga Fantasy* y quiere hacer competencia a otros juegos de temática similar como Comunio, Fútbol mánager o Golden Manager pero siendo éste un juego oficial de La Liga.

Una vez iniciado el proyecto, se escogió la nueva interfaz de Unity para llevarlo a cabo. No obstante, teniendo otras herramientas en el mercado más veteranas, se precisó un estudio comparativo de dos de ellas para conocer sus características y poder escoger en proyectos futuros la mejor opción.

1.1. Motivación

El mercado de los dispositivos móviles, tanto *tablets* como *smartphones* está en claro crecimiento. Un análisis de la firma Newzoo estimó que las ganancias globales generadas por los juegos para teléfonos inteligentes y tablets serán de unos 25 mil millones de dólares este año, un 42 por ciento superior a la de 2013, mientras que en 2015 superarán los 30 mil millones, explicó la web de noticias *Télam* sobre el estudio realizado por *Newzoo,* una empresa cuya especialización es la investigación de mercados[1][2].

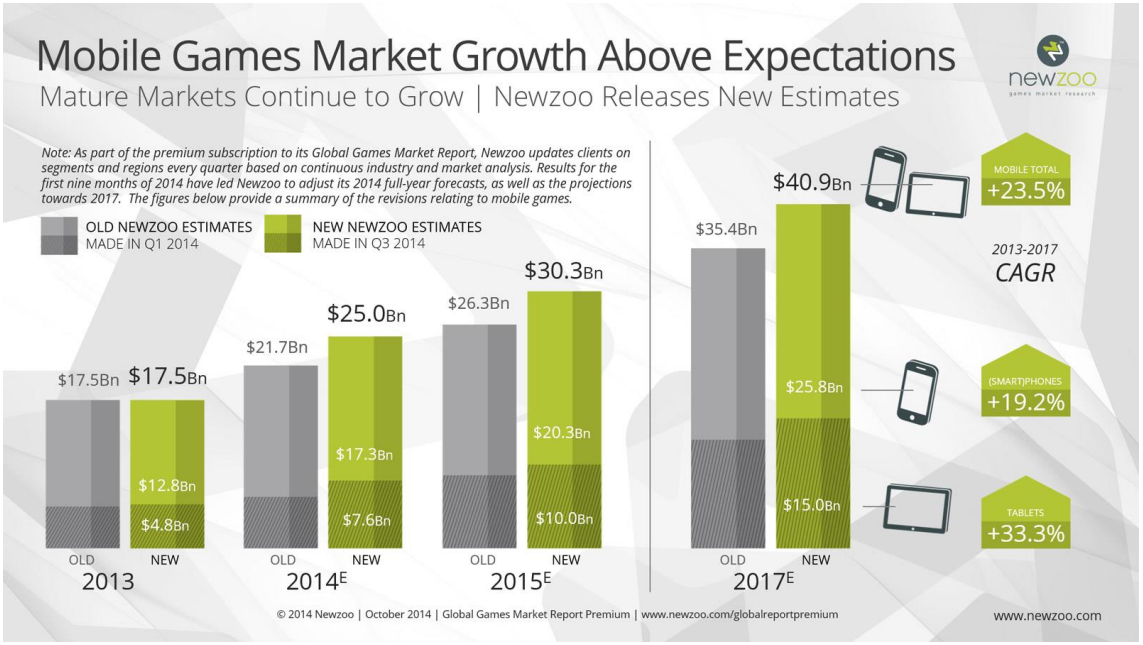


Figura 1. Gráfico de estimaciones de aumento del mercado de dispositivos móviles. Newzoo.

Según el estudio de Juniper Research llamado *'Mobile Entertainment Strategies: Markets, Opportunities & Forecasts 2011-2015'*, el mercado del ocio móvil generó 33,000 millones de dólares el año pasado, cifra que ascenderá a 54,000 millones en 2015 gracias al auge de los smartphones, las descargas y las aplicaciones de consumo, redacta Francisco Carrasco a través de la web[3].

Si bien es cierto que diferentes analistas pronostican distintas cantidades en el crecimiento de estas cifras, el resultado obtenido es el mismo, un rápido y gran crecimiento en el mercado de ocio en los dispositivos móviles. Visualizando estas cifras, muchos estudios, empresas y organizaciones han optado por aprovechar la oportunidad y desarrollar aplicaciones para este mercado. Una de estas empresas ha sido La Liga, optando por unirse a este movimiento y crear sus propias aplicaciones. Motivada por las perspectivas de ganancias y las grandes ventajas que ofrecen las publicadoras en cuanto a tiempos de publicación y distribución del producto se ha desarrollado este proyecto.

Por otro lado, la falta de recursos en los dispositivos móviles hace necesario que las aplicaciones desarrolladas para estos optimicen el uso de los componentes del dispositivo. La falta de memoria RAM hace que la aplicación no pueda tener mucha carga en ejecución, la poca memoria de disco obliga a que las compilaciones no puedan ser muy pesadas o los chips gráficos no muy potentes precisando gráficos no muy pesados. Además de esto, el desarrollo para dispositivos móviles requiere hacerlo multiplataforma además de que en cada plataforma existen dispositivos con características muy dispersas. Para ello, se utilizan las utilidades de optimización incorporadas en estas herramientas. Su uso imprescindible si se desea obtener un buen resultado para estas plataformas.

1.2. Objetivos

- El objetivo de este trabajo es hacer un estudio comparativo entre las herramientas NGUI y la nueva interfaz de Unity con el fin de poder aclarar ventajas y desventajas de ambas

Para ello es necesario cumplir los siguientes objetivos parciales:

- Entender el funcionamiento básico y componentes de la nueva interfaz de Unity.
- Entender el funcionamiento básico y componentes del *plugin* NGUI.
- Establecer parámetros de comparación entre los diferentes interfaces gráficos
- Aplicar los parámetros para obtener una evaluación objetiva de los mismos
- Conclusiones

2. El motor gráfico Unity

El motor gráfico Unity 3D lleva muchos años estando en boca de muchos programadores de videojuegos. Esto se debe principalmente a que fue uno de los motores de videojuegos en 3D que mejor calidad ofrecían siendo gratuitos para el público que deseara iniciarse o perfeccionar sus técnicas en este ámbito. A esto se suma una gran comunidad dando apoyo a los desarrolladores que necesitaban ayuda. No solamente otros usuarios, sino que los propios empleados de Unity prestan ayuda a todo el que la necesita en foros oficiales. Finalmente una gran documentación, tanto oficial como proporcionada por usuarios que comparten su experiencia y técnicas en la red, hacen del motor una excelente herramienta para diversos perfiles de usuario, tanto aquellos que se inician en el desarrollo de videojuegos, programadores que quieren mejorar sus habilidades o profesionales que deseen utilizar un motor dinámico y asequible. Una vez iniciado Unity nos topamos con su interfaz de usuario.

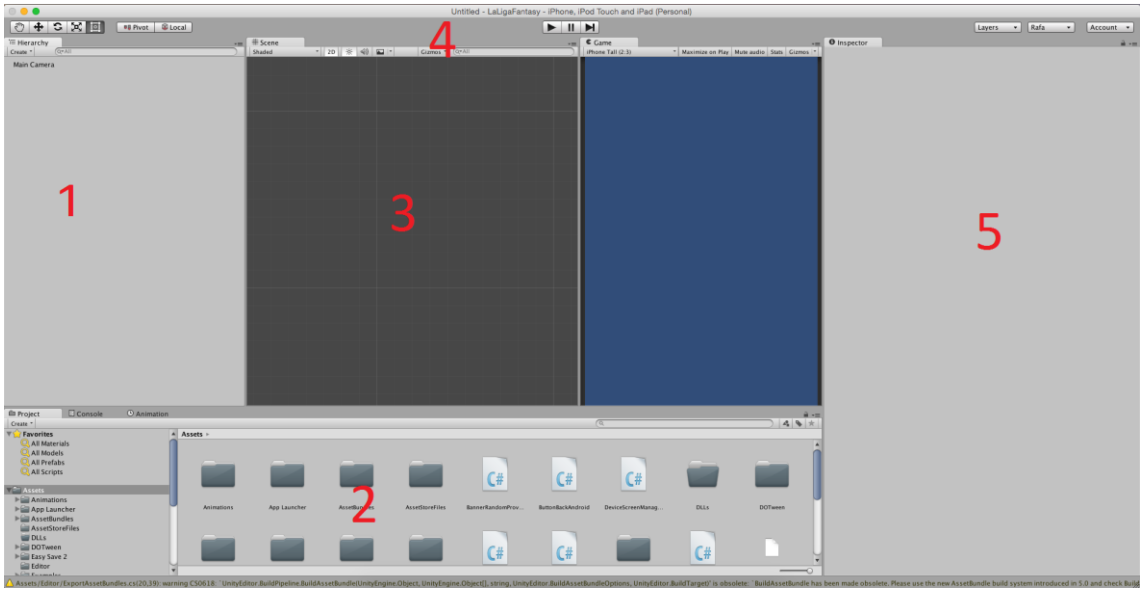


Figura 2. Ventana de Unity.

Podemos observar cinco separaciones claras en la figura 2. En primer lugar, podemos observar la ventana marcada con el uno y esta es la ventana de jerarquía. Aquí es donde estarán presentes todos los elementos en la escena de juego. En ella los elementos entablan una relación de padres e hijos para organizar la escena. En ella solo se albergan *GameObjects* (Ver glosario). Con el número dos, encontramos la venta de proyecto. Este apartado contiene todos los elementos o *assets* (Ver glosario) que tenga juego. El número tres es la vista de la escena, donde podemos movernos en tres dimensiones o dos según las dimensiones del juego, y la vista de juego, donde se mostrará el juego mientras se monta con la imagen final que tendrá. También es la vista que mostrará el juego mientras esté en ejecución en el editor. La barra de herramientas, que en la figura 2 se demarca con el número cuatro, contiene botones

que hacen más rápida la interacción con el usuario. Aquí podemos seleccionar el modo del cursor del ratón para la vista de escena, ejecutar, pausar o parar el juego y cambiar la disposición de las ventanas del editor entre otras cosas. Cabe decir, que estas ventanas pueden ser modificadas al gusto del usuario también mediante *drag and drop* (Ver glosario). Finalmente, tenemos el inspector marcado con el número cinco. Esta ventana es la encargada de ofrecer información detallada de todos los elementos seleccionables del proyecto.

Unity trabaja en escenas de juego. Cada escena de juego es un mundo donde se pueden añadir distintos componentes y objetos. Normalmente la comunidad utiliza cada escena para denominar cada uno de los niveles del juego, pero esta no es una regla que se deba cumplir al pie de la letra ya que no todos los juegos son iguales y cada uno necesitará de su propia lógica. Al crear una nueva escena, siempre se crea una cámara principal del juego en la jerarquía. Esta es la que se encargará de mostrar al usuario el entorno. Cada escena tiene su propia cámara que *renderizará* (Ver glosario) el mundo virtual para mostrarlo por pantalla al usuario. Hay que tener en cuenta que en el cambio entre escenas no persiste ningún objeto de la escena descargada.

En la ventana de jerarquía, Unity utiliza unos elementos denominados *GameObjects*. Cada uno de estos objetos puede contener diversos componentes para dotarlo de distintas funcionalidades. Desde físicas, navegación y gráficos hasta sonidos, iluminación y scripts propios con el código fuente de cada programador. Se podría decir que estos elementos son los contenedores en el juego de todas las cosas. A veces es posible que el usuario quiera tener instancias del mismo objeto en la escena múltiples veces como por ejemplo los mismos enemigos. Para ello se utiliza un tipo de *GameObject* denominado *Prefab* (Ver glosario). Una propiedad fundamental de estos elementos, es que siempre contienen un componente llamado T*ransform*. Este es un componente que dota al *GameObject* de propiedad de posición, rotación y escala para poder posicionar el objeto en el mundo (en tres o dos dimensiones) el cual se está desarrollando. Por otro lado, el *Transform* no es el único componente que un *GameObject* puede tener adjunto. Existe gran variedad de tipos para dotar el proyecto de todos los aspectos que el juego necesita para desarrollarse en su entereza. Hay componentes que dotan de cajas de colisiones para detectar choques en un mundo físico, componentes que dotan de una malla para dar forma (no solo forma de cubo) el objeto, componentes permiten reproducir sonido, otros que permiten el *render* de las texturas, etc. Una de las ventajas que ofrece el motor, es que los usuarios pueden programar sus propios componentes y comportamientos mediante tres lenguajes de *scripting*: C#, Javascript y Boo, una adaptación de python. Cabe destacar que en las versiones de Unity 5.x no se podrán crear más archivos en Boo, ni la documentación tendrá ejemplos suyos pero se seguirá soportando aquellos *scripts* ya creados con él. En la figura 3 podemos observar la proporción de uso de los distintos lenguajes por los usuarios[4].

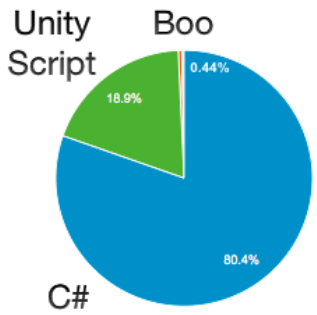


Figura 3. Gráfico de uso de los lenguajes de scripting en Unity. Unity blogs.

