

# Technical Test

La prueba se tiene que realizar en 24 hs, la entrega sólo incluye el proyecto completo de unity y un pdf con la justificación que corresponda.

Se adjunta en la misma folder de drive que este archivo, un package de unity (para 2020.3 o superior) con lo necesario para cada ejercicio.

- 1) Se necesita desarrollar un sistema llamado ScreenNavigator que controla la navegación entre las screens del juego, por ahora estas son Main Menu, Inventory y Shop, pero es muy probable que en el juego final haya más.

## Reglas:

- El ScreenNavigator conoce la Screen actual y puede activarla y desactivarla.
- Este setea la screen actual y se encarga de hacer lo necesario para mostrarla.
- Cada vez que se cambia la screen debe printear cuál es la actual.
- ScreenNavigator se hereda de MonoBehaviour (que es para poder instanciarlo en la escena) pero **no conoce ninguna clase de Unity**. *Aclaración: esto es solo para el ScreenNavigator.*
- Para poder resolver el ejercicio se puede añadir cualquier otro script que le sirva al sistema.

## Se necesita:

- 1) Arreglar la UI para que se adapte al formato 16:9 y que también sea responsive a cualquier resolución horizontal.
- 2) Crear el sistema de navegación de screens.
- 3) Añadir Shop Screen y que se pueda navegar de MainMenu a Shop y viceversa
- 4) Añadir Inventory screen, que se pueda navegar de Mainmenu a inventory y de shop a inventory, pero desde inventory solo se puede volver a la screen anterior.

- 2) Dado el siguiente código se tiene que refactorizar aplicando los estándares dados, respetando los principios SOLID y aplicando buenas prácticas de programación. Se puede renombrar, borrar y crear clases e interfaces según sea necesario, el código no va a ser ejecutado por lo tanto no hay que aplicarlo en ninguna escena, solo es refactorizar.

### Standards

- Todo debe estar escrito en Inglés.
  - No deben existir variables públicas, en caso de ser necesario usar una property.
  - Las variables privadas llevan el prefijo `_`, ejemplo: `_player`.
  - Respetar el orden jerárquico:
    1. Variables públicas.
    2. Variables privadas.
    3. Properties.
    4. Métodos.
  - Todas las clases e interfaces están contenidas en un namespace que identifique y exprese bien de qué conjunto se trata .
  - Los nombres de las clases, namespaces, properties están escritos en CamelCase.
  - Las interfaces comienzan con `I`.
  - Todas las clases e interfaces deben estar en su propio archivo.
  - Quitar Namespaces innecesarios.
- 3) Con alguna Clase consumir la Clase `LifeLocalRepository.cs` y mostrar en pantalla su valor. Después añadir un nuevo `LocalRepository<string>` que contenga el nombre del jugador y también mostrarlo en pantalla.

- 4) Observar el siguiente código que funciona perfecto, proponer una solución mejor y enumerar qué problemas tiene este código en un proyecto a largo plazo, no es necesario codear nada pero sí justificar el por qué.

El siguiente código elige según el tipo de Cloth que prefab le tiene que devolver.

```
public class CharacterCustomization : MonoBehaviour
{
    [Header("References")]
    [SerializeField] private GameObject[] _clothes;

    public GameObject GetClothPrefab(Cloth cloth)
    {
        string clothId = cloth.Id;
        GameObject clothPrefab = null;

        switch (clothId)
        {
            case "Shirt":
                clothPrefab = _clothes[0];
                break;

            case "Bag":
                clothPrefab = _clothes[1];
                break;

            case "Shoes":
                clothPrefab = _clothes[2];
                break;

            case "Hair":
                clothPrefab = _clothes[3];
                break;

            case "Polemico":
                clothPrefab = _clothes[4];
                break;

            case "Hoodie":
                clothPrefab = _clothes[5];
                break;
        }

        return clothPrefab;
    }
}
```

- 5) Compilar el juego para pc, mac o linux sin ningún error.