Contacto: *blartenixstudio@gmail.com*
Blartenix ®

Blartenix Common

namespace: Blartenix

Versión: *v2.0*

Tabla de Contenido

Contacto: *blartenixstudio@gmail.com*
Blartenix ®

Introducción

Paquete básico y esencial de la Librería de Blartenix para Unity.

Este documento además, es una guía de referencia de los scripts que hacen parte de este paquete, enfocada al usuario desarrollador final por lo que se muestra la documentación de solo los miembros públicos de su interés (clases, propiedades, métodos y componentes) con el fin de que comprenda los elementos que están contenidos dentro de este paquete y así le pueda dar una mejor idea a la hora de diseñar sus propios proyectos.

NOTA IMPORTANTE:
La versión 2.x no es retro-compatible con versiones 1.x del paquete. Se deben eliminar todas las referencias a archivos de las anteriores versiones del paquete para evitar inconvenientes.
Se recomienda empezar a utilizar desde un nuevo proyecto.

Contenido

Folder: *Assets/Blartenix/Common*

Listado de elementos que contiene este paquete.

Documentation

Folder: *Assets/Blartenix/Common/Documentation*

Se encuentra la documentación del paquete. Documentos en PDF conocidos como Guia de Referencia del Módulo (MRG), en idiomas Ingles y Español.

Prototyping

Folder: *Assets/Blartenix/Common/Prototyping*

Este folder contiene elementos que se pueden utilizar para un rápido prototipado de mecánicas que se quieren hacer a la hora de diseñar videojuegos.

- ➢ **Game Settings**

  Contiene todos los elementos requeridos por el menú de interfaz de usuario para controlar las configuraciones de juego básicas incluidas en este paquete (GameSettings). Para utilizar este menú, solo debe arrastrar el prefab de Game

Settings Canvas a su escena (Tenga en cuenta que debe existir una instancia de Event System en la escena)

Puede usar el contenido de este contenido para crear su propio menú de configuraciones.

➢ Pictograms

Contiene un prefab que puede ser usado para los componentes TriggerPictogram y TriggerPictogram2D. Puede usar este prefab como referencia para construir sus propios pictogramas.

Scripts

Folder: *Assets/Blartenix/Common/Scripts*

Components

➢ Billboard

Permite que un GameObject siempre mire hacia un punto objetivo.

Campos serializados

- ➔ bool **Look At Main Camera**
  Si el punto objetivo es la cámara con el tag MainCamera.

- ➔ string **Look At Target**
  Si el punto objetivo no es la cámara principal, se configura otro Transform objetivo.

- ➔ Transform **Look Position Offset**
  Diferencia o delta de la posición a mirar con respecto al punto objetivo.

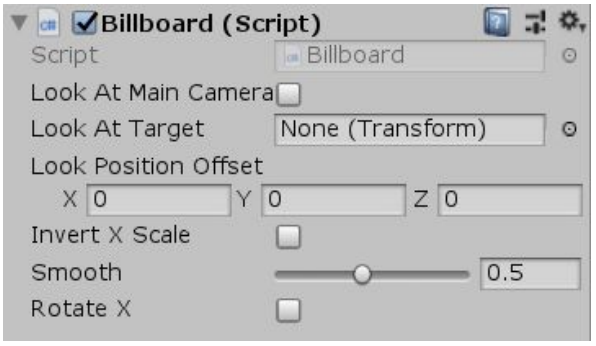
- ➔ Transform **Invert X Scale**
  If the X factor of the scale needs to be inverted.

- ➔ float **Smooth**
  Rango entre 0.1 y 1. Define la velocidad de rotación para conservar la dirección hacia el objetivo.

- ➔ bool **Rotate X**
  Si el Objeto debe rotar también en el eje X.

Métodos

➔ void **SetTarget**(Transform target)
Establece el objetivo.



Inspector del componente.

➢ CanvasEnabler

Para habilitar o deshabilitar canvases y sus elementos de forma rápida.

