

|  |
| --- |
| Documentación Técnica  ChemLab-AR  2024 |
|  |
| 24 octubre  Austral Software  Creado por: Francisco Zambrano |

|  |
| --- |
| Índice [Índice 2](#_Toc180672176)  [Objetivo 3](#_Toc180672177)  [Casos de uso 3](#_Toc180672178)  [Problema 4](#_Toc180672179)  [Solución 4](#_Toc180672180)  [Estructura del Proyecto 5](#_Toc180672181)  [Arquitectura del proyecto 6](#_Toc180672182)  [Configuración de Vuforia 7](#_Toc180672183)  [Scripts y Lógica de la aplicación 8](#_Toc180672184)  [Evidencia / Código 9](#_Toc180672185)  [Diagrama de secuencia 22](#_Toc180672186) Objetivo Desarrollar una herramienta educativa interactiva que permita a los usuarios experimentar con disoluciones químicas mediante la combinación de solutos y solventes en porcentajes específicos utilizando realidad aumentada.  Dentro de los objetivos, se pueden definir 3 grandes objetivos especificos:   1. Simular de manera visual la combinación de solutos y solventes mediante realidad aumentada. 2. Implementar controles interactivos para ajustar las proporciones de los componentes en una mezcla. 3. Desarrollar una interfaz amigable que permita ajustar proporciones de soluto y solvente y observar los resultados en tiempo real.  Casos de uso Dentro de los casos de uso, podemos definir lo siguiente:  **Actor principal:** Estudiante de química  **Escenario:** El estudiante escanea códigos QR que representan los reactivos (soluto y solvente). Mediante un control deslizante, ajusta las proporciones en la mezcla y observa el resultado en tiempo real, que muestra si la mezcla es válida o no.  **Flujo Normal:**   1. El estudiante abre la aplicación y escanea los códigos QR de los reactivos. 2. Utiliza los controles de porcentaje para ajustar la cantidad de soluto y solvente. 3. Valida la solución y observa el resultado visual (por ejemplo, una solución salina isotónica al 0.9%). 4. La aplicación notifica si la mezcla es correcta.   **Flujo Alterno:** Si el estudiante no ajusta bien las proporciones, la aplicación muestra una combinación errónea y un mensaje indicando que la mezcla no es válida y permite realizar ajustes. Problema El aprendizaje de disoluciones químicas puede ser abstracto y teórico. Las fórmulas matemáticas y los conceptos no siempre son fáciles de visualizar para los estudiantes, lo que dificulta su comprensión.  La necesidad de una herramienta visual e interactiva que permita a los estudiantes experimentar directamente con conceptos de química, como las disoluciones, sin depender de laboratorios físicos. Solución Chemlab-AR es una aplicación que permite a los estudiantes simular disoluciones químicas mediante la realidad aumentada. Los estudiantes pueden ajustar las proporciones de soluto y solvente en una mezcla, y observar el resultado visual en tiempo real, facilitando la comprensión práctica de cómo funcionan las disoluciones.  **Características Clave:**   * Control deslizante interactivo para ajustar proporciones de soluto y solvente. * Modelos 3D visuales que muestran los elementos a mezclar y la mezcla resultante. * Validación automática de la solución basada en las proporciones ajustadas.  Estructura del Proyecto El proyecto Chemlab-AR se organiza utilizando una estructura clara y modular que facilita la gestión de los diferentes componentes, desde los scripts de la lógica del juego hasta los modelos 3D y las bibliotecas externas.  A continuación, se describe la organización de las principales carpetas:   * **Assets/:** Contiene todos los recursos esenciales del proyecto, incluyendo scripts, modelos, prefabs y escenas.   + **Scripts/:** Almacena los scripts de C# que gestionan toda la lógica de la aplicación.     - **CloudManager2.cs:** Se encarga de manejar los estados de detección de los objetos en realidad aumentada (soluto, solvente y caldero). Además, gestiona las interacciones entre los diferentes elementos detectados y las acciones que se deben realizar en función de los resultados de la mezcla.     - **SolutionValidator.cs:** Controla la lógica para validar si las combinaciones de soluto y solvente son correctas, así como la gestión de la interfaz de usuario (como el slider para ajustar proporciones y el botón de validación).   + **Resources/:** Contiene recursos adicionales, como imágenes, audios, o archivos que pueden ser cargados dinámicamente durante la ejecución de la aplicación.   Estos recursos son útiles cuando se necesita cargar contenido en tiempo de ejecución sin que esté directamente vinculado a la escena desde el principio.   * + **Plugins/:** En esta carpeta se almacenan herramientas y librerías externas que se integran en el proyecto, como el SDK de Vuforia, que es esencial para la detección de imágenes y objetos en realidad aumentada.   