0. Información

Carpetas:

- GuessTheNumberTest - Contiene el proyecto de Unity.
- GuessTheNumber_Readme_Nico - La descripción del proyecto.

Controles:

- Click izquierdo para seleccionar las opciones.

Escena:

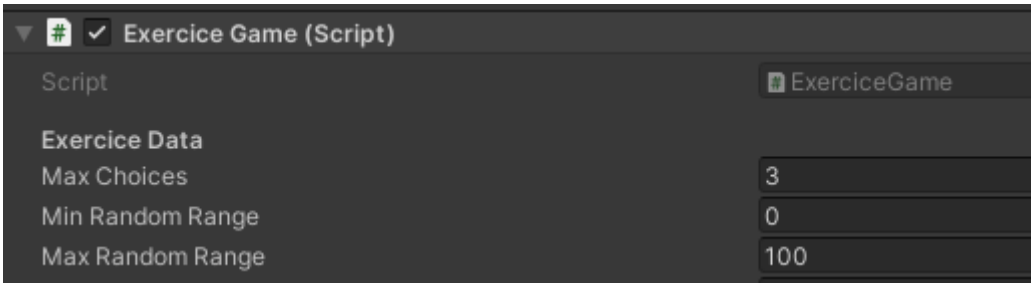
- Scenes/ExerciceGame.unity

1. Decisiones tomadas

He creado la clase **ExerciceGenerator** que no hereda de MonoBehaviour y que lo que hace es generar un objeto de la clase **Exercice** que tiene:

- El número correcto de este enunciado.
- La lista de opciones de este enunciado.
- El índice en el que está el número correcto del enunciado después de desordenar la lista de opciones.

Para poder ejecutar esta generación del ejercicio se tiene que pasar como argumento el número de opciones que queremos y el rango de números posibles. Así hacemos más escalable el ejercicio permitiendo tener las opciones que queramos en el rango de números que prefiramos. En la escena las opciones se colocan en un GridLayout dentro de un ScrollView para poner el número de opciones que queramos.



La clase **TextResourcesManager** se utiliza para transformar el número seleccionado en cualquier enunciado (p. ej. 35) en un número textual (p. ej. treinta y cinco). Contiene una función que va fraccionando y analizando el número mientras lo va transformando en una string.

He decidido parsear un JSON para coger diferentes unidades numéricas (unidades, centenas, decenas) textuales para que pudiese ser traducido en diferentes idiomas. Cabe decir que el lenguaje español tiene algunas excepciones que en otros no.

El JSON esta en: Assets/Resources/Text/TextResources

Los números seleccionables son el objeto **NumberOption** que tiene la funcionalidad de:

- Enseña el número y permite que lo elijan.
- Permite cambiar el color del número en función de la respuesta.
- Contiene la transición de enseñar y esconder la opción (también se encarga de permitir interactuar con el botón)

Estas opciones son guardadas en la clase **OptionsController** que sirve como padre de todas las opciones, las inicializa, les da el valor en cada enunciado y las enseña/esconde. También contiene las posibles reacciones en función de la elección del jugador en cada enunciado.

Al escoger una opción se invoca al objeto **ExerciceAnswer** que tiene una lista de **NumberExercice** donde se almacena por cada número, el número en sí, los intentos y un contador de errores que se reinicia al llegar a 2 (por los dos tipos de reacción a las respuestas).
Este objeto ExerciceAnswer tiene un array de **WrongAnswers**, una clase abstracta que se guarda en qué número de error tiene que hacer su funcionalidad. De esta clase heredan dos tipos de respuestas:

- cambiar el color del número erróneo
- cambiar el erróneo y el correcto también.

Con esto conseguimos poder tener diferentes tipos de respuesta de manera escalable.

AnswersCounter se suscribe a los eventos de *onCorrectAnswer* y *onWrongAnswer* del ExerciceAnswer para ir aumentando los contadores de la UI. Lo he hecho de esta manera porque estos eventos quizá en un futuro podrían ser útiles para más objetos.

Todo lo mencionado anteriormente se encarga de hacer la lógica del ejercicio.

ExerciceGame es por donde se inicia el flujo del proyecto, pero solo se encarga de crear el bucle infinito de enunciados utilizando los elementos mencionados anteriormente.

He añadido unos métodos extensions como GetValueOrDefault, Shuffle, DoAlphaTransition para diferentes tipos de variables y hacer más fácil algunas funcionalidades o control de errores.

2. Mejoras

- Clase Answers de donde hereden tanto aciertos como errores para poder responder de diferente manera en ambos casos en diferente número de intentos.
- Más variedad de animaciones en las respuestas.
- Guardar en un JSON/PlayerPref las respuestas del usuario para que cuando vuelva a entrar pueda ver su contador general.
- Cargar el JSON de texto mediante Addressables y no Resources.