**Random Number Test Document**

En este documento procederé a explicar la forma en la que he intentado estructurar mi proyecto.

En primer lugar hay un script que controla el flujo general del juego llamado MainFlow, este script se encarga del flujo principal del juego y cuenta con 3 estados:

* MainMenu
* Playing
* Paused

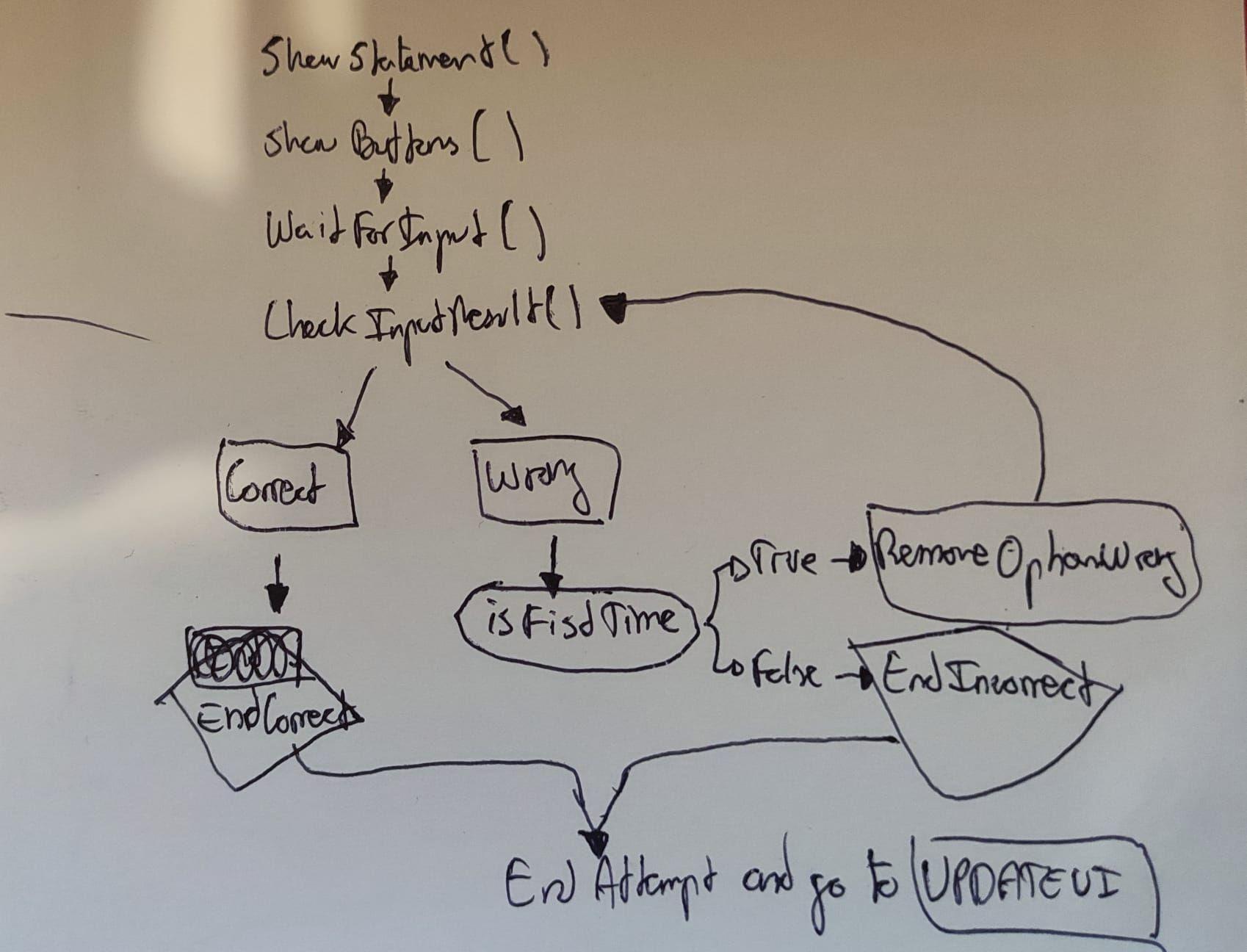
Para pasar de un estado a otro escucha los eventos lanzados al clickar los diferentes botones de la MainUIView (esta view engloba los botones y textos principales del juego

), y actúa en consecuencia cambiando el estado del juego lo que sirve para modificar su comportamiento, en esta prueba simple, los diferentes estados no aportan grandes cambios pero a medida que el proyecto escalase esta decisión facilita el mantener aislados los diferentes estados del juego para evitar inconvenientes. Este script también se encarga de inicializar el resto de controllers los cuales regentan sus correspondientes views.

A raíz de esta mención a los controllers y views, he intentado implementar el patrón de diseño MVC el cual se basa en diferenciar la parte visual y sus componentes/acciones en views, y la lógica que pueden seguir dichas views dependiendo de los datos recibidos(modelo) en controllers. En este caso el modelo es algo difuso ya que no se trabaja con una gran cantidad de datos, simplemente una lista de 10 números de los cuales se eligen 3 aleatorios y uno correcto para cada intento. Pero los controladores y views sí están bastante definidos, las views se encargan de desactivar/activar los diferentes componentes visuales, de animarlos(FadeIn/FadeOut), de modificar los textos, tamaños, colores, etc. Y los controllers se encargan de regentar cuando van a pasar todas estas acciones, por ejemplo hacer un fade out y colorear de rojo la opción errónea si es el primer fallo de un intento. Un ejemplo dentro del juego de estos view y controller son los que regentan el gameplay y los intentos.

A nivel de código base dentro del flow principal de los intentos, una vez que se selecciona el gameState en gameplay, empieza un flujo basado en coroutinas, este flujo va ejecutando los diferentes estados por los que un intento puede pasar(diagrama de abajo), y dependiendo del input del usuario y de si es correcto o no se resuelve de 3 maneras:

* Correcto -> Correcto y fin del intento
* Fallo 1 -> Remover opción fallida y segundo intento
* Fallo 2 -> Fallo total y fin del intento

A continuación añado un pequeño diagrama(hecho a mano) de flujo en el que me base para ello.

Como datos extra he intentado mantener el código limpio a base de diferentes funciones que realizan una sola tarea para aumentar la legibilidad y la facilidad a la hora de que hubiese que ampliar este proyecto. También creo que esto ayuda a mantener el flujo del código bastante controlado para evitar errores inesperados y ayudar a encontrar los que van surgiendo ahorrando tiempo y esfuerzo.

Creo que a grandes rasgos es una explicación básica de como he intentado estructurar el proyecto, espero que sea de vuestro agrado y pueda continuar en el proceso de selección, pasen un buen día y muchas gracias por su tiempo.