Vuforia proporciona la tecnología necesaria para reconocer los targets (códigos QR) utilizados en la aplicación y para activar los modelos 3D correspondientes.   * + **Scenes/:** Almacena las diferentes escenas que conforman la experiencia del usuario en la aplicación.     - **Main:** La pantalla principal de la aplicación.     - **Jugar:** Escena donde el usuario puede seleccionar el modo de juego.     - **Option:** Escena de configuración donde se pueden ajustar parámetros generales de la aplicación.     - **Soluciones:** Escena donde se visualizan y validan las soluciones entre soluto y solvente generadas por los usuarios.   + **Componentes/:** Contiene los modelos 3D utilizados en la aplicación, como los diferentes elementos químicos (solutos y solventes) y el caldero donde se realiza la mezcla.   Los modelos 3D permiten la visualización de los reactivos en realidad aumentada, brindando al usuario una experiencia interactiva y visualmente atractiva.   * + - **Prefabs/:** Aquí se encuentran los modelos 3D preconfigurados y otros componentes reutilizables, optimizados para que puedan ser instanciados fácilmente en distintas partes del juego.   Los prefabs son versiones guardadas de modelos 3D o conjuntos de objetos que pueden reutilizarse en varias escenas sin necesidad de configurarlos nuevamente. Arquitectura del proyecto  Configuración de Vuforia Vuforia es la plataforma utilizada en Chemlab-AR para la detección y seguimiento de imágenes en realidad aumentada, permitiendo que la aplicación reconozca diferentes targets (como solutos, solventes y el caldero) mediante códigos QR. La correcta configuración de Vuforia en el proyecto es esencial para asegurar una detección precisa y una interacción fluida entre el mundo físico y los modelos 3D en la aplicación. Esta sección cubre cómo se configuraron los targets, las bases de datos y las configuraciones dentro de Unity, así como la interacción entre los scripts y Vuforia para activar los modelos en la escena.  **Base de Vuforia:**   * La base de datos de Vuforia está configurada para contener los targets (códigos QR) que representan los diferentes solutos, solventes y el caldero. Cada target es un código QR único vinculado a un objeto en el entorno de realidad aumentada. * **Configuración de Targets:**   Para añadir nuevos targets, se utiliza el portal de Vuforia Developer, donde se suben imágenes o códigos QR y se generan las bases de datos para su integración en Unity. Una vez cargados, se enlazan con los modelos 3D correspondientes en Unity.  **Vuforia Settings:**   * **Licencia de Vuforia:**   Es necesario incluir una clave de licencia obtenida desde la plataforma de Vuforia. Se recomienda utilizar la opción de detección de alta calidad para mejorar la precisión de la detección de códigos QR.   * **Configuración de Base de Datos:**   En Unity, las bases de datos de Vuforia se habilitan desde el Vuforia Configuration, donde se seleccionan las bases de datos correspondientes a los targets creados.  **Interacción con Unity:**   * **Eventos de Detección:** Vuforia envía señales a Unity cuando se detecta un target. Estos eventos son gestionados por el script CloudManager2.cs, que controla la lógica de lo que debe ocurrir en la escena cuando los targets son reconocidos. * **Modelo 3D Activado:** Una vez que se detecta el target, el script activa el modelo 3D asociado y lo coloca en la escena.  Scripts y Lógica de la aplicación Descripción de los Scripts Principales  **CloudManager2.cs:**   * **Función Principal:** Es el controlador principal que gestiona la detección de los objetos, muestra el slider y maneja el botón de validar. Este script también se encarga de detectar el caldero, soluto, solvente, y asegurarse de que los tres estén presentes antes de mostrar el slider y el botón de validación. * **Estructura Interna:**   + Métodos:     - **Start():** Inicializa los elementos y configura el sistema de detección.     - **Update():** Verifica continuamente si los objetos han sido detectados y desencadena la validación cuando se cumplen las condiciones necesarias.     - **HandleDetection():** Función que maneja la lógica para la detección y acción sobre los objetos reconocidos por Vuforia. * **Variables Importantes:**   + **targetSoluto, targetSolvente, targetCaldero:** Variables que representan los objetos detectados.   + **isDetected:** Indica si un objeto ha sido detectado o no.   **SolutionValidator.cs:**   * **Función Principal:** Se invoca únicamente cuando se presiona el botón de validar en CloudManager2 para verificar las combinaciones de solutos y solventes y así generar una solución. * **Estructura Interna:**   + Métodos:     - **ValidateSolution():**Es el método principal que se ejecuta cuando CloudManager2 lo invoca tras presionar el botón de validar. Evalúa si la combinación y los porcentajes de los elementos son válidos y luego muestra la solución correcta o incorrecta.  