Estos lenguajes tienen las mismas librerías y herramientas que poseen nativamente. Además de ello, Unity ha incorporado funcionalidades para adaptarlos al uso de éstos dentro del motor. Así pues, todos los componentes que heredan de la clase *MonoBehaviour* (Ver glosario) pueden disfrutar de todas las funcionalidades otorgadas a los componentes por Unity. Estos scripts, tienen un ciclo de vida que indican los diferentes estados en los que se encuentra la aplicación y los scripts. Al cambiar de estado, se ejecutan distintos *callbacks* (Ver glosario) que pueden ser capturados por todos los herederos de *MonoBehaviour* para adaptar código a distintas situaciones.

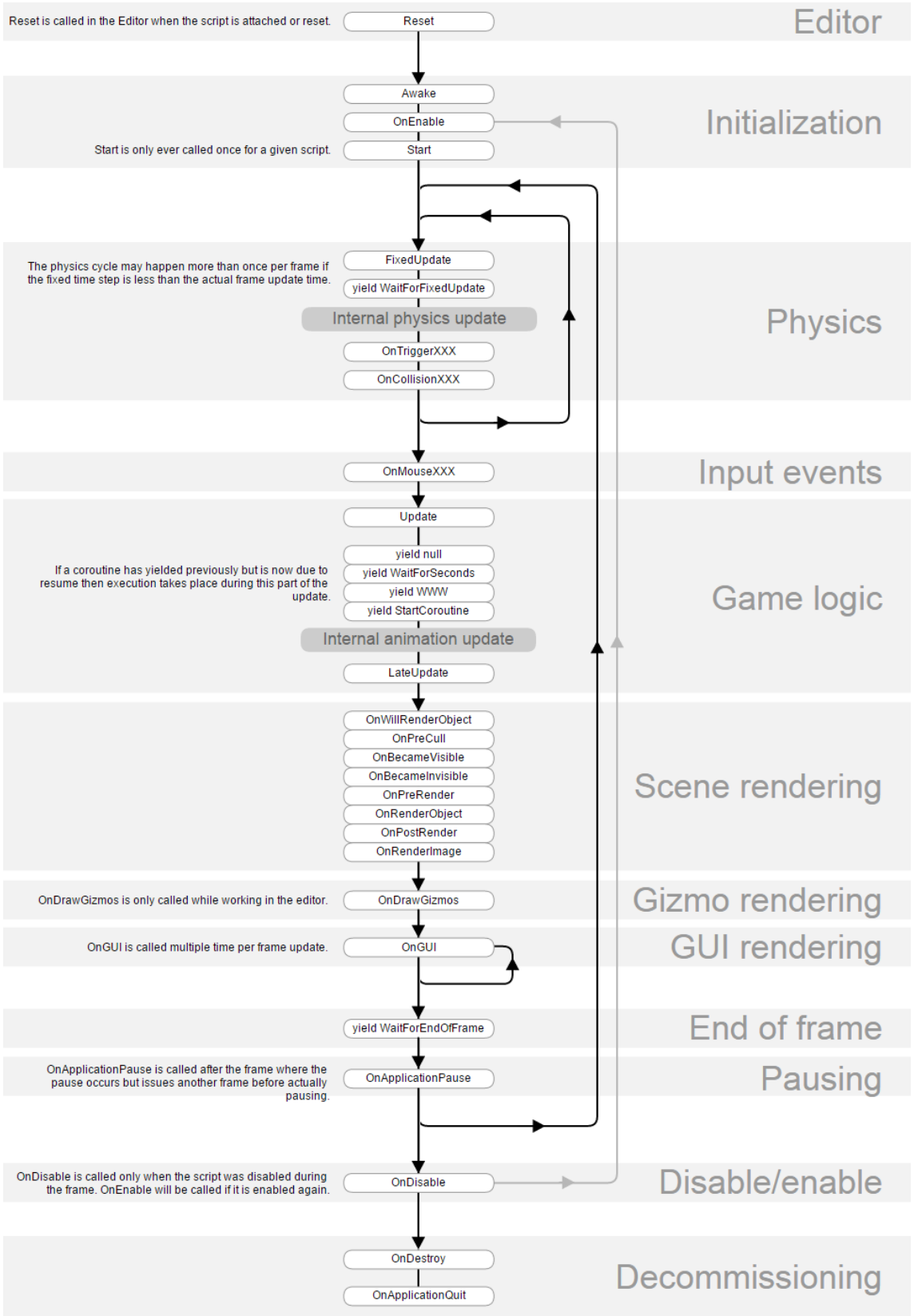


Figura 5. Ciclo de vida de un MonoBehaviour. Unity Documentation[5].

Una de las carencias del motor durante muchos años, ha sido la falta de implementación nativa de un módulo de interfaz gráfica de usuario. Durante este tiempo, la tarea de implementación de dicha interfaz recaía sobre las funciones de *scripting* de GUI, obligando a los usuarios que deseaban implementarla para su juego, a hacerlo de un modo ciego a base de programación, sin ayuda visual. Por esta razón, los miembros de esta comunidad vieron esta debilidad en el motor y desarrollaron distintos *plugins* (Ver glosario) que ayudaban a la implementación de la interfaz de usuario del juego de una manera mucho más visual y cómoda para el usuario. La denominada *Legacy GUI* se utilizaba programando dentro de un script que heredara de *MonoBehaviour* y que implementara una función llamada *OnGUI()*. Dentro de esta función el programador puede crear objetos botón, cuadros de texto y otros componentes mediante la programación y puntos de coordenadas.

```
//C#
using UnityEngine;
using System.Collections;

public class GUITest : MonoBehaviour {

    void OnGUI () {
        // Make a background box
        GUI.Box(new Rect(10,10,100,90), "Loader Menu");

        // Make the first button. If it is pressed, Application.Loadlevel (1) will be executed
        if(GUI.Button(new Rect(20,40,80,20), "Level 1")) {
            Application.LoadLevel(1);
        }

        // Make the second button.
        if(GUI.Button(new Rect(20,70,80,20), "Level 2")) {
            Application.LoadLevel(2);
        }
    }
}
```

Figura 6. Ejemplo de código fuente de LegacyGUI en C#. Unity Documentation[6].

Esta metodología es poco intuitiva. Por ello, para juegos pequeños e interfaces no muy complejas es posible utilizar este método. Sin embargo, para interfaces muy complejas o juegos con una trayectoria más profesional, este método no proporciona las comodidades para que los usuarios puedan utilizarlo bien. Por ello surgió la nueva interfaz de Unity. Una interfaz mucho más intuitiva y montada a partir de componentes como todos los otros elementos del motor. La pieza que le faltaba para poder competir a un nivel nativo con otros motores.

La nueva interfaz de usuario tiene múltiples cambios con respecto a la anterior. Al estar basada en componentes, todos los objetos de la escena deben tener el componente *Transform*, como hemos dicho anteriormente. Por otro lado, un componente de interfaz no es un elemento en tres dimensiones, sino que es un plano. Para ello, se ha

adaptado el componente T*ransform* a para conseguir el *RectTransform*. Un *Transform* pero adaptado a la nueva funcionalidad que necesita la nueva característica.

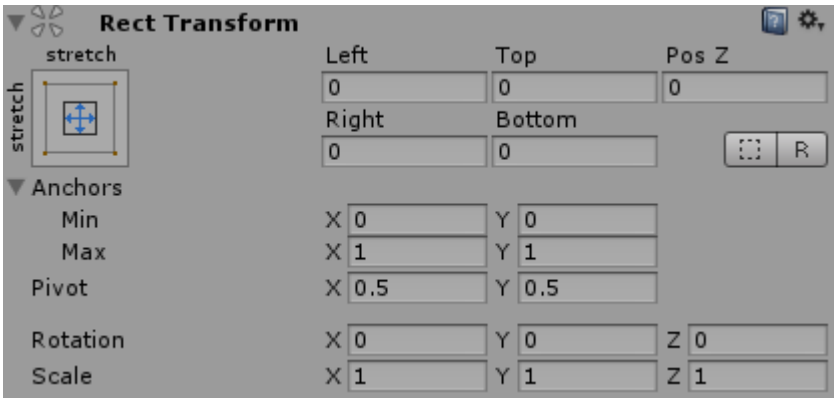


Figura 7. Vista del componente RectTransform.

En la *figura 7* podemos observar que seguimos manteniendo una concordancia con su predecesor. Vemos que sigue teniendo los campos de rotación y escalado pero por otro lado no vemos la posición. Esto es debido a que la nueva *UI* se guía por un sistema de *anchors (Ver glosario),* el cual basa su posición y tamaño dependiendo de su padre en la jerarquía. Estos *anchors,* no son más que dos puntos (cuatro valores) que determinan una recta. Estos puntos determinan cada uno una esquina del padre donde se anclaran las cuatro esquinas del hijo. Así pues al poner los valores deseados, mantendrán una proporción con el padre, haciendo variar el tamaño y posición cuando el tamaño del padre varíe. Pongamos un ejemplo. Tenemos un objeto cuyos *anchors* están situados en el punto del centro del padre como se muestra en la figura 8. Cada punto de cada esquina del hijo está relacionado con cada punto del *anchor* (triángulos blancos del centro). Así pues si el padre cambia si posición o tamaño, el hijo mantendrá la proporción y posición dentro del padre. Por otro lado, si anclamos como observamos en la figura 9, si el padre modifica su tamaño en horizontal, el hijo modificará su tamaño en horizontal estirándose pero siempre manteniendo las proporciones con respecto a la posición tanto en alto como en distancia con los lados derecho e izquierdo.

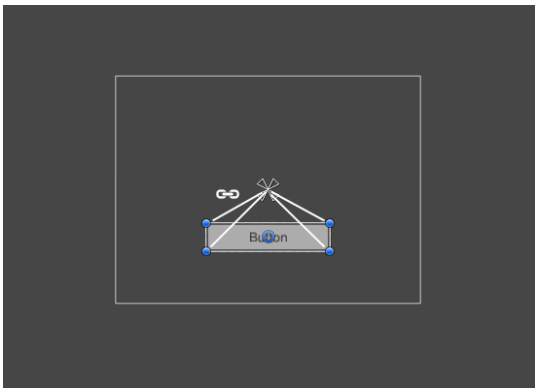


Figura 8. Ejemplo de anchors en un punto.

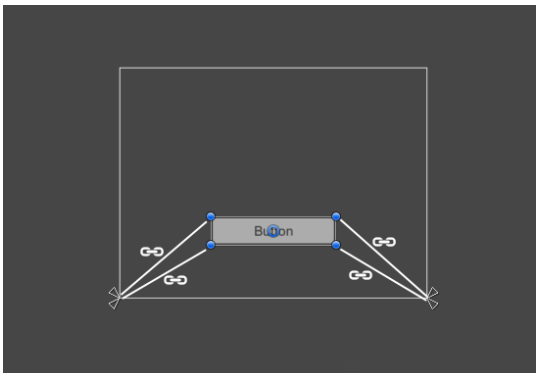


Figura 9. Ejemplo de anchors en dos puntos.

En la figura 7 también podemos observar un campo llamado *pivot (Ver glosario)*. Esto se representa en el editor con un círculo en el objeto como se puede ver en las figuras 8 y 9. Las acciones de rotación, cambiar el tamaño o la escala ocurren sobre el *pivot*, es decir, según la posición de este, la misma acción tendrá distinto efecto en el objeto. Un uso muy común es la rotación no sobre el centro del objeto sino sobre una esquina. No es necesario que el *pivot* esté dentro del rectángulo que delimita el propio objeto, puede estar fuera y conseguir rotaciones de más radio.

Al añadir cualquier tipo de elemento de la nueva *UI (Ver glosario),* si no existe en la escena se añade un elemento primordial de la nueva interfaz, el *canvas*. El *canvas* es el elemento base del cual todos los demás componentes deben que ser hijos. Todos los elementos de la interfaz tienen el componente *RectTransform* pero este es el único el cual no puede modificar el usuario ya que se adapta y auto escala con el tamaño de pantalla del dispositivo automáticamente. Para trabajar con los elementos de forma intuitiva, se ha desarrollado una forma visual de adaptar los componentes de la interfaz de forma visual, el *RectTool*. El *RectTool* es un modo de edición el cual podemos acceder a él presionando al botón que se muestra en la figura 10 y podemos encontrar en la barra de herramientas.

Figura 10. Botones de la barra de herramientas y seleccionado la herramienta RectTool.

Seleccionando la herramienta, los elementos en la escena mostraran en sus cuatro esquinas puntos de color azul además de marcar su perímetro con líneas blanquecinas y mostrar el *pivot,* como podemos ver los elementos botón en las figuras ocho y nueve.

El orden de visualización en pantalla se establece en Unity en función del orden en la jerarquía. En primer lugar se visualizarán los elementos más altos en la misma, y elementos situados más abajo podrán superponerse a los anteriores. La figura 11 muestra un ejemplo al respecto.

