TRABAJO INDIVIDUAL R.A.A.

COMBINACIÓN DE COLORES Y CÓMO ENGAÑAR AL JUGADOR

Por: Noé Fernández Moro (UO251683)

ÍNDICE

1.- INTRODUCCIÓN:

Para este proyecto he usado el nivel de las pociones del juego de mi grupo “Danzantes del Filo”, inspirado en Harry Potter.

La combinación de colores se explicará en base a lo realizado en dicho nivel, así como los “trucos” para engañar al jugador, junto con otros trucos conocidos en videojuegos de talla mundial, con objetivo de demostrar que un desarrollar no siempre tiene que hacer que algo suceda de forma literal, tan solo *que el jugador crea que sí.*

2.- El truco de la interacción con objetos invisibles:

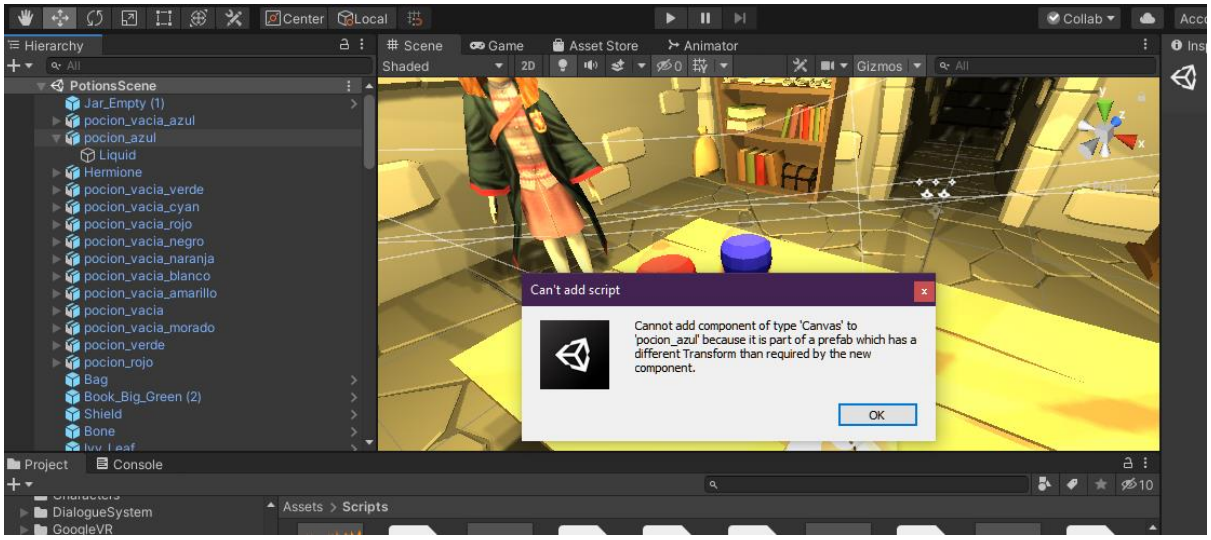
Pese a lo que pueda pensar un usuario novato cuando juega a un videojuego, no todo sucede como parece a simple vista. Hay casos en los que, en medio del desarrollo, un programador no puede conseguir (o al menos no de forma sencilla) que cierto objeto o enemigo interactúe de forma directa con el jugador, y viceversa.

Dado ese contexto, no son pocos los desarrolladores que hacen esfuerzos titánicos para que la acción o reacción que quieren conseguir sea de forma directa entre los dos actores. No obstante, otros prefieren buscar la forma más sencilla de conseguir que *a simple vista* se haga dicha acción o reacción. Y *parece* debería de sustituir en el desarrollo a *hace*.

Centrándonos en este segundo grupo, muchas veces es más sencillo conseguir que un actor interactúe con un objeto intermediario invisible, consiguiendo el mismo efecto que si lo hiciera de forma directa pero en un tiempo mucho más pequeño. Aquí es donde entran los objetos invisibles.