Campos Serializados

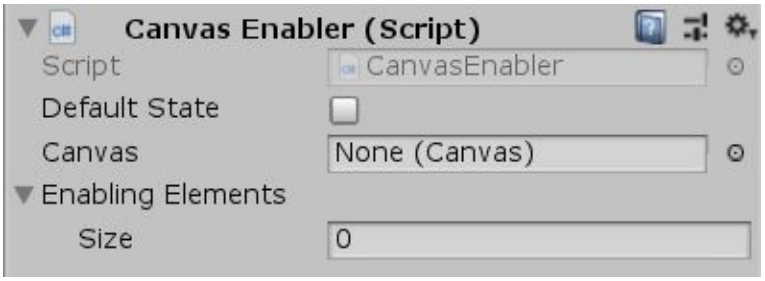
➔ bool **Default State**
Estado inicial.

➔ Canvas **Canvas**
Componente Canvas. Si no se especifica uno, en Awake se captura el componente

➔ Behaviour[ ] **Enabling Elements**
Los elementos internos del canvas que se habilitan y deshabilitan cuando se cambia el estado. (Botones, imágenes, etc)

Propiedades

➔ bool **enabled** { get; set; }
Habilitar o deshabilitar el Canvas junto a los Enabling Elements.



Inspector del componente.

➢ TriggerPictogram

Permite mostrar un pictograma cuando se ejecuta el método OnTriggerEnter y/o cuando usted así lo desee.

Campos Serializados

➔ CanvasEnabler[ ] **Pictograms**
Instancia de los pictogramas en world space de la escena.

➔ int **Default Pictogram**
Índice del pictograma por defecto en la lista de pictogramas anterior.

➔ string[ ] **Triggering Tags**
Listado de Tags para detectar y filtrar GameObjects en el OnTriggerEnter.

➔ LookForTagAt **Look For Tag At**
El GameObject a verificar si cumple con los triggering tags, asociado al Collider que ejecuta el Trigger.

➔ bool **Look At Camera**
Si el pictograma debe mirar siempre hacia una cámara.

➔ bool **Look At Main Camera**
Si debe mirar a la cámara principal (con el tag Main Camera). Solo se muestra si el valor de la propiedad anterior es verdadero.

➔ Camera **Target Cam**
La cámara objetivo a la que el pictograma de mirar. Solo se muestra si el valor de la propiedad anterior es falso.

Propiedades

➔ bool **IsShowing** { get; private set; }
Si se está mostrando un pictograma actualmente.

Métodos

➔ void **HidePictogram**()
Oculta el pictograma que se está mostrando actualmente.

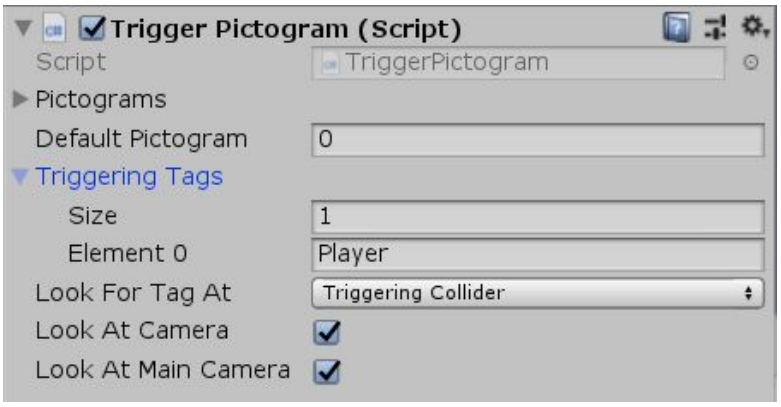
➔ void **ShowPictogram**()
Muestra el pictograma predeterminado.

➔ void **ShowPictogram**(int pictogramIndex)
Muestra el pictograma correspondiente al índice de la lista de pictogramas, especificado en el parámetro.

➔ void **ShowPictogram**(int pictogramIndex, float hideTime)
Muestra el pictograma correspondiente al índice especificado en el parámetro y lo oculta automáticamente luego de *'hideTime'* segundos.

➔ void **SetDefaultPictogram**(int pictogramIndex)
Establece el pictograma por defecto.