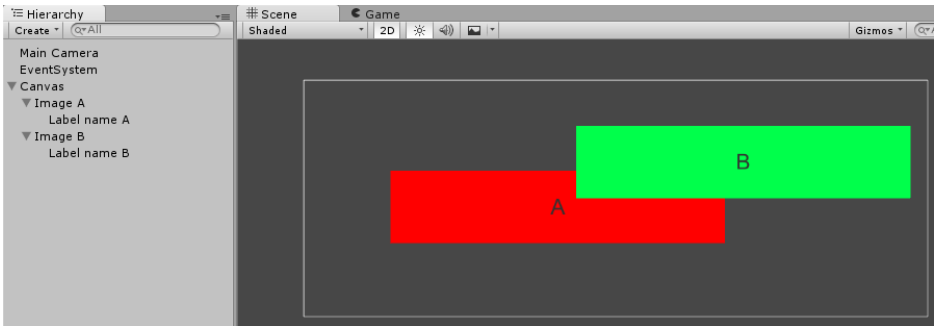


Figura 11. Ejemplo de orden de pintado en pantalla.

En dicho ejemplo también puede observarse que al añadir el *canvas* en la escena se ha añadido un componente *EventSystem* (Ver glosario), el cual es el encargado de adaptar la interfaz para recibir la interacción con el usuario. Ya sea mediante dispositivos móviles o interacción más clásica mediante teclado o ratón este componente permite utilizar la nueva interfaz en los proyectos. Este componente y los demás que componen la interfaz de Unity serán analizados más adelante.

2.1. Componentes incorporados

Vamos a analizar los componentes incorporados en la nueva interfaz. Podemos clasificar todos los elementos en cuatro categorías. Los primeros son los elementos de diseño o *layout* necesarios para poder mostrar e interactuar con las piezas de la interfaz además de todos los elementos que ayudan a clasificar y ordenar el resto de componentes para que se muestren correctamente en la pantalla. Seguidamente están aquellos elementos que reciben los eventos que genera la interfaz. Son los elementos de eventos. Luego tenemos aquellos que más simbolizan la interfaz, los elementos gráficos. Aquellas piezas que el usuario puede ver y que formarán parte de los elementos con los que finalmente interactuará. Por último, los elementos de interacción nombrados anteriormente, que el usuario podrá en la versión final utilizar mediante clics y otros eventos de entrada (dependiendo de la plataforma) para interactuar mediante la interfaz con el juego. A continuación presentaremos cada componente, su funcionalidad y los aspectos más relevantes de éstos. Nos referiremos a ellos con el nombre original en inglés tal y como Unity nos los ofrece para mayor claridad.

Elementos de diseño

- *Canvas*

Es el elemento raíz de todos los elementos de la interfaz de usuario. Todos aquellos elementos que sean hijos suyos deben tener el componente *RectTransform*. También pueden existir elementos con el *Transform* original, pero no es recomendable por estabilidad. Se pueden escoger tres modos de *Canvas*: *Screen Space – Overlay, Screen Space – Camera* y *World Space*. En el modo *Overlay* el *canvas* se adaptará al tamaño

de pantalla y siempre se pintará por encima de todos los elementos, es decir, por encima del mundo de juego que estemos creando. La modalidad de *Camera* permite la misma funcionalidad que el modo *Overlay* pero con la diferencia que toma una cámara como referencia. Lo que significa que la cámara influye en el comportamiento de la *UI*. Con este modo es posible crear efectos de profundidad en la interfaz. Finalmente la forma *World Space* permite al usuario que la interfaz actúe como cualquier elemento del juego, pudiendo así integrar totalmente la interfaz en un mundo de tres dimensiones con profundidades.

- *Canvas Scaller*

Es el encargado de adaptar la interfaz diseñada a cualquier resolución y relación de aspecto, independientemente del valor de estos parámetros cuando fue creada, es decir, dependiendo del modo y sus parámetros adaptará la interfaz al tamaño de pantalla del dispositivo. Entra en juego cuando se diseña para multiplataforma o plataformas con distintas pantallas. Por otro lado, el correcto uso de los *anchors* puede sustituir a este elemento en alguna de sus modalidades de modo de escalado si se desea. Los modos de escalado son *Constant Pixel Size, Scale With Screen Size* y *Constant Physical Size*. El primer modo es el asumido por defecto. Posiciona los elementos en la interfaz y no modifica su tamaño. El segundo, ajusta el tamaño del elemento en función de la proporción entre la pantalla original y la actual pantalla.. Finalmente el tercero no utiliza píxeles sino que utiliza unidades físicas indicadas como milímetros o puntos. Se utiliza este modo en los dispositivos que informan al usuario de su pantalla en DPI (Ver glosario) en lugar de píxeles.

- *Canvas Group*

El *canvas* group es un componente que se usa para controlar diversos aspectos de un grupo de *canvas* como la transparencia o controlar si son interactivos mediante la entrada del dispositivo.

- *Vertical, Horizontal, Grid Layout y Layout Element*

Estos componentes se utilizan para ordenar los hijos según su naturaleza. Cada hijo del *vertical, horizontal* o *grid layout* se adaptará según su orden en la lista de hijos modificando su tamaño y adaptándolo al padre. Así pues, si el padre dispone de cinco hijos, el padre con uno de estos componentes adaptará el tamaño de los hijos para que estos ocupen su tamaño total y cada uno tenga el mismo tamaño. El *layout element* se utiliza para modificar el tamaño de cada hijo si se desea que no siga las reglas del padre.

- *Content and Aspect Ratio Fitter*

Su función es la de cambiar el tamaño del componente según diferentes aspectos. El elemento *content fitter* adapta el tamaño del padre al tamaño de los hijos. Mientras que con el *aspect ratio fitter* adapta el tamaño del padre a la proporción de aspecto que se marque según la opción seleccionada y modifica el tamaño siempre en relación a la relación de aspecto dada.

Elementos de eventos

- *Event System*

Este componente es el encargado de controlar y gestionar todos los componentes de eventos de la interfaz de usuario. Controla qué *GameObject* en la *UI* esta seleccionado actualmente o controla qué entrada debe actuar con el actual dispositivo.

- *Event Trigger*

Una de las propiedades más potentes de la nueva interfaz es el *Event Trigger*. Esta herramienta permite añadir a cualquier elemento de la interfaz la recepción de eventos que nativamente no contemplaba. Por ejemplo, un botón puede recibe el evento *OnClick* nativamente, pero añadiendo este componente dicho botón reaccionar a cualquier evento de la lista de eventos. Otra posibilidad sería dotar del evento *PointerClick* a un componente *Text* y lograr un enlace en el texto.

- *Physics y physics 2D Raycaster*

Estos componentes permiten mandar mensajes a los objetos que implementen físicas en tres y dos dimensiones respectivamente.

- *Standalone y touch input module*

Los componentes *Standalne* y *Touch Input* se encargan de controlar las entradas. El primero utiliza el teclado y ratón para la navegación y selección de objetos. El segundo tiene como objetivo administrar las entradas de usuario de dispositivos con entrada táctil. Estos dos elementos pueden convivir juntos, ya que como se ha nombrado anteriormente, es el *Event System* el encargado de administrar las entradas.

- *Graphic Raycaster*

El *Graphic Raycaster* administra si un elemento de la interfaz ha sido golpeado para llamar los eventos pertinentes de cada elemento. Traza un rayo desde el clic hacia dentro, comprobando qué elementos de la interfaz han sido golpeados. Este elemento se puede configurarse para que ignore distintos elementos de la interfaz.

Elementos gráficos

- *Text*

El componente que permite introducir texto plano en la interfaz. Pude modificar su fuente, tamaño, posición, espaciado, color, etc. Además tiene la opción de *Rich Text* que permite introducir en el texto algunas etiquetas en código HTML para modificar el color, estilo y otras propiedades de diversas partes del texto.

- *Image*

Para poder visualizar las imágenes necesitamos añadir al *GameObject* este componente. Podemos añadir un *sprite* y seleccionar el tinte de la imagen.

- *Raw image*

El componente *Raw Image* sirve igual que una *image*. La diferencia es que la imagen fuente que admite para mostrar no debe ser necesariamente un *sprite, sino que, por ejemplo, es posible* escoger una textura. Es especialmente útil para mostrar

imágenes adquiridas de la web que no tengan el formato requerido por el componente *Image*.

- *Mask*

A veces es necesario que una imagen no sobresalga de un determinado espacio. Para ello tenemos el componente *Mask*. Este elemento siempre acompaña a una *Image*. La máscara definirá la forma que tenga la imagen haciendo que se visualice solo la porción de todo hijo gráfico que recaiga dentro de la misma. El programador puede elegir que se muestre la imagen que sirve para la máscara o solamente se muestre el contenido.

- *Efectos*

Estos componentes permiten añadir diversos efectos visuales a los gráficos como sombras o efecto de contorno.

Elementos de interacción

- *Button*

Elemento indispensable en toda interfaz. El botón es un elemento con una imagen que sirve de delimitador del perímetro del botón y opcionalmente otra imagen o un texto para indicar su funcionalidad. El botón recibe un evento *OnClick* para dotar de funcionalidad cuando se presiona el botón.

- *Toggle y toggle group*

Otro elemento que no puede faltar en toda interfaz es el *Toggle*. Es un elemento que estará seleccionado o no. Mediante el evento *OnValueChanged* podemos reaccionar en el código el cambio del valor de estos componentes.

- *Slider*

El *slider* es una barra de desplazamiento que tiene un máximo y un mínimo. Dependiendo del valor de la posición de la barra, seleccionado por el usuario, se devuelve un valor de estado entre ambos. El desarrollador puede mediante la entrada, cambiar el valor del componente y con el evento *OnValueChanged* actuar cuándo se modifica dicha posición, medir su valor y emplearlos en el código.

- *Scrollbar*

Similar al *slider*, su rango de valores es de cero a uno. La diferencia más significativa con el *slider* es que la barra de desplazamiento puede modificar su tamaño para adaptarse a la ventana de visualización, es decir, si se visualiza una parte significativa del contenido la barra será mayor que sí, para la misma visualización, el contenido es mucho mayor. Se utiliza para hacer *scroll* sobre el elemento *Scroll Rect* que contiene imágenes o textos muy grandes que no caben en el lugar que el desarrollador determina para ello. También es posible recoger el cambio de valor con el evento *OnValueChanged*.

- *Scroll Rect*

Se utiliza para delimitar el área donde el *scroll* se puede llevar a cabo y se va a visualizar. Puede ir acompañado de hasta dos *Scrollbars*, uno para el eje horizontal y otro para el eje vertical. También es muy común encontrarlo combinado con los

elementos *Image* y *Mask* para que la información que sobresalga de este no se muestre a los usuarios.

- *Input Field*

El *Input Field* es un cuadro de texto modificable por el usuario. Se compone de dos textos. Un texto que aparecerá por defecto y el usuario podrá aceptar y otro que es el que el usuario final entrara por teclado. Al empezar a introducir texto, el texto por defecto dejará paso al texto introducido. Por otro lado, si el usuario borra todo el texto que está escribiendo y deja en blanco el componente, el texto por defecto volverá. A diferencia de los anteriores componentes, este elemento tiene dos eventos. El primero es el anteriormente nombrado *OnValueChanged*. Por otro lado, al ser texto introducido, también dispone de un evento llamado *EndEdit* para que el código reaccione cuando el usuario deja de introducir texto.

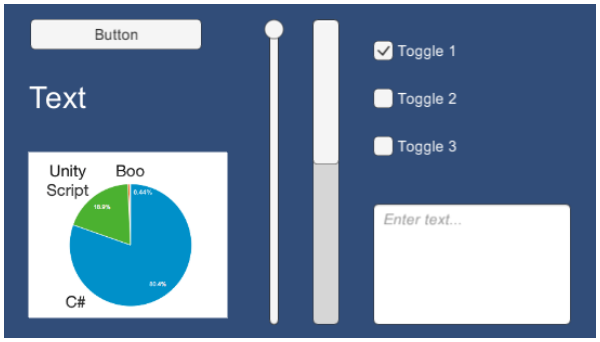


Figura 12. Vista de algunos componentes de la nueva interfaz.

Para que un método pueda ser ejecutado por un evento de la interfaz, éste debe estar declarado como público, no tener parámetros o únicamente uno de los tipos: *int, float, bool, string* y *object*. Los parámetros de tipo *object* permiten crear estructuras de datos para emplear múltiples parámetros de diversos tipos.

Además de todos los componentes descritos anteriormente, para su correcto funcionamiento son necesarias diversas herramientas y técnicas auxiliares. Unity incorpora dos herramientas llamadas *Sprite Editor* y *Sprite Packer* para poder desarrollar las técnicas *9-slice* y la técnica de los atlas.

2.2. Técnica 9-slice y Sprite Editor

La técnica del 9-slice es una técnica obligada en el uso de las interfaces actuales. Ésta técnica no solo es muy útil para reducir el número de *sprites* que tenemos en nuestro proyecto y en la aplicación final, ya que no es necesario crear una imagen para cada elemento, sino que aquellas imágenes que utilicen esta técnica, podrán reducir al

mínimo su tamaño en píxeles. Esta técnica se incorpora a Unity mediante la incorporación del modo de dos dimensiones y la utilidad *sprite slicer*. Se basa en dividir un *sprite* en nueve secciones distintas, las cuales tendrán un comportamiento distinto cuando la imagen se redimensione al tamaño que el usuario desee. El *sprite* no tiene que ser necesariamente cuadrado ni potencia de dos, aunque es recomendable para mejorar la compresión. Al dividir la imagen, se colocan cuatro líneas, dos horizontales y dos verticales paralelas entre ellas y a los ejes 'x' e 'y', formando nueve paralelogramos.