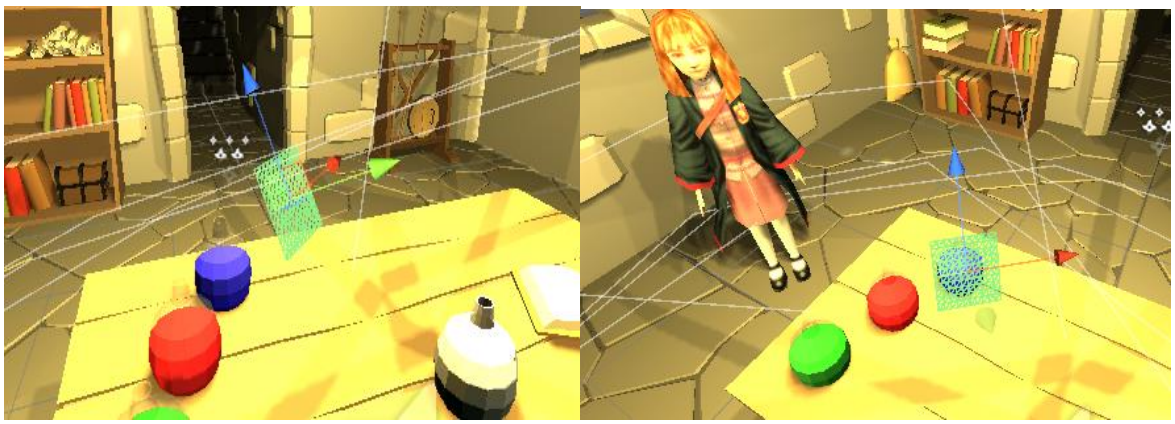
Para ejemplificar esto, explicaré con mucho detalle el uso que le he dado yo a esta idea en mi proyecto.

Tenemos en la mesa 3 pociones, una de cada color: rojo, azul y verde. No obstante, estas pociones se hicieron con un programa externo y han sido cargadas en Unity como “prefab”. ¿Qué significa esto? Que nosotros como desarrolladores NO podemos asignarles scripts a elementos considerados como “prefab”:



Claro que siempre habría la forma de hacer que dicho elemento no fuera considerado como “prefab” (lo cual desconozco). Sin embargo, partiendo de la idea de que personalmente no tengo el conocimiento para abordarlo de una forma directa, perdería bastante tiempo realizando búsquedas por Internet, quizás para darme cuenta después de varias horas que es algo inviable.

En vez de esto, coloqué la cámara del editor de Unity en la misma posición que la cámara del juego, para saber el ángulo de visión. Una vez lo supe, coloqué **3 planos invisibles** delante de las pociones, de un tamaño similar al que tienen, entre la visión del jugador (que siempre estará quieto) y dichas pociones:



Dichos planos contienen los scripts necesarios para interactuar con la vista del jugador, así como con el “click” que puede hacer sobre las

“pociones”. Cabe mencionar que este truco se aplica también a la poción del centro de la mesa, la cual hablaremos más en profundidad en el siguiente punto.

Un ejemplo del uso de planos invisibles es “Animal Crossing”. En dicho juego, puedes hablar con la pata del mostrador si te colocas delante y clicas el botón “A”.

No obstante, si manipulamos las coordenadas del jugador modificando la memoria RAM del juego y traspasamos el mostrador, no podemos hablar con ella por mucho que presionemos el botón.

Esto se debe a que, aunque no lo veamos, estamos interactuando con un panel invisible colocado delante del mostrador, no interactuamos directamente con la pata. En caso de poder depurar el juego, veríamos algo así:

Por esta razón, si la abordamos por un lateral, al no estar el juego preparado para una interacción desde ese ángulo, no podemos interactuar con ella. Y realmente no lo necesitamos. Solamente tenemos que conseguir que *parezca* que lo hacemos.

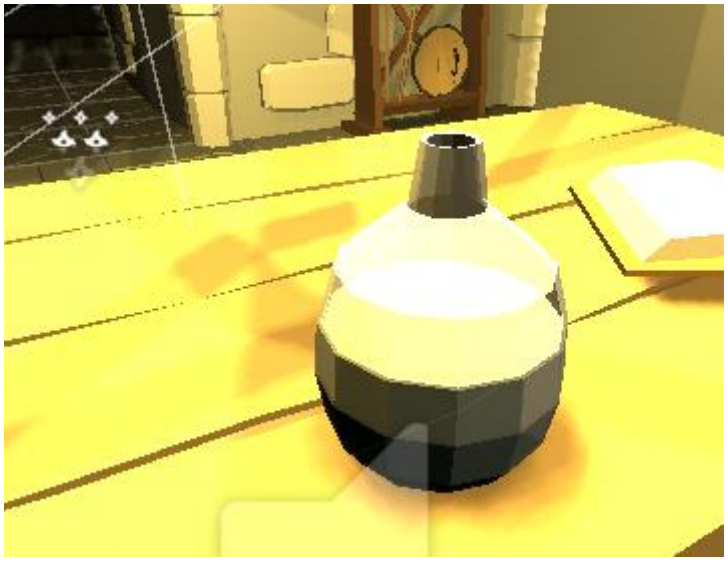
3.- El truco de la (no) generación de objetos en tiempo real:

En desarrollo de videojuegos, cargar constantemente elementos en tiempo real y generarlos en tiempo de ejecución puede llegar a ser una tarea muy costosa. En vez de eso, normalmente se opta por generar la mayor cantidad de elementos posibles al inicio, e ir jugando con su posición o teletransportando al jugador a zonas que ya están cargadas para aumentar la fluidez.

En mi proyecto personal, ejemplifico esto con **el cambio de color de las pociones.**

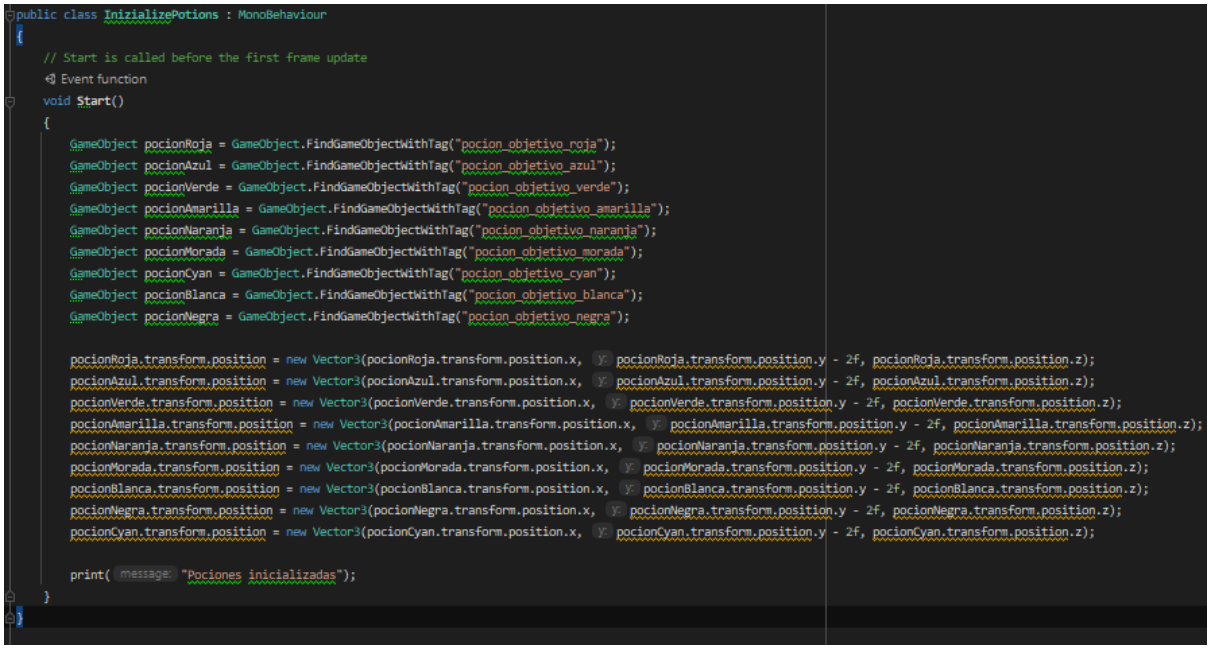
Aunque en tiempo de ejecución dé la sensación de que el material de la poción central va variando, sabemos que es imposible, ya que el jugador no puede interactuar con ninguna poción como hemos explicado en el punto anterior. Entonces, ¿cómo lo hacemos? Muy simple: **todas las pociones que pueden resultar ya están creadas.**

Si pudiéramos detener la ejecución del nivel al instante exacto en que todo se carga, veríamos esto encima de la mesa:



Esta poción “de color raro” que vemos encima de la mesa, se debe a que las 9 pociones posibles están generadas una encima de la otra. Al

comenzar el nivel, **la mesa baja todas las pociones salvo la vacía**, accediendo a ellas gracias a tags únicas.



```
public class InizializePotions : MonoBehaviour
{
    // Start is called before the first frame update
    Event function
    void Start()
    {
        GameObject pocionRoja = GameObject.FindGameObjectWithTag("pocion_objetivo_roja");
        GameObject pocionAzul = GameObject.FindGameObjectWithTag("pocion_objetivo_azul");
        GameObject pocionVerde = GameObject.FindGameObjectWithTag("pocion_objetivo_verde");
        GameObject pocionAmarilla = GameObject.FindGameObjectWithTag("pocion_objetivo_amarilla");
        GameObject pocionNaranja = GameObject.FindGameObjectWithTag("pocion_objetivo_naranja");
        GameObject pocionMorada = GameObject.FindGameObjectWithTag("pocion_objetivo_morada");
        GameObject pocionCyan = GameObject.FindGameObjectWithTag("pocion_objetivo_cyan");
        GameObject pocionBlanca = GameObject.FindGameObjectWithTag("pocion_objetivo_blanca");
        GameObject pocionNegra = GameObject.FindGameObjectWithTag("pocion_objetivo_negra");

        pocionRoja.transform.position = new Vector3(pocionRoja.transform.position.x,  y: pocionRoja.transform.position.y - 2f, pocionRoja.transform.position.z);
        pocionAzul.transform.position = new Vector3(pocionAzul.transform.position.x,  y: pocionAzul.transform.position.y - 2f, pocionAzul.transform.position.z);
        pocionVerde.transform.position = new Vector3(pocionVerde.transform.position.x,  y: pocionVerde.transform.position.y - 2f, pocionVerde.transform.position.z);
        pocionAmarilla.transform.position = new Vector3(pocionAmarilla.transform.position.x,  y: pocionAmarilla.transform.position.y - 2f, pocionAmarilla.transform.position.z);
        pocionNaranja.transform.position = new Vector3(pocionNaranja.transform.position.x,  y: pocionNaranja.transform.position.y - 2f, pocionNaranja.transform.position.z);
        pocionMorada.transform.position = new Vector3(pocionMorada.transform.position.x,  y: pocionMorada.transform.position.y - 2f, pocionMorada.transform.position.z);
        pocionBlanca.transform.position = new Vector3(pocionBlanca.transform.position.x,  y: pocionBlanca.transform.position.y - 2f, pocionBlanca.transform.position.z);
        pocionNegra.transform.position = new Vector3(pocionNegra.transform.position.x,  y: pocionNegra.transform.position.y - 2f, pocionNegra.transform.position.z);
        pocionCyan.transform.position = new Vector3(pocionCyan.transform.position.x,  y: pocionCyan.transform.position.y - 2f, pocionCyan.transform.position.z);

        print( message: "Pociones inicializadas");
    }
}
```

Este script, coge todas las pociones y las baja –2.0f en la dimensión Y, de forma que quedan totalmente fuera de la visión del jugador. La poción actualmente encima de la mesa, cuya altura es –0.7f, queda por encima del resto.

Sabiendo esto, el script que ejecutan los planos de las pociones RGB encima de la mesa poseen un método de detección que devuelve la poción que está encima de la mesa, en base a su altura.

Dicho método, recorre todas las pociones “objetivo” y aquella con altura superior a –1.0 (-0.7 > -1.0) es la poción actual:

```
public void actualizaPocionActual()
{
    String tagActual = "";
    if (GameObject.FindGameObjectWithTag("pocion_objetivo").transform.position.y > -1)
    {
        tagActual = "pocion_objetivo";
    }

    else if (GameObject.FindGameObjectWithTag("pocion_objetivo_roja").transform.position.y > -1)
    {
        tagActual = "pocion_objetivo_roja";
    }

    else if (GameObject.FindGameObjectWithTag("pocion_objetivo_azul").transform.position.y > -1)
    {
        tagActual = "pocion_objetivo_azul";
    }

    else if (GameObject.FindGameObjectWithTag("pocion_objetivo_verde").transform.position.y > -1)
    {
        tagActual = "pocion_objetivo_verde";
    }

    else if (GameObject.FindGameObjectWithTag("pocion_objetivo_amarilla").transform.position.y > -1)
    {
        tagActual = "pocion_objetivo_amarilla";
    }

    else if (GameObject.FindGameObjectWithTag("pocion_objetivo_cyan").transform.position.y > -1)
    {
        tagActual = "pocion_objetivo_cyan";
    }
```