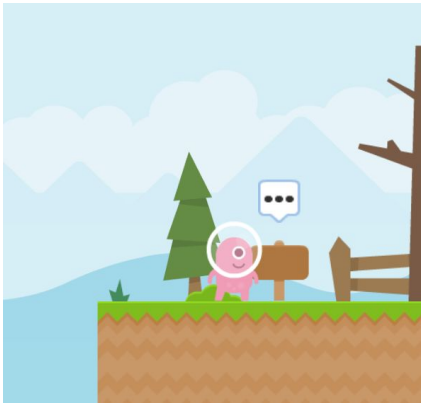
Pictograma entorno 3D.



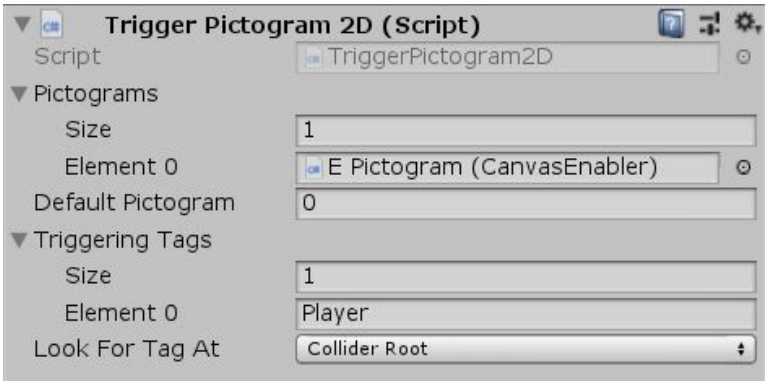
Inspector del componente.

➢ TriggerPictogram2D

Corresponde al componente *TriggerPictogram* con la diferencia que este es para el entorno 2D, es decir, para cuando se ejecuta el OnTriggerEnter2D. Tiene los mismos elementos serializados, propiedades y métodos.



Pictograma entorno 2D.



Inspector del componente.

➢ SingletonBehaviour<T>

Componente abstracto genérico para la implementación de otros elementos que se manejen bajo el patrón de diseño Singleton.

Campos serializados

➔ bool **Dont Destroy On Load**

Propiedades

➔ static T **Instance** { get; private set; }
Instancia estática del tipo del componente especificado.

Métodos

➔ protected virtual void **OnAwake**( )
Método que reemplaza al Awake en el componente que hereda de éste y que debe sobreescribir si desea hacer uso del método Awake.
No se debe usar el Awake

➔ protected virtual void **OnDestroyed**( )
Método que reemplaza al OnDestroy en el componente que hereda de éste y que debe sobreescribir si desea hacer uso del método OnDestroy.
No se debe usar el OnDestroy

➢ LanguageManager

Componente encargado de administrar y manejar los idiomas del juego. Usa el patron singleton y hereda del SingletonBehaviour<LanguageManager>

Campos Serializados

➔ bool **Dont Destroy On Load** (*SingletonBehaviour*)

➔ GameSettings **Game Settings**
ScriptableObject de las settings del juego. *Es opcional. Se usa para mantener persistente la selección del lenguaje actual.*

➔ List<TextAsset> **Language Files**
Listado de los archivos xml de idiomas para el juego.

➔ int **Selected Language**
Índice del idioma actual de la lista archivos.

Propiedades

➔ static LanguageManager **Instance** { get; private set; } (*SingletonBehaviour*)

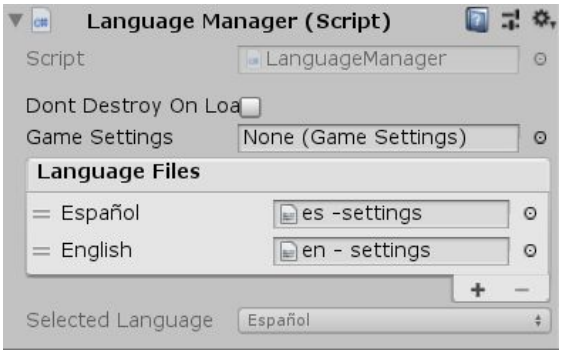
➔ int **SelectedLanguage** { get; }
Índice del idioma actual. Solo lectura.