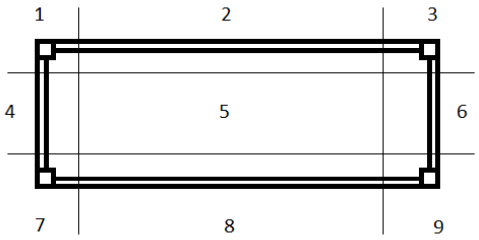


Figura 13. División de un sprite para 9-slice.

Habiendo obtenido los nueve paralelogramos, el comportamiento de la imagen frente al redimensionamiento cambiará de un modo drástico. En lugar de cambiar su forma de modo lineal, es decir, manteniendo las proporciones del original, se modificarán según esta tabla:

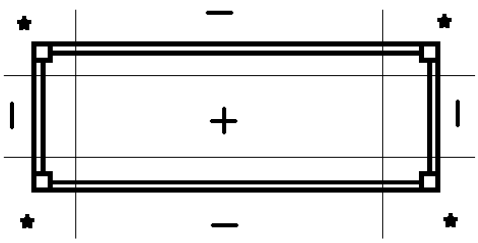


Figura 14. Comportamiento frente al escalado con la técnica aplicada.

Como podemos observar, las cuatro esquinas están marcadas por un asterisco. Esto significa que esta área no se verá afectada frente al cambio de tamaño, quedando con las mismas dimensiones siempre. Esto hace que los cantos queden siempre bien y no se deformen con el estirado. Seguidamente, las superficies marcadas en la figura 13 con los números 2 y 8 se alargarán en el eje horizontal. Del mismo modo, las zonas que se delimitan con los números 4 y 6 de la misma figura, harán lo mismo pero en sentido vertical. Finalmente, la región numerada con el cinco se expandirá tanto en horizontal y verticalmente para cubrir toda el área. Para poder utilizar esta técnica en Unity, hay que ir a cada *sprite* en nuestro proyecto y en el inspector hacer clic sobre el botón de *Sprite Editor*. Esta acción abrirá la ventana del editor. En ella podemos situar las líneas que formará el *9-slice* de dos formas. La primera es de una manera visual arrastrando las cuatro líneas verdes situadas en los bordes. La segunda es mediante el número de

píxeles exactos. Escogemos el número de píxeles que deseemos que disten de los bordes de la imagen y lo introducimos por teclado.

2.3. Atlas y Sprite Packer

Otra de las posibilidades que ofrece la nueva interfaz de Unity, es la posibilidad de crear nativamente un Atlas mediante de las texturas del proyecto. La técnica del atlas, consiste en agrupar diversas texturas pequeñas en una más grande. Mediante esta opción, conseguimos reducir las llamadas a la API gráfica (Ver glosario). Todos los *spites* que se incluyan en el atlas, generarán una única llamada a la API gráfica, reduciendo considerablemente el tiempo de respuesta. El tamaño máximo del Atlas en Unity es de 2048 x 2048 píxeles. Si al insertar más imágenes en el atlas el espacio es insuficiente, se creará una nueva página en el atlas. Esto significa que se creará un atlas con el mismo nombre pero no en el mismo atlas. Esto conlleva que si se llama a un *sprite* de cada página del atlas, generará dos llamadas a la API gráfica. Para poder utilizar esta técnica en Unity, hay que ir a cada *sprite* en nuestro proyecto y en el inspector hacer clic sobre el elemento *Packing Tag* e introducir el nombre del atlas en el cual deseamos introducir el *sprite*. Añadir el mismo nombre no significa que se añada en el mismo atlas ya que la imagen debe tener los mismos ajustes de importación para agruparlos juntos.

3. El plugin NGUI

El *plugin Next Generation UI* (de ahora en adelante NGUI) es uno de los predecesores de la actual interfaz de usuario de Unity. Su creador Michael Lyashenko y su equipo participaron en la creación e implementación de la nueva interfaz de Unity, dado su éxito cosechado anteriormente con su software. Al ser un *plugin*, el usuario debe adquirirlo a través de la *Asset Store* de Unity. Una vez adquirido, debe descargar e importar el contenido del paquete en el proyecto.

Aunque NGUI sea el predecesor de la nueva interfaz de Unity, no son idénticamente iguales. Existen diferencias significativas en el funcionamiento de ambas herramientas, aunque su uso es muy parecido. Lo primero en que debemos fijarnos es que al no ser nativo del motor, no tiene mucha integración en este. Así que, todos los elementos de control de la interfaz los manejan scripts del propio *plugin*. Otro aspecto de esta característica es que todos los *GameObjects* de la interfaz tienen el componente *Transform* necesario en estos en lugar del *RectTransform*. Otra diferencia que el programador debe tener en cuenta a la hora de usar NGUI son los *anchors*. A diferencia que Unity, NGUI no se basa en la jerarquía de padres e hijos para anclar los componentes. Para esto se utiliza una referencia al *Transform* al cual se anclará el objeto. Esto permite la independencia de los objetos con la jerarquía y aporta al usuario mucha más libertad a la hora de organizar su escena como prefiera. Como se puede observar en la figura 15, NGUI también tiene los indicadores azules de las esquinas para delimitar el área del *sprite*. Además incluye cuatro adicionales en los ejes 'x' e 'y' para ensancharlo en cada uno. En la figura 15 también encontramos representados los *anchors*. Se muestran en la escena mediante un punto amarillo unido a cada punto azul de los ejes cartesianos mediante una línea amarilla con el número que indica la cantidad de píxeles que dista del punto de anclaje. Otra característica muy importante también es el centro o *pivot*. En este caso, la representación es menos intuitiva. Fijándonos en la misma figura, podemos ver la característica *Widget* en el inspector. En ella encontramos el *pivot* y podemos determinar su posición mediante los botones. Al seleccionar un *pivot* distinto del centro, es decir en uno de los ocho puntos, el punto seleccionado pasara a tener un color naranja avisando al usuario del cambio. No se permiten *pivots* que no sean uno de estos puntos exactamente.

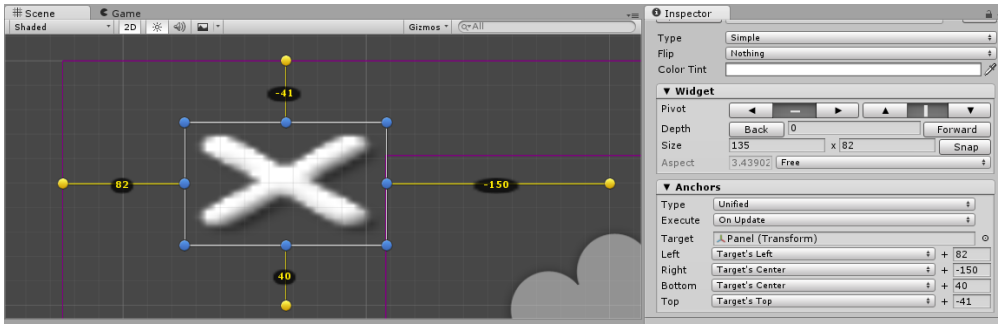


Figura 15. Sprite anclado al GameObject Panel.

Al añadir el primer elemento de NGUI a la escena se crea un elemento que es necesario para el uso de la interfaz, el *Root UI*. Es el equivalente del componente *Canvas* en la nueva interfaz. En ella se crean diversos elementos. Además, todos los elementos de la interfaz son creados en una capa especial que NGUI crea en Unity llamada 'UI'. El primero es el *GameObject* que contiene el script *UI Root*. Como hijo de este, se añade una cámara. Ésta es la encargada de mostrar todos los componentes de la interfaz en la pantalla. Para ello, la cámara contiene una cámara estándar de Unity que solamente muestra por ella los componentes que pertenezcan a la capa UI. El segundo componente es una *UI Camera* que sirve de controlador de colisiones para lanzar los eventos correspondientes. Es similar al elemento *EventSystem* en la nueva interfaz. Por otra parte, es imprescindible añadir otro componente a los elementos con el fin de poder hacerlos interactivos. Se debe añadir un elemento *Box Collider 2D* para que los elementos puedan recibir la pulsación en ellos. Por otro lado, al contrario que en la interfaz nativa, NGUI utiliza otro sistema para el orden de pintado. Cada elemento gráfico posee una propiedad *Depth* dentro de *Widget* que permite elegir la profundidad donde se encuentra el objeto. Esto, junto a la característica de anclado independientemente de la jerarquía, hacen que sea muy fácil controlar por parte del usuario las posiciones de los objetos dentro de la vista *Hierarchy*.

3.1. Componentes incorporados

Una vez que el usuario dispone de todas las herramientas incorporadas está en plena disposición de crear interfaces mediante NGUI. Lo primero que podemos observar, es que se ha añadido a la barra superior ha añadido un nuevo menú llamado "NGUI". En este menú es posible utilizar todos los aspectos de NGUI de una manera intuitiva y sencilla. Encontramos nueve elementos en disponibles. Primero encontramos el submenú *Selection*. En este submenú podemos ejecutar diversas acciones para un componente previamente seleccionado. Como por ejemplo traerlo al frente, traerlo al fondo o ajustar en uno su número de capa de pintado. En segundo lugar encontramos el menú *Create*, que permite crear los elementos de la interfaz. Seleccionando alguna opción del submenú, NGUI creará el objeto seleccionado en nuestra jerarquía. En tercer lugar esta *Attach*. Su función es la de añadir componentes a elementos previamente creados y seleccionados. Aquí podemos incluir todos los elementos interactivos a nuestros objetos. Más adelante se explican los componentes gráficos e interactivos. Seguidamente encontramos el apartado encargado de aplicar transiciones a la interfaz, el submenú *Tween*. Al no estar tan integrado nativamente, deben añadirse las transiciones mediante scripting y cálculos de posiciones. Con ellos, podemos aplicar transiciones a propiedades como la posición, rotación, transparencia o color. Luego vemos el submenú *Open*. Se usa para abrir diferentes funcionalidades y herramientas que ayudan al uso del *plugin* como el *Panel Tool, Camera Tool* o *Atlas Maker* y *Font Maker* que veremos en profundidad más adelante. El siguiente submenú es *Options*. En él se puede configurar distintos aspectos del *plugin*. *S*eguidamente encontramos el apartado *Extras* el cual nos permite ejecutar algunas acciones útiles. Podemos cambiar todas las cajas de colisión de dos dimensiones a tres y viceversa o centrar la vista de escena sobre la interfaz que estamos desarrollando. Después, encontramos una gran

utilidad de cara a la gestión de la profundidad de los paneles. Con la opción *Normalize Depth Hierarchy* NGUI puede leer todas las profundidades de las capas y normalizarlas para que el usuario no tenga muchos valores dispares. Esto permite al usuario tener siempre ordenado el número de profundidad de cada panel con respecto a los otros. Para finalizar, encontramos una opción de ayuda la cual nos lleva al foro oficial del software donde poder buscar toda la información referente a nuestro problema.

Al igual que Unity, NGUI contiene componentes imprescindibles para la creación de las interfaces. Podemos dividirlos en cuatro grandes grupos. El primero contiene los elementos de diseño, aquellos que son los encargados de mostrar, organizar y tratar los elementos de la interfaz. Seguidamente, se encuentran todos los elementos gráficos que muestran información por pantalla. En tercer lugar, otros componentes que permiten al usuario dotar de otra funcionalidad a la interfaz como transiciones, localizaciones, sonidos o animaciones. En estas últimas no entraremos en gran detalle. Finalmente, se encuentran los componentes de interacción, responsables de la interacción del usuario con el software.

Elementos de Diseño

- *2D, 3D UI*

Este es el componente encargado de administrar toda la interfaz, como se ha descrito anteriormente. Contiene un componente *UI Root* el cual se encarga de escalar la interfaz mediante el tamaño de pantalla del dispositivo. Admite tres tipos de escalado. El primero es *Flexible*. Este modo permite que el tamaño de la interfaz esté especificado en píxeles. El problema de este modo es que en dispositivos con una baja resolución los menús se verán más grandes de los que realmente son. Seguidamente encontramos la modalidad *Constrained*. Este es el modo opuesto al anterior. En lugar de medirse por píxeles se mide por proporciones. Esto quiere decir que si un elemento de la interfaz mide en píxeles 400 x 800 en una pantalla de 800 x 800, cuando el usuario cambie la resolución, siempre mantendrá la proporción con respecto a la resolución base. En último lugar encontramos una combinación de los dos modos anteriores, *ConstrainedOnMobiles*. El cual se comportará en dispositivos de escritorio como si fuera *Flexible* y *Constrained* en el caso de *build* para móviles.