Este método, compartido por los 3 planos de las pociones RGB, recoge e identifica de forma muy sencilla la poción que tiene el jugador encima de la mesa. Una vez lo sabe, el mismo plano posee otro método de reacción, en base a la tag de la poción actual:

```
public void Update()
{
    if (clicada)
    {
        if (pocionActual == "pocion_objetivo")
        {
            GameObject pocActual = GameObject.FindGameObjectWithTag(pocionActual);
            pocActual.transform.position = new Vector3(pocActual.transform.position.x,  y: pocActual.transform.position.y - 2f, pocActual.transform.position.z);

            GameObject pocClicada = GameObject.FindGameObjectWithTag(pocionClicada);
            pocClicada.transform.position = new Vector3(pocClicada.transform.position.x,  y: pocClicada.transform.position.y + 2f, pocClicada.transform.position.z);
        }

        else if (pocionActual == "pocion_objetivo_roja")
        {
            GameObject pocActual = GameObject.FindGameObjectWithTag(pocionActual);
            pocActual.transform.position = new Vector3(pocActual.transform.position.x,  y: pocActual.transform.position.y - 2f, pocActual.transform.position.z);

            GameObject pocClicada = GameObject.FindGameObjectWithTag("pocion_objetivo_morada");
            pocClicada.transform.position = new Vector3(pocClicada.transform.position.x,  y: pocClicada.transform.position.y + 2f, pocClicada.transform.position.z);
        }

        else if (pocionActual == "pocion_objetivo_verde")
        {
            GameObject pocActual = GameObject.FindGameObjectWithTag(pocionActual);
            pocActual.transform.position = new Vector3(pocActual.transform.position.x,  y: pocActual.transform.position.y - 2f, pocActual.transform.position.z);

            GameObject pocClicada = GameObject.FindGameObjectWithTag("pocion_objetivo_cyan");
            pocClicada.transform.position = new Vector3(pocClicada.transform.position.x,  y: pocClicada.transform.position.y + 2f, pocClicada.transform.position.z);
        }

        else if (pocionActual == "pocion_objetivo_azul")
        {
```

Si bien no se ven todas las reacciones del método, se repite la misma estructura: Primero baja la poción actual –2.0f y luego sube la poción que nos interesa +2.0f, dependiendo de la conjunción de colores y resultado que obtenemos en cada caso.

De esta forma, la sensación para el jugador es que el color de la poción central varía, cuando realmente una poción totalmente distinta sube y la que antes estaba encima, baja.

Un ejemplo en la industria de los videojuegos lo tenemos con la generación de escenarios en distintos puntos del mapa, donde el jugador es teletransportado justo después de pasar una puerta, prácticamente sin tiempo de espera:

En la imagen superior, podemos ver un nivel de un juego en Unity donde la fase en cuestión ha sido colocada totalmente fuera de alcance de la vista del jugador en una posición muy elevada. Pasada cierta puerta, el jugador es transportado en cuestion de milésimas de segundo al inicio de esta escena, dando sensación de fluidez y ahorrando recursos.

4.- Explicación de la combinación de colores:

Una vez conocemos cómo el jugador “interactúa” con las pociones, y cómo estas “cambian de color”, es momento de explicar cómo se ha programado dicha combinación.

Lo primero que hice al programar esta combinación fue crear el resultado de (vacía) + (color básico: R-G-B), lo cual resulta en la poción exactamente igual a la clicada:

Una vez hecho esto, creé las combinaciones simples:

Rojo + Azul = Morado

Rojo + Verde = Amarillo

Azul + Verde = Cyan

Tras esto, tuve en cuenta las saturaciones de color. Es decir, qué debería pasar si a un color hecho de A+B le añadías de nuevo A (por ejemplo).