Métodos

➔ void **SetLanguage**(int languageIndex)
Change the selected language by sending the index of the new language.

➔ IList<string> **GetLanguagesNames**( )
Retorna el listado de los nombres de los lenguajes.

➔ string **GetText**(string idName)
Retorna el valor del texto con el idName especificado para el lenguaje seleccionado actualmente.



Inspector del componente

➢ LanguageText

El componente para los textos del juego que implementan el sistema de diálogos.

Campos Serializados

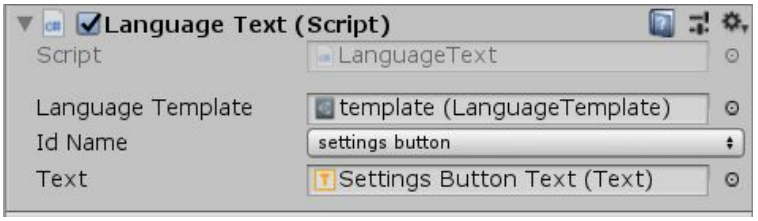
➔ LanguageTemplate **Language Template**
Template que contiene el identificador del texto.

➔ string **Id Name**
Nombre identificador del texto

➔ Text **Text**
Componente tipo Text.

Propiedades

➔ string **Text** { get; }
Texto o contenido del componente Text asociado.



Inspector del componente

➢ ObjectPool<T>

Componente abstracto base para la implementación de object pooling. El tipo T deben ser de tipo Component.

Campos serializados

➔ T **Prefab**
The prefab.

➔ int **Initial Count**
The amount of instances in the pool when initialized.

➔ T[ ] **Preloaded**
Active or inactive instances in the scene to help or reduce pool initialization.

Métodos

➔ void **InitPool**( )
Initializes the pool. should be called in the Awake method.

➔ T **Get**(bool active = true)
Return an instance. by default active.

➔ T **Get**(Transform parent, bool active = true)
Return an instance as a child of transform specified. if null, with no parent.

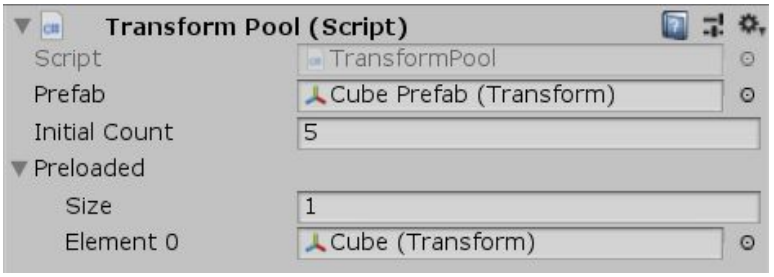
➔ T **Get**(Vector3 position, Quaternion rotation, bool active = true)
Returns an instance in the position with the rotation specified.

➔ T **Get**(Vector3 position, Quaternion rotation, Transform parent, bool active = true)
Returns an instance with the position, rotation and parent transform specified.

➔ void **Put**(T go)
Finalizes an instance and returns it to the pool.

➢ TransformPool

Es la implementación de un pool para objetos de tipo transform. Hereda de ObjectPool<Transform> por lo que sus campos serializados y métodos son los definidos en ObjectPool<T> y T (el tipo) es Transform.



Inspector del componente.

➢ MultipleObjectsPool<T>

Componente abstracto básico para la implementación de pool de objetos del mismo tipo con características diferentes entre sí. El tipo T debe ser de tipo Component.

Campos serializados

➔ T[ ] **Prefabs**
The prefabs of the game objects for the pool

➔ T[ ] **Preloaded**
Active or inactive instances in the scene to be preloaded to help performance.

Métodos

➔ void **InitPool**( )
Initializes the pool. should be called in the Awake method.

➔ T **Get**(Predicate<T> predicate, bool active = true)
Find and return an instance based on a predicate query. By default active. if no instance is found returns null.