- *Widget*

Es un componente imprescindible de la interfaz. Es el que dota a los elementos de las características color, centro, profundidad y tamaño. No es un elemento el cual el usuario pueda escoger para la interfaz. Lo llevan integrado aquellos elementos que lo necesitan. A pesar de no poder escogerlo el usuario voluntariamente, es necesario mencionarlo dada su gran importancia.

- *Panel*

Otro elemento muy importante del software es el *Panel*. Es el componente encargado de obtener, administrar y mostrar todos los elementos por debajo de él. Es el encargado de efectuar las *Draw Calls* y por tanto de una manera indirecta, pintar los elementos en la interfaz (de una manera indirecta, ya que el elemento que los muestra al usuario es la cámara de la *UI)*. Un uso importante de los paneles es el de administrar los gráficos y elementos de interacción por secciones. Como se ha dicho anteriormente, los componentes gráficos poseen un elemento *Depth* para organizar su profundidad en

el orden de pintado en la pantalla. El elemento *Panel* también lo posee. Esto es debido a que la profundidad de los paneles predomina sobre la de los objetos. Por ejemplo, las imágenes contenidas en paneles con menor profundidad no serán visibles aunque su profundidad sea superior a las de paneles de mayor profundidad. Así pues, podríamos decir que el panel es el encargado de hacer *render* y administrar cada contenido.

- *Grid*

  Es un componente auxiliar que permite al usuario ordenar objetos vertical u horizontalmente de una manera sencilla e intuitiva. Agrupa elementos del mismo tamaño mediante celdas de un tamaño específico introducido por el usuario. También permite una separación entre elementos.

- *Table*

  Este elemento es muy similar al componente *Grid*. La diferencia radica en que no posee celdas predefinidas, sino que se autoajusta al tamaño del contenido. Permite especificar el número de columnas (o filas en caso de tener la orientación horizontal) para ordenar los elementos que contiene. Además, permite contener elementos que tengan distintos tamaños.

Elementos Gráficos

- *Texture*

  Es el elemento base de todas las imágenes. Permite escoger una imagen fuente y mostrarla. También permite controlar los *UV* (Ver glosario), pudiendo cambiar los valores para adaptar la posición y tamaño de la textura al contenido. Tiene distintas opciones para la imagen como restablecer el tamaño predeterminado de la imagen o cambiar el tipo de relación de aspecto. Finalmente, como se ha hablado en el apartado de panel, posee un componente *Widget* que es el encargado de posicionar su *pivot*, escoger sus dimensiones y su profundidad.

- *Sprite*

  Tiene la misma funcionalidad que el componente *Texture* pero, al contrario que *Texture*, que utiliza una imagen directamente como fuente, el elemento *Sprite* requiere que la imagen provenga de un Atlas. Para ello, la imagen previamente debe haber sido insertada en él. Para ello se usará una herramienta muy útil incorporada a NGUI llamada *Atlas Maker*.

- *Label*

  Toda interfaz necesita un elemento para mostrar texto. Para ello se utiliza un componente *Label*. Este componente requiere de una fuente para mostrar el texto que el usuario desee introducir. Además de esto, tiene otras muchas opciones para tratar y modificar el texto introducido como el tamaño, el comportamiento del texto cuando es más grande que el tamaño que lo contiene, el espaciado entre las líneas o entre los caracteres o el color. Finalmente, cabe destacar que NGUI posee una herramienta llamada *Font Maker* para modificar fuentes.

Elementos de transiciones

- *iTweens*

Existen *scripts* para hacer un interpolado de diversas características del objeto que lo contiene, como por ejemplo la transparencia, el color, la posición, la rotación, la escala, etc.

Elementos de Interacción

- *Botón*

Es el componente necesario para recibir los eventos de *clic*. Pero necesita un objetivo para poder aplicar las transiciones. La gran ventaja de este componente es que no es necesario que el elemento que queramos que sea botón tenga un elemento *Widget*. Es posible añadir a cualquier elemento el componente para dotarlo de esta funcionalidad. El botón posee cinco elementos para la administración de cambio de estado del botón. Los posibles estados son normal, remarcado, pulsado y desactivado. Para cada uno de estos estados es posible cambiar el color del objetivo o incluso cambiar a una imagen. El tiempo que dura la transición también es configurable por el usuario. Para acabar, contiene un evento para que el usuario decida qué función o funciones se puedan ejecutar al recibir el clic en el botón. Con respecto al evento clic, es necesario que el *GameObject* que contiene el botón posea un *Box Collider* para recibir el evento.

- *Toggle*

Es un componente muy básico. Tiene dos estados, encendido y apagado. El cambio entre los estados puede ser recogido mediante el evento *OnValueChanged*. El usuario puede gestionar la información del cambio de estado como prefiera por código. Como con el componente botón, es posible gestionar las transiciones. Para finalizar, es posible agrupar varios *Toggles* en un mismo grupo para que solamente uno de ellos pueda estar seleccionado. Para ello, todos los *toggles* de un mismo grupo deben tener la propiedad *Group* al mismo valor.

- *Slider*

El elemento *Slider* se compone de una barra, un fondo de dicha barra y un objeto de deslizamiento. Este objeto siempre se encuentra dentro de los límites del fondo y al final de la barra. Su función es la de ser usado mediante arrastre para cambiar el valor actual. Al igual que el componente *Toggle* posee una llamada al evento *OnValueChanged* que permite adquirir el nuevo valor cuando se modifica. El usuario puede escoger el número de pasos desde el inicio de la barra hasta el final.

- *Scroll Bar*

Un elemento muy similar al anterior. La diferencia radica en que no existe barra desde el inicio hasta el valor sino que se muestra solamente el objeto que se usa para modificar dicho valor.

- *Progress Bar*

De este componente heredan los dos anteriores. Implementa la funcionalidad y lógica de la barra pero sin tener el objeto que hace servir el usuario para cambiar el valor.

- *Scroll View*

Al crear este componente se añaden dos elementos. El primero es un panel que manejará todos los elementos gráficos. Otro es el propio elemento *Scroll View*. Es el encargado de administrar las propiedades del componente. En el podemos escoger la dirección del movimiento, el tipo de desplazamiento, la velocidad y además se pueden añadir dos *Progress Bar* (o todo componente que herede de él como *Scroll Bar* o *Progress Bar*), para gestionar el desplazamiento en cada eje. Es imprescindible para el funcionamiento de este componente añadir un *Box Collider* para poder registrar la entrada y un elemento *Drag Scroll View* para permitir que la vista pueda desplazarse.

- *InputField*

Para finalizar, el componente *Input Field*. Es el componente que permite al usuario introducir texto para que el software pueda gestionarlo como deba. Precisa de un componente *Label*. Tiene diversas propiedades que permiten que el componente sea más dinámico. Por ejemplo, un campo para dar un valor predeterminado al texto cuando no tenga valor. También tiene transiciones entre estados y cambio de color dependiendo del estado. Es posible también cambiar el tipo de teclado que se muestra al clicar o añadir un límite de caracteres para introducir. Para acabar, se pueden ejecutar dos eventos en este componente. El primero es *OnSubmit*. Este evento se invoca cada vez que el usuario pierde el foco del InputField. El segundo evento es *OnChange*.

Figura 16. Algunos elementos de NGUI.

3.2. Atlas Maker y Font Maker

Anteriormente se ha nombrado el *Atlas Maker*, necesario para crear los atlas de NGUI. Es imperativo para poder utilizar los componentes *Sprite* ya que requieren una imagen que pertenezca a un atlas. Además de esto, esta técnica permite ahorrar en *Draw Calls* (Ver glosario) ya que todo que todo acceso a los *sprites* del atlas genera una única *Draw Call*. NGUI viene con distintos atlas predefinidos para que el usuario pueda utilizarlos. Por otro lado, en casi la totalidad de los proyectos el usuario deseará crear sus propios Atlas. Para ello, debemos abrir el *Atlas Maker* haciendo clic en NGUI/Open/Atlas Maker, o con clic derecho en la pestaña del proyecto/NGUI/Open Atlas Maker. Una vez abierto podemos seleccionar un atlas existente para modificarlo. A la derecha encontramos un botón llamado *New* que permite crear un nuevo Atlas. Una vez pulsado, debemos seleccionar todas las imágenes que queramos añadir al atlas en el inspector, manteniendo el *Atlas Maker* abierto, y pulsamos al botón *Create*. Una vez pulsado, insertará las imágenes seleccionadas en el atlas y nos pedirá la ruta de guardado y su nombre. El proceso de modificar el atlas es el mismo. En caso de borrado, cada *sprite* tiene al lado un botón con una cruz que permite eliminarlo. Si no hay seleccionada ninguna imagen con la vista del *Atlas Maker* abierta, el botón *Create* se transforma en *View Sprites* la cual permite mostrar en otra ventana todos los elementos que se encuentran en él. Cuando creamos el nuevo atlas, podemos observar que no es más que un *prefab* de Unity. Además de este objeto, se han creado dos elementos más. Una textura que compone todos los elementos que componen el atlas y un material a partir de esta textura. Podemos localizar estos componentes en el proyecto pulsando en el *Atlas Maker* a los botones de *Material* y *Texture*, los cuales seleccionarán en el proyecto cada elemento del atlas seleccionado. Finalmente, existen diferentes opciones para el atlas como cambiar manualmente las dimensiones de cada *sprite* en el atlas, guardar las nuevas dimensiones como una imagen nueva, cambiar el color de fondo o añadir efectos a la imagen. Para ello debemos seleccionar el *prefab* atlas en el proyecto y en el inspector cambiar alguno de estos aspectos.

Por otro lado, no solo las imágenes se pueden añadir a los atlas, NGUI permite también insertar fuentes para ahorrar en llamadas a la API gráfica. Para ello, el usuario debe abrir mediante el mismo procedimiento que el *Atlas Maker* la herramienta *Font Maker*. Para poder insertarla en un atlas, debe existir éste previamente. Una vez abierta la herramienta, seleccionamos la fuente en nuestro proyecto y el tamaño. Para finalizar, seleccionamos el atlas en el cual insertamos la fuente y presionamos el botón *Create Font*. El software nos pregunta por la ruta y el nombre del archivo. Guardamos, al igual que con el *Atlas Maker*, un elemento *prefab*. Al finalizar el proceso, la fuente se habrá incluido en el atlas y estará lista para su uso. Observando el objeto creado, podemos ver que tiene un componente *UI Font* en él. Este *script* nos indica el tipo de fuente, el atlas al que pertenece y el nombre de la fuente. También es posible añadir un emoticono al texto. Solamente hace falta seguir las instrucciones indicadas y seleccionar un *sprite*. Al igual que en las imágenes, nos permite añadir diversos efectos a la fuente como sombras o transparencia.

4. Comparativa de herramientas

Una vez tenemos conocimiento de ambas herramientas, procedemos a la comparación de ambas para determinar ventajas, desventajas y carencias de ambas. Para ello debemos primero que nada fijar unos parámetros de comparación. En base a estos parámetros, decidiremos que herramienta es mejor en cada aspecto. Se aportarán gráficas de diversos y resultados para ilustrar mejor el resultado de las pruebas.

4.1. Parámetros de comparación

- *Precio*

Compararemos el precio de ambas herramientas.

- *Integración, usabilidad y funcionamiento*

Integración de ambas herramientas dentro del propio motor gráfico, además compararemos la facilidad o complejidad de uso de ambas y finalmente también se equipararán los funcionamientos y robustez del uno y el otro.

- *Soporte y Actualizaciones*

Comparativa del soporte técnico, comunidad y actualizaciones oficiales de ambas herramientas.

- *Rendimiento*

Compararemos el rendimiento mediante distintas pruebas creando distintas interfaces, analizando el tiempo de respuesta, llamadas a la API gráfica, número de objetos necesarios, etc.

4.2. Estudio

La nueva interfaz de Unity es el legado que NGUI deja y gracias a esta, ha sido posible su desarrollo. A pesar de ello, las dos herramientas no son idénticas en cuanto a su funcionamiento tanto interno como externo. Por eso al coexistir ambas en el mercado actualmente, se ha realizado un estudio para averiguar puntos fuertes y débiles de ambas.

Precio:

La primera característica que se ha comparado entre ellas ha sido el precio. Una variable importante dependiendo de qué tipo de usuario lo precise. El precio de NGUI es de 95$, ya que es un *plugin* de terceros, mientras que la interfaz de Unity es gratuita, integrada como una característica del motor. Esta característica no es decisiva per se,

pero tiene mucha importancia ligada a otras comparaciones que más adelante desarrollaremos.

Integración:

Otras características que se han estudiado han sido la integración, la usabilidad y el funcionamiento. Mientras que NGUI funciona con un orden de *renderizado* mediante profundidades de capas, Unity funciona mediante el orden en la jerarquía. Por un lado, NGUI ofrece una versatilidad muy grande para el usuario, permitiéndole independencia en la jerarquía y quebraderos de cabeza al intentar ordenar difíciles combinaciones. Por otro lado, el método que ofrece Unity permite una gran organización pero es menos flexible.