Para evitar andar con una gama de colores absurdamente alta, creé una saturación instantánea hacia el color dominante. Es decir, “gana” el que acabas de añadir. De esta forma:

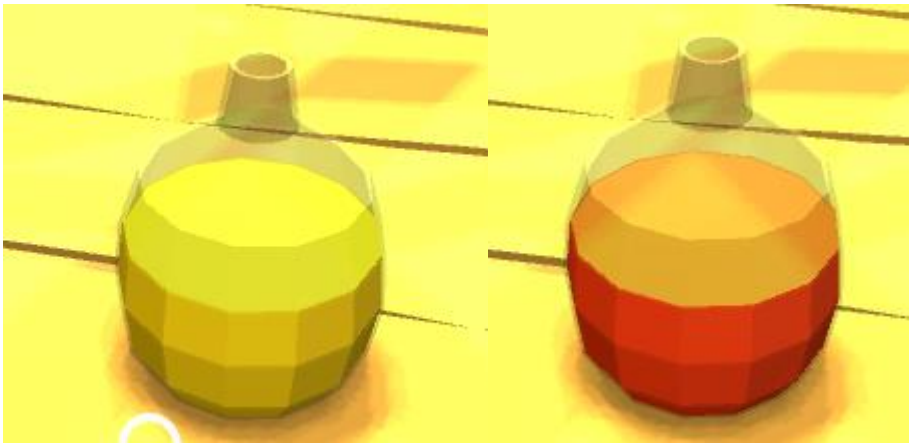
Morado + Azul = Azul Cyan + Azul = Azul

Morado + Rojo = Rojo Cyan + Verde = Verde

No obstante, hay una excepción: el naranja. En caso de añadir rojo al color amarillo, resultará naranja. De la misma forma, si a ese naranja le añades de nuevo verde, volverá a su tonalidad amarilla. De esta forma:

Amarillo + Rojo = Naranja Naranja + Verde = Amarillo

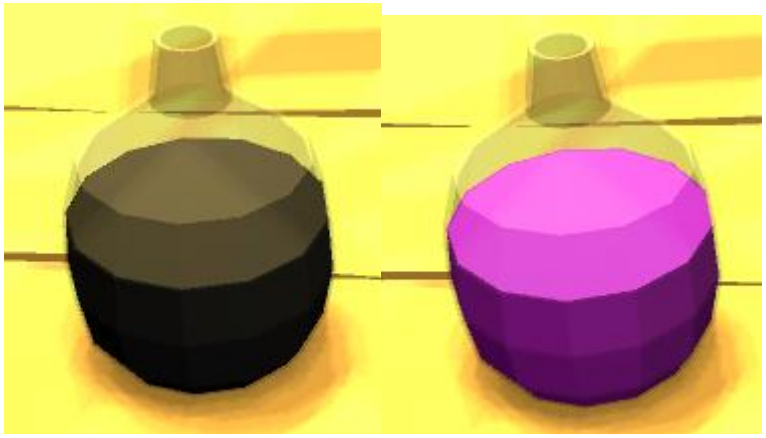
Amarillo + Verde = Verde Naranja + Rojo = Rojo



Finalmente, una vez hemos tenido en cuenta las saturaciones, tan sólo quedaba la opción de combinar un color compuesto con el color faltante. De esta forma:

Morado + Verde = Negro Naranja + Azul = Verde

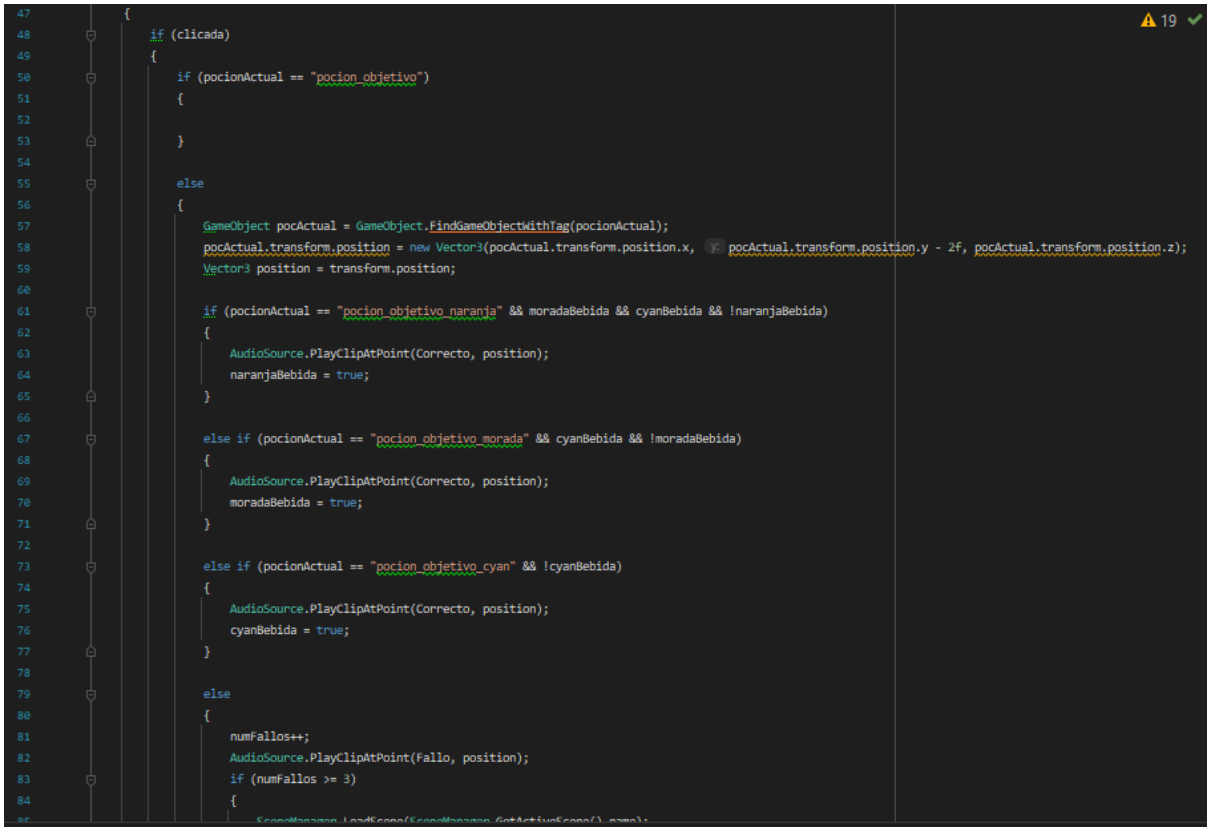
Amarillo + Azul = Verde Cyan + Rojo = Morado



Para finalizar, explicaré cómo “ganamos” en este juego.

El panel de la poción central contiene una memoria de las pociones que se beben. Así, si te bebes 3 pociones cualesquiera en un orden incorrecto (es decir, “fallas” 3 veces) el nivel se reinicia (además, se reproduce un sonido de beber junto con el de fallo en cada intento).

Cuando bebes una poción en el orden correcto, se reproduce un sonido de acierto y se guarda esa información en el propio panel central. Una vez el jugador ha bebido las 3 pociones en el orden correcto (cyan-morado-naranja), se pasa al siguiente nivel. A continuación dejo un fragmento del script de la poción central:



```
47          {
48              if (clicada)
49              {
50                  if (pocionActual == "pocion_objetivo")
51                  {
52
53                  }
54
55                  else
56                  {
57                      GameObject pocActual = GameObject.FindGameObjectWithTag(pocionActual);
58                      pocActual.transform.position = new Vector3(pocActual.transform.position.x,  y: pocActual.transform.position.y - 2f, pocActual.transform.position.z);
59                      Vector3 position = transform.position;
60
61                      if (pocionActual == "pocion_objetivo_naranja" && moradaBebida && cyanBebida && !naranjaBebida)
62                      {
63                          AudioSource.PlayClipAtPoint(Correcto, position);
64                          naranjaBebida = true;
65                      }
66
67                      else if (pocionActual == "pocion_objetivo_morada" && cyanBebida && !moradaBebida)
68                      {
69                          AudioSource.PlayClipAtPoint(Correcto, position);
70                          moradaBebida = true;
71                      }
72
73                      else if (pocionActual == "pocion_objetivo_cyan" && !cyanBebida)
74                      {
75                          AudioSource.PlayClipAtPoint(Correcto, position);
76                          cyanBebida = true;
77                      }
78
79                      else
80                      {
81                          numFallos++;
82                          AudioSource.PlayClipAtPoint(Fallo, position);
83                          if (numFallos >= 3)
84                          {
```

Cabe destacar que, al igual que ocurría con la generación de combinaciones de colores, al ser “bebida” la poción central, esta baja –2.0f en la dimensión Y, subiendo la poción vacía 2.0f en la misma dimensión.