➔ T **Get**(Predicate<T> predicate, Transform parent, bool active = true)
Find and return an instance as a child of the transform specified. if null, then with no parent. return null if an instance is not found.

➔ T **Get**(Predicate<T> predicate, Vector3 position, Quaternion rotation, bool active = true)
Find and return an instance with the position and rotation specified.

➔ T **Get**(Predicate<T> predicate, Vector3 position, Quaternion rotation, Transform parent, bool active = true)
Find and return an instance with the position, rotation and parent game object specified.

➔ void **Put**(T go)
Finalizes an instance and returns it to the pool.

Core

Scriptable Objects

➢ **GameSettings**

Objeto para el manejo de algunas configuraciones básicas para tu juego. Se mantiene persistente en las PlayerPrefs para el juego.

Create → Blartenix → Game Settings

Campos serializados

➔ AudioMixer **Game Audio Mixer**
Audio Mixer del juego

➔ string **Music Volume Param Name**
Nombre del parámetro expuesto en el audio mixer para el manejo del volumen de la música.

➔ string **Sfx Volume Param Name**
Nombre del parámetro expuesto en el audio mixer para el manejo del volumen de los efectos de sonido.

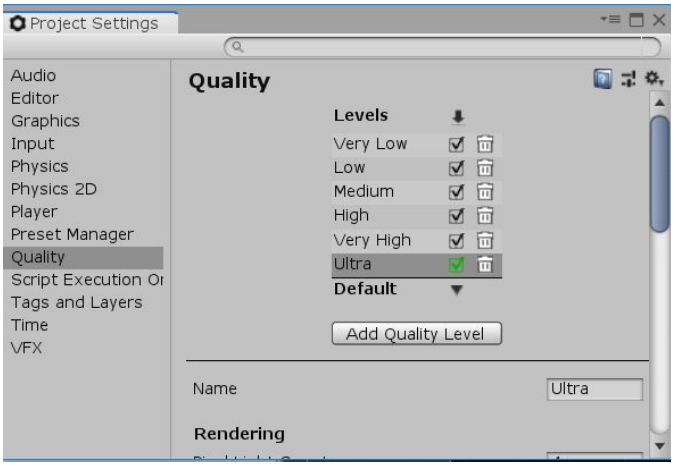
➔ float **Default Music Volume**
Volumen por defecto de la música.

➔ float **Default SFX Volume**
Volumen por defecto de los efectos de sonido.

Propiedades

➔ int **Language** { get; set; }
Índice del lenguaje seleccionado actualmente.

➔ int **Graphics** { get; set; }
Índice de la calidad de gráficos seleccionado actualmente. Este listado está definido en las Configuraciones del proyecto.



➔ int **Resolution** {get; set; }
Índice de la resolución actual. Este listado se obtiene de las resoluciones soportadas por la pantalla actual.

➔ bool **Fullscreen** { get; set; }
Modo pantalla completa.

➔ float **MusicVolume** { get; set; }
Volumen de la música.

➔ float **SfxVolume** { get; set; }
Volumen de los efectos de sonido.

Métodos

➔ void **ResetValues**( )
Borra las player pref. restableciendo los valores por defecto para las configuraciones.

Inspector

➢ **LanguageTemplate**

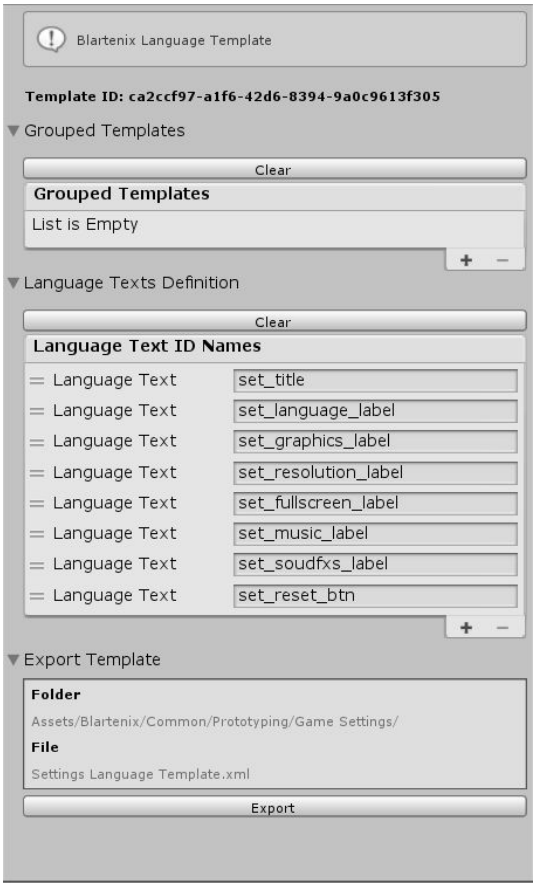
Objeto que contiene la definición de todos los textos de un idioma y que además permite generar los archivos xml de los lenguajes usados por el LanguageManager.