Seguidamente, el tipo de componentes tampoco es el mismo. Mientras que NGUI se compone de más elementos utilizables por el usuario para amenizar la utilización y construcción de los elementos de la interfaz, Unity solamente proporciona los elementos básicos para que todos aquellos usuarios que quieran puedan construir elementos cada vez más complejos con piezas básicas. Por otro lado, esta simplicidad puede hacer que se pasen por alto elementos básicos y necesarios. Por ejemplo centrar en un elemento de un *scroll*, que más adelante veremos cómo se ha suplido esta carencia. Por ello, en este aspecto, NGUI puede tener la ventaja sobre Unity al introducir componentes más avanzados pero igual de útiles para el usuario. Aunque también cabe destacar que es posible que incorpore elementos que no sean estrictamente necesarios como elementos primarios, como por ejemplo la tripla *Slider, Scroll Bar* y *Progress Bar*.

Por contraposición, NGUI no es un elemento nativo de Unity, por lo tanto precisa de toda la infraestructura para su correcto funcionamiento. Esto es, necesita *scripts* propios para el control de las transiciones, depurado, etc. Así que, hablando en términos de integración en el motor, es lógico que la herramienta nativa esté más integrada en él. Por consecuencia, todas las utilidades que integra Unity son utilizables por la propia interfaz. Por ejemplo, el *profiler*. El *profiler* es una herramienta muy útil que incorpora Unity que permite mostrar al usuario diversos datos para ayudar a optimizar su proyecto. Mira diversos componentes como CPU, GPU, r*endering,* memoria, audio, físicas y red para mostrar en cada apartado información de interés para el usuario. Además, permite filtrar esta información o hacer una búsqueda en profundidad. Al ser un componente nativo del motor, muestra información gráfica mucho más detallada frente a NGUI que debe proporcionarla por sus propios métodos.

Otra característica que está muy integrada con la interfaz de Unity son las animaciones. El *Animator* permite animar todos los elementos nativos de Unity, tanto posiciones, rotaciones, físicas, cajas de colisión, todo. Al ser la interfaz un componente nativo del motor, también es posible animar mediante este componente. Así todo queda mucho más integrado, sencillo e intuitivo.

Por otro lado, hablando de integración también es necesario hablar de integración para la comodidad de usuario. Unity proporciona una integración para la creación de elementos de la interfaz en el motor muy cómoda permitiendo crear elementos de diversas formas como por ejemplo con clic derecho en la jerarquía, atajos de teclado o en la barra de herramientas. Además, no es necesario adjuntar los *scripts* objetos vacíos

creados anteriormente ya que Unity crea los objetos y añade los componentes y jerarquiza los objetos para que estén como deben. Mientras que NGUI, precisa que para la creación de algunos objetos se creen desde cero y no existe ningún estándar de jerarquía para los componentes. Por ello, Unity se hace fuerte frente NGUI en este aspecto.

Soporte:

El área de soporte lo podemos dividir en dos sectores, el soporte oficial o soporte que realiza el desarrollador y el soporte de la comunidad. Centrándonos en el soporte oficial, Unity posee un gran manual en lo que al motor respecta, pero falla en la interfaz. También es debido a que es una herramienta acaba de incorporar. Aunque cada vez más se está añadiendo contenido. Esto demuestra que no todo está visto y que queda mucho por ver. De otro lado, NGUI trae una veteranía de por sí. La web de ayuda del *plugin* es extensa y está llena de ejemplos muy útiles de mano del propio desarrollador. Pero, la web no presenta un aspecto de profesionalidad a primera vista, ya que se trata de un hilo de un foro. Mientras, el soporte de la comunidad es muy extenso en ambas herramientas, aplicando los anteriores hándicaps en ambas utilidades. Mientras Unity parece más flojo en el la documentación, está mejor posicionado en las actualizaciones. Se solucionan errores en las actualizaciones del propio motor. Además, en errores críticos se publican parches para que los usuarios pueden continuar con su proyecto. Esto quiere decir que las actualizaciones son constantes.

Rendimiento:

Es posible que cada uno tenga sus defectos y sus perfecciones, pero dónde al usuario finalmente desea obtener resultados en el rendimiento de la herramienta. Por ello, se han diseñado pequeñas pruebas de rendimiento de ambas interfaces. La primera consta de una interfaz común, donde se estudiará el rendimiento de una interfaz que un juego del mercado podría tener. Mientras que la segunda constará de un test de estrés. Constará de un elemento *scroll* con un número distinto de hijos y se compararán diferentes parámetros: tiempo de instanciado, llamadas a la API gráfica, *frames* por segundo de la aplicación y variación de los *frames* con el *scroll* en movimiento. Para realizar estas pruebas se ha utilizado un ordenador de prueba con las siguientes características:

- Ordenador portátil Asus K56CB
- Sistema operativo Windows 8.1, 64 bits
- Procesador Intel Core i5-3337U – 1.80GHz – 4 núcleos
- 4 GB de memoria RAM
- Tarjeta gráfica NVIDIA GeForce GT 740M – 2GB dedicados – DDR3

Para la primera prueba, se han diseñado dos escenas. Una contiene la interfaz de un videojuego común con NGUI y otra contiene lo mismo con componentes nativos de Unity. En ellas encontramos dos elementos *slicer*, que actúan de barra de vida y barra de maná, tres elementos botón, dos elementos *label* y un *scroll*. Para mirar rendimiento

en ambas en esta prueba se han comparado los objetos necesarios para crear la escena, el tiempo del hilo principal de ejecución por *frame* y los *frames* por segundo.

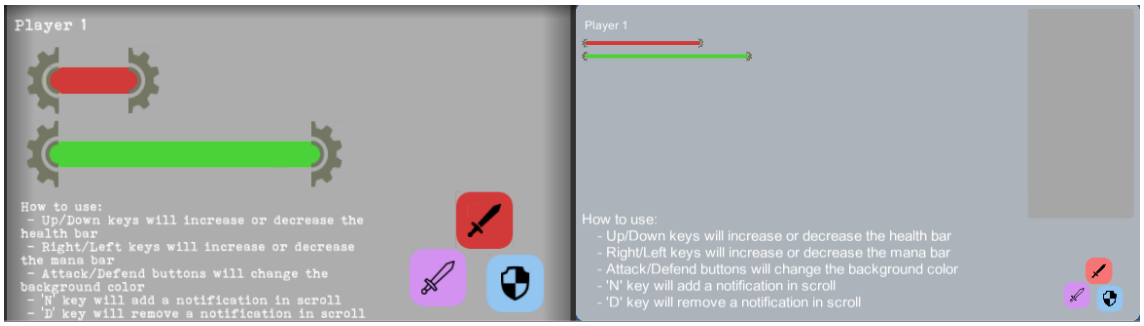


Figura 17. Interfaces creadas para el Test 1.

Los resultados obtenidos han sido los que se muestran en la figura 18. Como podemos observar, ambas herramientas están muy equitativas. La única discrepancia que es observable, es el número de *Draw Calls*. Podemos afirmar pues, que para una interfaz simple ambas herramientas son viables.

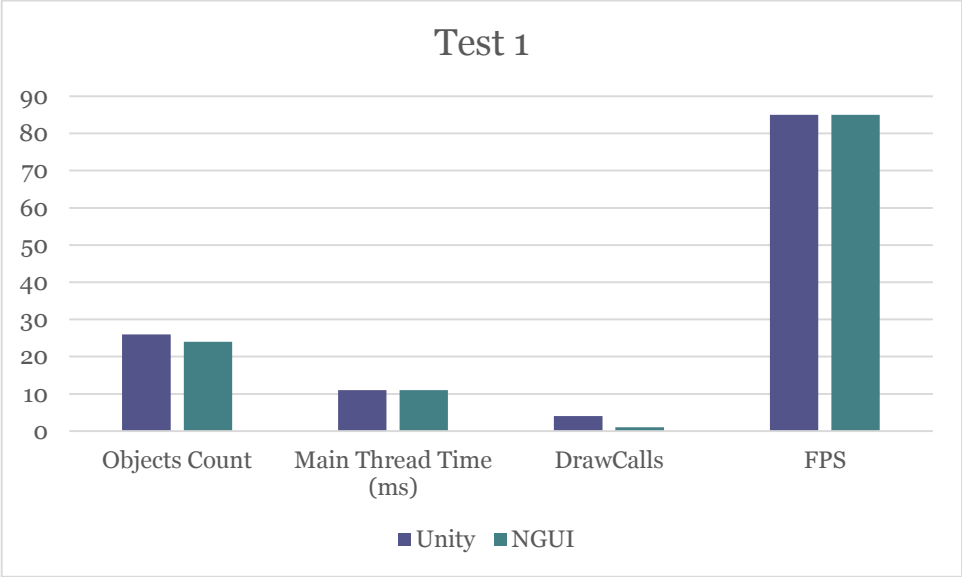
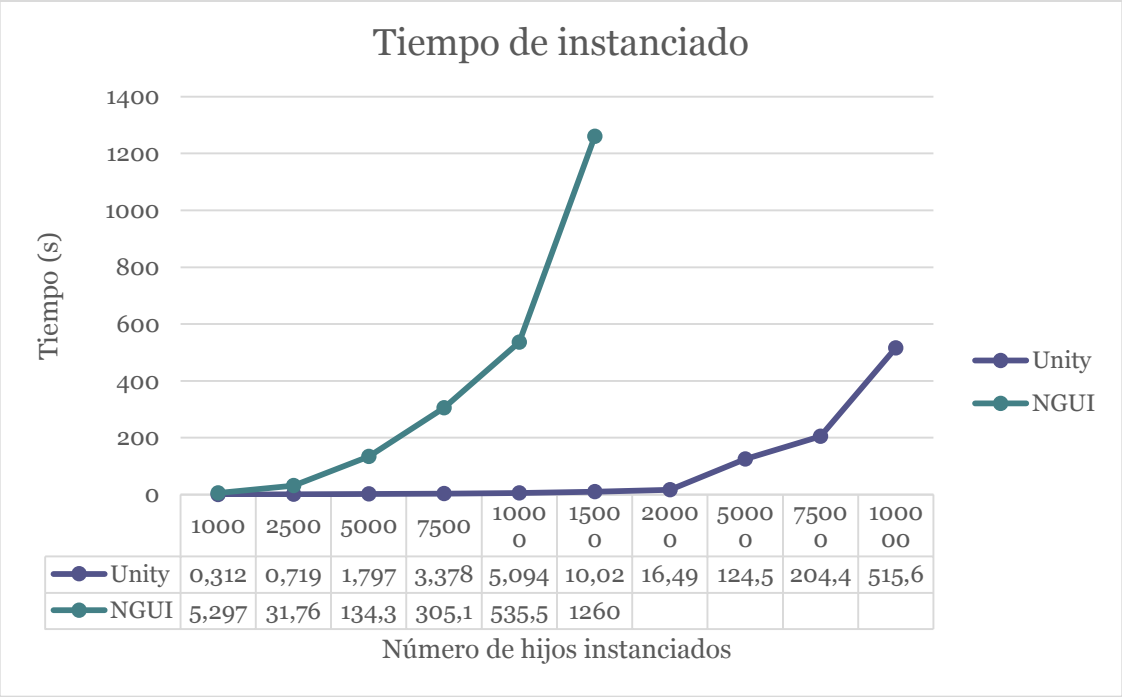


Figura 18. Resultados del Test 1.

Finalizado el primer test, pasamos al segundo. Tiene como objetivo estresar las interfaces creadas con las dos herramientas. Para ello utiliza un sencillo método. Consta simplemente de un *scroll* en el cual se instanciarán diferente número de hijos y mediremos los mismos parámetros mencionado anteriormente además del tiempo de instanciado de los elementos y la caída de *frames* al hacer *scroll* sobre la vista. Para medir el tiempo de instanciado, se han tomado dos tiempos, antes del inicio del método

de instanciado y otro cuando termina. Así, podemos obtener el tiempo que tarda el método en llevarse a cabo. Tras ejecutar el experimento hemos obtenido los siguientes resultados.



	1000	2500	5000	7500	10000	15000	20000	50000	75000	100000
Unity	0,312	0,719	1,797	3,378	5,094	10,02	16,49	124,5	204,4	515,6
NGUI	5,297	31,76	134,3	305,1	535,5	1260				

Figura 19. Tiempo de instanciado de número de elementos.

Podemos observar que ambos tienen un crecimiento exponencial. Mientras que NGUI posee una curva de saturación mucho más pronunciada. NGUI a partir de los 5000 hijos, su tiempo de creación es de más de 2' 30" y con 15000 asciende a 20'. Mientras que Unity no llega a este tiempo a partir de los ni con 100000 elementos.

Cabe destacar un resultado obtenido en la prueba. Utilizando Unity, realizando el test de los 2500 elementos, estos no se mostraban en el *scroll*. Esto es debido a que el *canvas* no soporta más de 65535 vértices dentro de él y deja de pintar la información en pantalla. Sin embargo, los componentes siguen estando presentes en la escena. Con NGUI esta casuística no ocurre.

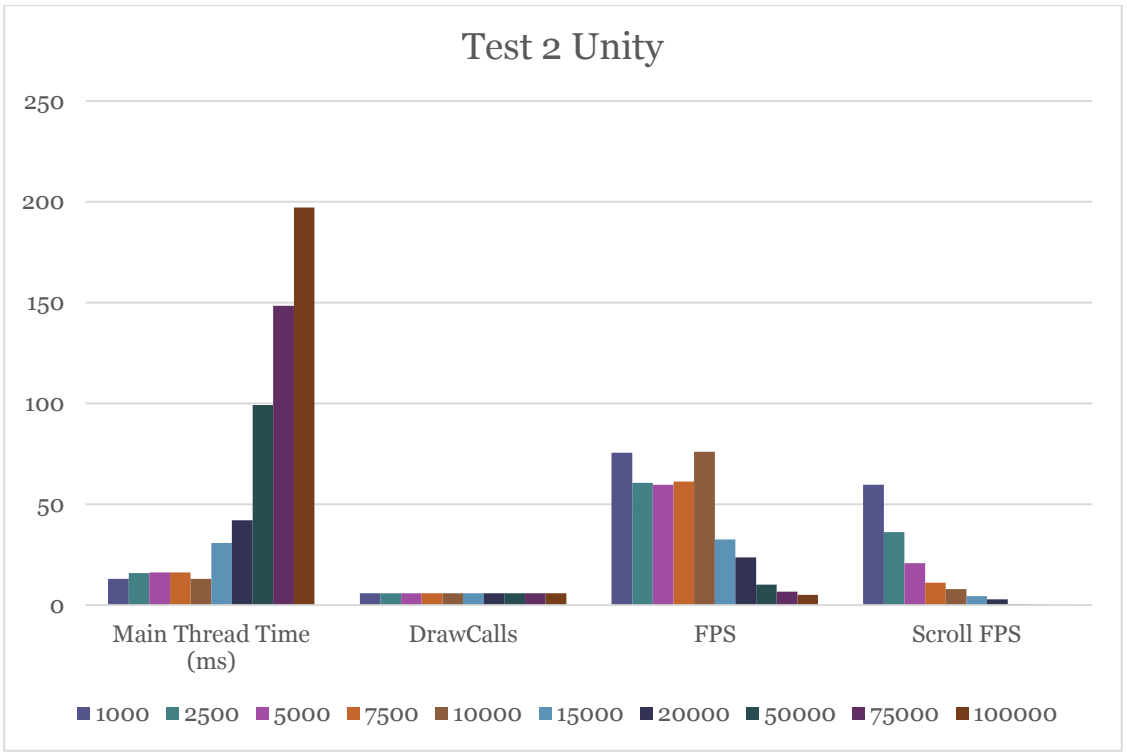


Figura 20. Resultados Test 2 Unity para distinto número de hijos.

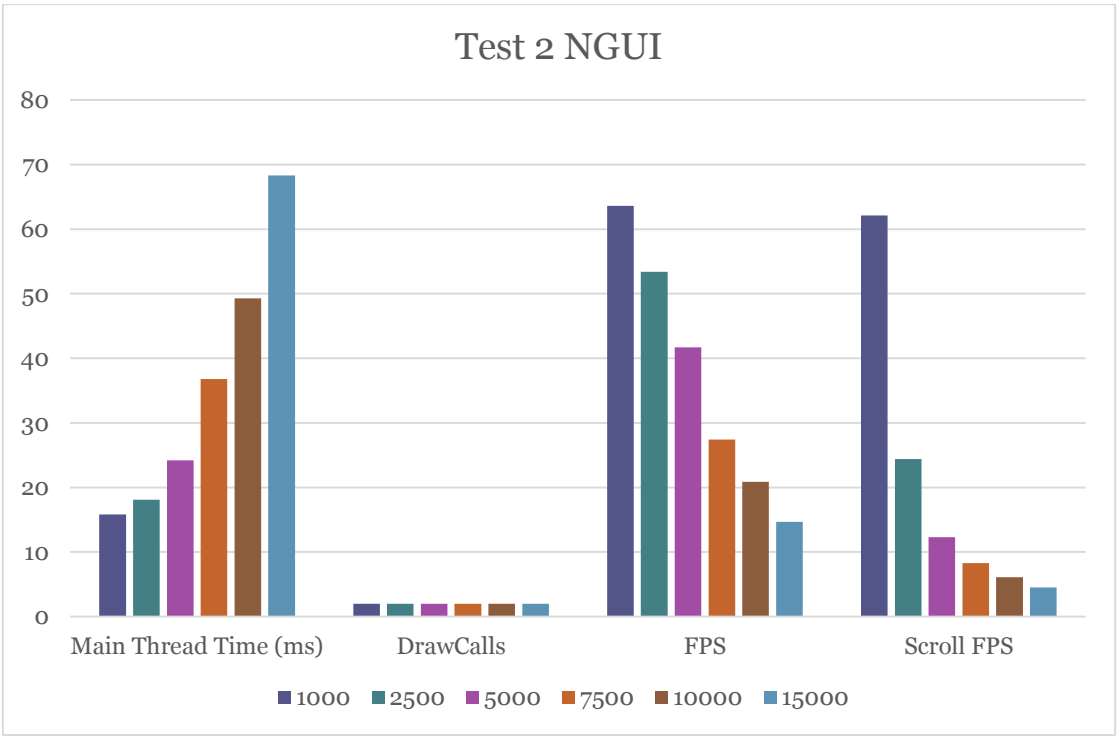


Figura 21. Resultados Test 2 NGUI para distinto número de hijos.

En las figuras 19 y 20 podemos observar los resultados del Test 2. En la primera gráfica, existe cierta estabilidad hasta los 10000 elementos con respecto al tiempo del hilo

principal y los *frames* del proyecto A partir de esta cifra, vemos una bajada gradual de *frames* y subida de tiempo del hilo con cuantos más elementos introducimos. Por otro lado, al hacer *scroll* la bajada es gradual desde la primera muestra, demostrando que se consumen recursos con este evento. Finalmente, las *Draw Calls* son constantes gracias a las técnicas anteriormente nombradas.

Al contrario que Unity, NGUI no tiene una cierta estabilidad hasta un determinado número de elementos. Estos van consumiendo más recursos en cuanto mayor número de elementos contiene. Mientras que el número de *Draw Calls* también se mantiene constante en este.

Cabe destacar que Unity posee un mayor número de llamadas a la API gráfica que NGUI. Esto se debe a que diversos componentes o algunas construcciones generan *Draw Calls* extras. Por ejemplo, el componente *Mask* altera las llamadas de gráfica dependiendo de los componentes de su interior. Por otro lado, el *overlaping* o superposición de componentes visuales de la interfaz también genera llamadas adicionales. Esto no está especificado en el manual de Unity sino que la comunidad ha dado con ello, así que provoca un mal uso o de bajo rendimiento por parte de los programadores. Una mejor documentación de Unity mejoraría el uso de esta. Por último, característica muy beneficiosa para ahorrar *Draw Calls* es añadir las fuentes al atlas como hace NGUI.

		Unity	**NGUI**
Coste		Gratuito	Pequeño (95 $)
Integración	Nativo	Si	No: infraestructura propia
	Órden renderizado	Jerárquico	Por profundidades
	Número de componentes	Menor (simplicidad)	Mayor (potencia)
	Utilidades (profiler, animator, etc)	Integración total	Módulos externos
Soporte	Oficial	En elaboración	Completa, poco profesional
	Comunidad	Bueno. Actualizaciones frecuentes	Bueno.
Rendimiento	Tiempo de instanciado	Cae por encima de 20000	Cae por encima de 2500
	FPS	Bueno	Bueno
	Drawcalls	5	2
	Nº de hijos para tiempo de respuesta aceptable (0.5s)	Sobre 1500 hijos	Menos de 1000 hijos

Tabla 1. Comparativa entre Unity y NGUI.

Finalmente, exponer que el *package* de Unity con todos los componentes, escenas y *scripts* utilizados para las pruebas están adjuntos a este trabajo para todo aquél que desee consultarlos.

4.3. Conclusiones

Expuestas las comparaciones en todos los aspectos, cabe destacar que ambas herramientas son muy buenas elecciones para el desarrollo de interfaz de usuario en nuestro proyecto. Primero, el precio puede ser un factor determinante. Aquel usuario que no desee una aplicación comercial o sea un iniciado en la programación de los videojuegos, debería escoger Unity como herramienta de desarrollo. Por otro lado, un proyecto comercial o una mediana o gran empresa, puede escoger el *plugin* NGUI sin ningún problema para la implementación de la interfaz. Segundo, si bien es cierto que la nueva interfaz está mejor integrada en el motor, NGUI suple perfectamente estas carencias con *scripts* y herramientas propias. No obstante, en el aspecto de facilidad de uso, Unity tiene más opciones. En tercer lugar, el tipo de funcionamiento de la interfaz de debería ser un factor determinante en la elección de la herramienta. Ya que solamente consiste en que el usuario se adecue a cada funcionamiento, como cambiar de lenguaje de programación pero conservando el mismo paradigma. Un aspecto que sí que podría ser determinante en cuanto a la elección de la herramienta, es el rendimiento. Pero como podemos ver en los resultados de los test, no existe gran diferencia en cuanto a interfaces convencionales. Tal vez, Unity precisa un tiempo para madurar y resolver ciertos problemas de rendimiento y completitud de componentes. Pero esto se puede suplir con la comunidad de usuarios que lo usan que comparten sus trabajos e ideas para poder mejorar. Finalmente, Unity como sucesora de NGUI tiene mucho camino por recorrer pero por ahora tiene ya mucho hecho y es una opción viable para el desarrollo de interfaces tanto para principiantes o proyectos personales como para expertos o proyectos comerciales.

5. Descripción del proyecto

En este apartado, vamos a describir el proyecto realizado mediante la nueva interfaz de Unity. Como se ha descrito al principio de este trabajo, el juego ha sido desarrollado por la empresa BraveZebra para la Liga de Fútbol Profesional. Al ser un juego multijugador online y tener toda la funcionalidad de juego en el servidor, es muy importante escoger bien las herramientas empleadas para el desarrollo del servidor, cliente y *middleware*. La selección de herramientas no es cosa de un día, ya que la calidad final del proyecto depende de la buena elección de las herramientas y de la buena interconexión de estas. Además, también hay que tener en cuenta el tiempo de aprendizaje de nuevas herramientas. Un recurso costoso no solamente temporal sino que también económico. Por ello, se ha optado por escoger herramientas que ya se han usado en otros proyectos en la empresa y que han dado buenos resultados.

5.1. Arquitectura

Para el diseño de la arquitectura del software, se ha optado por la estructura de cliente-servidor. Concretamente una arquitectura por capas de tres niveles. Esta arquitectura separa claramente la carga entre tres partes. La primera parte es la capa de presentación o interfaz de usuario. En esta capa, se agrupa todo el trabajo dedicado a la interacción del usuario con el software. Esta es la capa en la que está centrado este trabajo y que más se ha estudiado. Seguidamente encontramos la capa de negocio. Es la capa que se dedica a la recepción de datos por ambas partes, cliente y servidor, y procesa los datos para poder enviar los resultados a quienes lo precisen. Finalmente encontramos la capa de almacenamiento. Es aquella que se encarga de almacenar los datos y enviarlos a la capa de negocio. En ella también se ha incorporado funcionalidad de servidor ya que el juego precisa de un servidor que gestione todas las acciones ocurridas por los jugadores y poder ejecutar pequeñas tareas que dotan de jugabilidad al juego. Al ser tres capas tan diferenciadas e independientes entre sí, hablando en términos de desarrollo de código, se ha encargado un programador de cada capa para hacer más llevadero y resistente a fallos de comunicación.

5.2. Método de desarrollo

Se ha usado una metodología de desarrollo ágil, *Scrum*[7]. Consiste en planificar *sprints* o metas para un período de tiempo corto, en este caso entre una y dos semanas. En estas metas, se debe alcanzar una versión estable que permita entregar el producto cuando el cliente lo solicite. La metodología se divide en tres partes. La primera es la planificación del *sprint*. En ella, se seleccionan los requisitos que contendrá, se

dividirán en tareas y los desarrolladores estiman el tiempo dedicado a cada una. Las tareas son asignadas por cada uno de los miembros del equipo a sí mismos. La segunda parte es la ejecución, en la que los desarrolladores desarrollan y completan tareas. La tercera y última etapa es la revisión y adaptación. En ella al final del *sprint,* se realiza una revisión del trabajo y adaptación de estimación de tiempos en cada tarea.

Esta metodología se ha empleado por inercia de uso en otros proyectos anteriores. Además también es apropiada para proyectos que tienen cierta urgencia o, como el desarrollado por ejemplo, por reaccionar ante otros juegos de la misma temática o plazos de entrega fijos, ya que al estar ligado a la liga real, es necesario su lanzamiento antes del inicio de la liga.

Los roles de este proyecto son:

- ***Product Owner****:* Es el cliente que recibirá el producto. En este caso La Liga de Fútbol Profesional.
- **Administrador**: Es el que se encarga de administrar el entorno *scrum* del proyecto. En este caso Rubén Picó Ruíz, líder del proyecto.
- **Equipo de desarrollo**: Los desarrolladores del proyecto. Rafael Faulí Soria (capa de usuario), José Luís Martínez Pérez (lógica de negocio) y Sergio Canós Tevar (capa de datos).
- ***Stakeholders:*** Son aquellas personas que no influyen diariamente en el proyecto pero son necesarios. Como por ejemplo Mahou como patrocinador o Yeeply como intermediario.