Create → Blartenix → Language Template

Campos serializados

➔ LanguageTemplate[ ] **Grouped Templates**
Listado con otras plantillas agrupadas para asociar sus identificadores de los textos.

➔ string[ ] **Language Text Id Names**
Listado de los identificadores para los textos de la plantilla.



Inspector del language template.

Clases

➢ *BlartenixLogger*

Clase estática que contiene la instancia global del logger de la librería de Blartenix para Unity.

Propiedades

➔ static IBlartenixLogger **GlobalInstance** { get; set; }
Instancia estática global del logger de Blartenix.

➢ *Utilities*

Clase estática con métodos utilitarios usados en la Librería de Blartenix para Unity.

Métodos

➔ string **EncodeString(**string decodedString**)**
Codifica un string en su representación base64.

➔ string **DecodeString(**string encodedString**)**
Decodifica un string en base64 a un string.

➔ T **DeserializeXML<T>(**string xml, bool xmlIsAFile**)**
Deserializa un XML a su representación de datos estructurados. Recibe un xml como string o como archivo especificado en el parámetro *xmlIsAFile.*

➢ ***BlartenixLanguage***

Definición estructurada de un idioma.

Campos serializados

➔ string **Template ID**
identificador del template asociado al idioma.

➔ string **Name**
Nombre del idioma.

➔ List<LanguageXmlTag> **Language Texts**
Lista de los textos definidos para el idioma.

➢ ***LanguageXmlTag***

Definición de la etiqueta xml para textos de idiomas.

Campos serializados

➔ string **ID Name**
Nombre identificador del texto.

➔ string **Value**
Valor del texto.

Enums

➢ ***LogType***

Tipo de mensaje de log de Blartenix.

Tipos

➔ *Info*
➔ *Warning*
➔ *Error*

➢ ***LookForTagAt***

Usado para la interacción con los métodos OnTrigger de Unity para indicar donde buscar el tag especificado.

Tipos

- ➔ *TriggeringCollider*
- ➔ *ColliderParent*
- ➔ *ColliderRoot*

Interfaces

➢ ***IBlartenixLogger***

Interfaz que define la clase de logging para Blartenix.

Métodos

➔ void **DisplayMessage**(string message, LogType type = LogType.Info)
Método para mostrar mensajes de logs. Pensado para registrar mensajes de logs durante la etapa de desarrollo o directamente en la aplicación.

➔ **Log**(string message, LogType type = LogType.Info, [*CallerMemberName*] string callMember = null, [*CallerFilePath*] string file = null, [*CallerLineNumber*] int codeLine = -1)
Método para registrar un mensaje de log. Pensado principalmente para registrar mensajes de logs en archivos externos en tiempo de ejecución.

Los parámetros *callMember*, *file* y *codeLine* son enviados automáticamente cuando se invoca el método, por tal razón son opcionales.

Referencias

➔ Microsoft docs. CallerFilePathAttribute

➔ Microsoft docs. CallerLineNumberAttribute

➔ Microsoft docs. CallerMemberNameAttribute

➔ Microsoft docs. Predicate<T>

➔ Tutoriales en nuestro canal de Youtube. Puedes encontrar videos y listas de reproducción con demostraciones.

¡Gracias por el apoyo!