5.3. Plataforma objetivo

El producto final se ha desarrollado para plataformas móviles con sistema operativo Android e iOS. Más adelante se implementará una versión web. Se han escogido como plataformas principales los móviles por la portabilidad y las características del juego. El juego está orientado a un tiempo de juego asíncrono. Esto quiere decir que el jugador ejecutará la aplicación y no interactuará en vivo con otros usuarios. La media de uso del juego seguido es de diez minutos. El usuario gestiona su equipo, ojea los jugadores del mercado y en caso de estar en más de una liga repite el proceso. Esto hace que el máximo tiempo de uso de la aplicación sea aproximadamente de quince minutos. A lo que suma, que el juego se regenta por un mercado con tiempo fijo, pero también tiene un componente atemporal donde otros usuarios escogen con que oferta quedarse por el jugador suyo en el mercado. Así, es necesario una utilización intermitente del software que es más apropiada de dispositivos móviles que de navegador web.

5.4. Lenguajes

Como se ha nombrado anteriormente, existen tres lenguajes de programación en Unity, Javascript, C# y Boo (este último está obsoleto a partir de la versión 5.x). Primeramente, se ha descartado Boo. Dos razones han sido las que han llevado a esta decisión. La primera es el poco uso de Python por los usuarios como lenguaje de programación. La segunda y decisiva, es el poco soporte por la comunidad en este lenguaje como hemos podido ver en la figura 3. Una vez descartado Boo, la decisión recae sobre los dos lenguajes más usados. Comparando la sintaxis de los lenguajes, apoyo de la comunidad y documentación, se decidió por C#. Su sintaxis es casi idéntica a Java, el cual se ha trabajado desde el inicio de carrera. La comunidad es muy amplia y está en continua ayuda a todos los usuarios. Finalmente, una gran documentación aportada por el lenguaje nativo hace que la decisión sea esta.

Por otro lado, no se ha podido escoger el mismo lenguaje para la persistencia. Se ha utilizado la plataforma Parse como desarrollo de servidor, la cual explicaremos más delante en detalle. La plataforma utiliza Javascript en su totalidad. Así que no hay más remedio que utilizar este lenguaje para la capa de datos.

5.5. Herramientas empleadas

La principal herramienta empleada en el proyecto es Unity. La elección de esta herramienta es simple por tres razones, coste económico, curva de aprendizaje y conocimientos previos. Unity siempre se ha relacionado con el bajo precio para un motor gráfico con muy buenas prestaciones y buen rendimiento. Por otro lado, el proyecto es un juego en dos dimensiones, por lo que los grandes motores como Unreal Engine y Cry Engine quedan totalmente descartados. Si a ello sumas una gran documentación y apoyo no solo en inglés sino en otras muchas lenguas, Unity se convierte en una opción muy viable para ser elegida. Finalmente, los conocimientos previos de la plataforma mediante la realización de otros proyectos y la posesión de su versión profesional, permitiendo el uso de características que la versión gratuita no tiene, hacen que la elección de Unity sea la más adecuada.

Al elegir Unity como motor de desarrollo, hace falta una pequeña pero no poco importante elección de desarrollo de código, el entorno de desarrollo de código. Existen dos grandes elecciones, MonoDevelop y Visual Studio. Visual Studio es un reconocido entorno de desarrollo interactivo, con mucho tiempo y con mucha y útil funcionalidad. No es nativo en Unity pero es compatible. Por otro lado, MonoDevelop es el IDE (Ver glosario) que Unity tiene por defecto con muchas funcionalidades. Por otro lado, también es conocido por tener algunos *bugs* y dificultar en pequeña medida al programador a la hora de hacer su trabajo. No obstante, se ha considerado que los

pocos errores que tiene MonoDevelop es más útil no perder el tiempo en la instalación y configuración de otro IDE, quedando el primero como elección para el uso.

Para el servidor se ha utilizado la plataforma Parse. Esta es una plataforma muy completa, la cual no solamente permite el almacenamiento de datos en una base de datos no relacional, sino que además ofrece la posibilidad de crear *cloud code*. Este código se puede llamar desde el cliente para que ejecute el código que previamente se ha programado. En el proyecto, esto se ha usado para calcular todo el código de juego que hace el servidor. Para este código el lenguaje de programación obligado es Javascript. Otra ventaja que ofrece Parse es que tiene una API para integrar con Unity. No solamente el *middleware* de interconexión con Unity está resuelto, sino que además ofrece otros servicios como administración de notificaciones *push* para usuarios de dispositivos móviles. Esto hace que sea mucho más atractivo que una base de datos SQL estándar y un servidor.

Anteriormente hemos nombrado la metodología de desarrollo ágil *Scrum*. Como ayuda para la realización de dicho método, se ha usado la herramienta Trello. Una web en la que es posible crear un tablero con listas y añadir tarjetas. Cada tarjeta hace referencia a una de las tareas creadas por los desarrolladores. Luego, cada usuario puede crear tarjetas y añadir componentes como usuario de la tarjeta, título, descripción, tiempo estimado de la tarea, tiempo real de la tarea e incluso añadir comentarios e imágenes para facilitar el entendimiento.

Otros componentes se han usado para implementar distintos aspectos del juego. Por ejemplo el guardado de datos en el dispositivo. Para ello se ha usado el *plugin* EasySave2. Un software que sirve de *wrapper* para la clase PlayerPrefs de Unity que además permite el guardado de objetos enteros, no solamente tipos de variables básicas. Otra funcionalidad suplida con nuevas herramientas ha sido las invitaciones vía email. Para ello se ha utilizado la plataforma Mandrill. Una plataforma que permite el envío de emails desde su API. Permite el envío de plantillas, mensajes en HTML y otras opciones útiles para este servicio.

6. Ampliaciones de Unity

Vistas las carencias de la novel interfaz de usuario, en la implementación de la interfaz de la aplicación LaLiga Fantasy desarrollada, surgió un problema que las herramientas nativas no podían solucionar.

6.1. Análisis

Durante el desarrollo de la interfaz de LaLiga Fantasy, era necesario la implementación de paneles para la interfaz principal del juego. Cada uno de estos paneles debería estar dentro de un objeto con un *scroll* para poder moverse entre estos de manera horizontal. Por desgracia, la nueva interfaz de Unity no permite nativamente que la vista del *scroll* se centre en uno de sus elementos. Por ello, se tuvo que implementar una nueva herramienta que permita suplir esta carencia.

6.2. Diseño

Para la resolución del problema propuesto, se siguió el modelo que usa NGUI para ello. Se diseñó un sistema basado en un padre, en este caso el *Scroll Rect,* y múltiples hijos. El resultado final se compone de 2+x *GameObjects*, donde necesitamos un elemento padre, que contiene el *scroll*, un elemento contenedor, que se encarga de ordenar y adaptar los elementos hijos al padre y finalmente 'x' es el número de elementos dentro del *scroll*. La jerarquía debe quedar como en la *figura 21*.

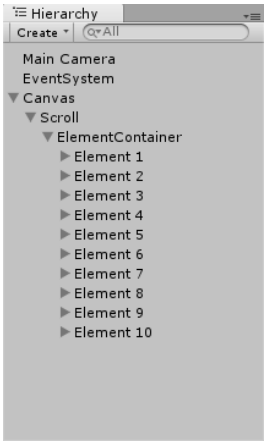


Figura 22. Orden de jerarquía del scroll.

Esta es la forma de crear elementos *Scroll Rect* más típica y común. Sin embargo, no podemos centrar en cada uno de los elementos. Para ello hace falta modificar el

elemento *Scroll Rect* nativo de Unity. Se ha creado una nueva clase que se ha nombrado *Scroll Rect Element* la cual hereda de *Scroll Rect*. En ella se añaden distintas características para el movimiento y centrado en los elementos. Se han añadido un valor booleano para escoger si se desea la transición con animación o sin. También podemos escoger el tiempo de la animación en caso de que haya. Finalmente encontramos la referencia de una máscara. Esta es necesaria para centrar el elemento donde se desea.

Además de lo anteriormente nombrado, es necesario otro *script* para que funcione todo correctamente, *CenterChild*. Este elemento se debe añadir a cada hijo del *scroll*. Sirve para que el padre sepa la posición del hijo y mueva este mediante el tipo de transición escogida a la posición central.

6.3. Implementación

Estos nuevos *scripts* se han implementado en C# ya que es el lenguaje con el cual se ha creado toda la aplicación. Los *scripts* nombrados anteriormente están adjuntos a este trabajo para todo el que desee pueda leerlo.

7. Conclusiones y trabajo futuro

Como conclusión de este trabajo podemos afirmar que se han explicado el funcionamiento de NGUI tanto en funcionamiento como en componentes incorporados, así como también Unity. Se han dispuesto una serie de parámetros de evaluación entre ambas que son precio, integración, soporte y rendimiento. Además, se ha elaborado una comparación en base a estos parámetros para determinar carencias y puntos fuertes de ambas. Finalmente como futuras ampliaciones podemos utilizar esta misma metodología de comparación para comparar otras herramientas. Como por ejemplo otros *plugins* existentes en el mercado o las mismas herramientas nativas de las plataformas para las cuales se va a desarrollar.

Finalmente, gracias a LaLiga Fantasy he podido aprender a fondo el uso de la nueva interfaz de Unity sin previo conocimiento de la programación de interfaces. Este proyecto, me ha permitido aprender también NGUI, predecesor de Unity. Además, no ha sido un camino de rosas el desarrollo de la aplicación. He recordado todo aquel material que en la carrera parecía tener poca utilidad sin contexto. Sin embargo, asignaturas como ingeniería del software, bases de datos, introducción a la programación de videojuegos o estructuras de datos y algoritmos han sido de gran utilidad para el desarrollo e implementación.

8. Términos y vocabulario

Anchor: Punto o componente relativo a cual un elemento mantendrá su posición o escala.

API: De las siglas en inglés *Application Programming Interface*, es el conjunto de métodos, clases y funciones que proporciona una biblioteca para ser utilizados por otro software.

Asset: Todo elemento del proyecto.

Callback: Función llamada cuando ocurre un determinado evento.

DPI: De las siglas en inglés Dots Per Inch. También de utiliza para medir fotograma píxeles por pulgadaen una pantalla. Es una resolución de medida de densidad de píxeles en una pulgada de pantalla.

Drag and Drop: En iglés arrastrar y soltar. Acción utilizada comúnmente en interfaces.

Frame: Fotograma. Elementos pintados en la pantalla durante un.

Game Object: Elemento básico que Unity añade a la escena. Alberga en él componentes del motor.

IDE: *Integrated Development Enviorment*. Software que permite al programador utilidades para el desarrollo de código. Suele consistir en un editor de software, compilador y depurador.

Monobehaviour: Clase de Unity de la cual deben heredar todos aquellos *scripts* que quieren utilizar las funciones y *callbacks* que el motor proporciona

Pivot: Pivote de un elemento sobre el cual se aplicarán las transformaciones de este.

Prefab: Game Object previamente creado y almacenado para poder usar cuando precise.

Renderizado: Pintado en la pantalla del dispositivo.

UV: Coordenadas en dos dimensiones resultantes del mapeado de un elemento en tres dimensiones a una textura en dos dimensiones.

9. Bibliografía

[1] Télam, agencia nacional de noticias (2014). “En 2015, los ingresos por juegos para móviles superarán a los de consolas” en Argentina, 2014.

<http://www.telam.com.ar/notas/201410/82755-videojuegos-apple-android-ganancias-consolas.html> [Consulta: 28 de Julio de 2015]

[2] Newzoo, Gamers market research (2014). “Global Mobile Games Revenues to Reach $25 Billion in 2014”.

<http://www.newzoo.com/insights/global-mobile-games-revenues-top-25-billion-2014/> [Consultado el 30 de Julio de 2015]

[3] Francisco Carrasco (2014). “El ocio móvil generará 54,000 millones de dólares en 2015”.

<http://www.pcworldenespanol.com/2014/02/10/el-ocio-movil-generara-54-000-millones-de-dolares-en-2015/> [Consultado el 29 de Julio de 2015]

[4] Aleksandr (2014). “Documentation, Unity scripting languages and you”.

<http://blogs.unity3d.com/2014/09/03/documentation-unity-scripting-languages-and-you/> [Consultado el 3 de Agosto de 2015]

[5] Unity (2014). “Execution order of event functions”.

<http://docs.unity3d.com/es/current/Manual/ExecutionOrder.html>
[Consultado el 3 de Agosto de 2015]

[6] Unity (2014). “GUI Basics (Legacy)”.

<http://docs.unity3d.com/Manual/gui-Basics.html>
[Consultado el 4 de Agosto de 2015]

[7] ProyectosAgiles.org (2015). “Qué es SCRUM”.

<http://www.proyectosagiles.org/que-es-scrum>
[Consultado el 22 de Agosto de 2015]

[8] Unity (2015). “Unity Manual”.

<http://docs.unity3d.com/Manual/index.html>

[Consultado el 20 de Agosto de 2015]

[9] Tasharen Entertainment (2014). “NGUI Overview”.

<http://www.tasharen.com/forum/index.php?topic=6754>

[Consultado el 24 de Agosto de 2015]