Evidencia / Código A continuación, se detalla el funcionamiento de los scripts con el detalle de cada funcionalidad.  **SCRIPT CloudManager2:**  **Variables:**   * + **cloudRecoBehaviour:** Maneja el comportamiento de reconocimiento en la nube de Vuforia.   [SerializeField] private CloudRecoBehaviour cloudRecoBehaviour;   * + **mandatoryTarget:** Objeto que siempre debe estar presente (el caldero).   [SerializeField] public GameObject mandatoryTarget;   * + **possibleSolutes/possibleSolvents:** Arrays que contienen los objetos posibles que pueden ser solutos o solventes.   [SerializeField] private GameObject[] possibleSolutes;  [SerializeField] private GameObject[] possibleSolvents;   * + **solutionObjects:** Lista de objetos que representan las posibles soluciones correctas.   [SerializeField] public GameObject[] solutionObjects;   * + **cauldronTransform:** Ubicación del caldero para fusionar los objetos.   [SerializeField] private Transform cauldronTransform;   * + **percentageSlider, percentageText:** Slider y texto que permiten ajustar los porcentajes de soluto/solvente.   [SerializeField] private Slider percentageSlider;  [SerializeField] private Text percentageText;   * + **validationButton, clearButton:** Botones para validar la combinación y limpiar la pantalla.   [SerializeField] public Button validationButton;  [SerializeField] private Button clearButton;   * + **incorrectSolutionObjectWaterSalt/incorrectSolutionObjectAlcoholIodine:** Objetos que se muestran en caso de una combinación incorrecta.   [SerializeField] private GameObject incorrectSolutionObjectWaterSalt;  [SerializeField] private GameObject incorrectSolutionObjectAlcoholIodine;  **Máquina de Estados:**  Este script utiliza una máquina de estados básica, que regula las transiciones entre varios estados clave:   * **State.WaitingForDetection:** El estado inicial en el que el sistema espera que se detecten los elementos.               case State.WaitingForDetection:                  percentageSlider.gameObject.SetActive(false);                  percentageText.gameObject.SetActive(false);                  validationButton.gameObject.SetActive(false);                  break;   * **State.DetectionComplete:** Cuando se detectan el caldero, el soluto y el solvente, muestra el slider y el botón de validación.   case State.DetectionComplete:                  percentageSlider.gameObject.SetActive(true);                  percentageText.gameObject.SetActive(true);                  validationButton.gameObject.SetActive(true);                  break;   * **State.Validating:** Ejecuta la validación de la combinación al presionar el botón de validar.               case State.Validating:                  break;   * **State.Cleaning:** Limpia los objetos de la escena, prepara la pantalla para una nueva detección y restablece los valores.               case State.Cleaning:                  clearButton.gameObject.SetActive(true);                  break;  **Ciclo de Estados:**   * **WaitingForDetection:** El sistema está esperando que se detecten los elementos clave. * **DetectionComplete:** Los elementos han sido detectados, se muestra la interfaz de validación. * **Validating:** El usuario ha presionado el botón de validar, y se está evaluando la combinación de soluto y solvente. * **Cleaning:** La pantalla se limpia y se prepara para una nueva interacción.   **Métodos:**   1. **Awake():**    * **Descripción:** Es el método que se llama cuando el script se inicializa. Configura todos los componentes necesarios para que la escena funcione correctamente.    * **Función:**       + Inicializa el sistema de detección de Vuforia.      + Asigna los objetos detectables como soluto, solvente y caldero.      + Configura el slider y los botones, ocultándolos inicialmente.    * **Uso en el código:** Este método es crucial para la configuración inicial de los elementos y las referencias, garantizando que el sistema esté listo para funcionar desde el inicio de la escena.   private void Awake()      {          cloudRecoBehaviour.RegisterOnInitializedEventHandler(OnInitialized);          cloudRecoBehaviour.RegisterOnStateChangedEventHandler(OnStateChanged);          cloudRecoBehaviour.RegisterOnNewSearchResultEventHandler(OnNewSearchResult);          SetState(State.WaitingForDetection);          targetObjects[mandatoryTarget.name] = mandatoryTarget;          foreach (var target in possibleSolutes)          {              targetObjects[target.name] = target;          }          foreach (var target in possibleSolvents)          {              targetObjects[target.name] = target;          }          if (percentageSlider != null)          {              percentageSlider.minValue = 0;              percentageSlider.maxValue = 100;              percentageSlider.value = 50;              percentageSlider.onValueChanged.AddListener(OnSliderValueChanged);          }          if (validationButton != null)          {              validationButton.onClick.AddListener(OnValidateButtonClick);          }          if (clearButton != null)          {              clearButton.onClick.AddListener(OnClearButtonClick);              clearButton.gameObject.SetActive(false); // Ocultar el botón al inicio          }          SetSolutionObjects();      }   1. **SetState(State newState):**    * **Descripción:** Es el método que se llama cuando el script se inicializa. Configura todos los componentes necesarios para que la escena funcione correctamente.    * **Función:**      + Inicializa el sistema de detección de Vuforia.      + Asigna los objetos detectables como soluto, solvente y caldero.      + Configura el slider y los botones, ocultándolos inicialmente.    * **Uso en el código:** Este método es crucial para la configuración inicial de los elementos y las referencias, garantizando que el sistema esté listo para funcionar desde el inicio de la escena.   private void SetState(State newState)      {          currentState = newState;          Debug.Log($"Estado cambiado a: {currentState}");          switch (currentState)          {              case State.WaitingForDetection:                  percentageSlider.gameObject.SetActive(false);                  percentageText.gameObject.SetActive(false);                  validationButton.gameObject.SetActive(false);                  break;              case State.DetectionComplete:                  percentageSlider.gameObject.SetActive(true);                  percentageText.gameObject.SetActive(true);                  validationButton.gameObject.SetActive(true);                  break;              case State.Validating:                  break;              case State.Cleaning:                  clearButton.gameObject.SetActive(true);                  break;          }      }   1. **SetSolutionObjects():**    * **Descripción:** Configura las combinaciones válidas de solutos y solventes, determinando qué resultados serán correctos y cuáles no.    * **Función:**      + Define las combinaciones correctas de soluto-solvente y el objeto de solución que se debe mostrar en la escena.    * **Uso en el código:** Se utiliza para almacenar las combinaciones permitidas de soluto y solvente, asegurando que la validación reconozca las combinaciones correctas.   private void SetSolutionObjects()      {          solutionCombinations["water"] = new Dictionary<string, GameObject>();          solutionCombinations["water"]["salt"] = solutionObjects[0];          solutionCombinations["alcohol"] = new Dictionary<string, GameObject>();          solutionCombinations["alcohol"]["iodine"] = solutionObjects[1];      }   1. **OnInitialized(), OnStateChanged():**    * **Descripción:** Son métodos de callback que se invocan cuando Vuforia inicializa o cambia de estado (por ejemplo, comienza o termina el escaneo).    * **Función:**      + **OnInitialized():** Notifica cuando el sistema de reconocimiento de Vuforia está completamente inicializado.      + **OnStateChanged():** Proporciona retroalimentación cuando cambia el estado de escaneo en Vuforia.    * **Uso en el código:** Facilita la integración de Vuforia con el sistema de detección y la lógica de la escena.       private void OnInitialized(CloudRecoBehaviour cloudRecoBehaviour) => Debug.Log("Cloud initialized");      private void OnStateChanged(bool scanning) => Debug.Log(scanning ? "Is scanning" : "Not scanning");   1. **OnNewSearchResult(CloudRecoBehaviour.CloudRecoSearchResult result):**    * **Descripción:** Este método se invoca cuando Vuforia detecta un nuevo objeto. Activa los objetos detectados y empieza a verificar si todos los elementos están presentes.    * **Función:**      + Activa los objetos observados y los añade a la lista de elementos detectados.      + Verifica si ya están presentes todos los elementos necesarios para la validación.    * **Uso en el código:** Funciona como un puente entre el sistema de Vuforia y la lógica de validación, activando los elementos detectados en la escena.   private void OnNewSearchResult(CloudRecoBehaviour.CloudRecoSearchResult result)      {          if (targetObjects.ContainsKey(result.TargetName))          {              GameObject target = targetObjects[result.TargetName];              cloudRecoBehaviour.EnableObservers(result, target);              detectedTargets.Add(result.TargetName);              StartCoroutine(CheckAllTargetsDetectedWithDelay());          }      }   1. **CheckAllTargetsDetected():**    * **Descripción:** Verifica si todos los objetos (caldero, soluto y solvente) han sido detectados para proceder con la validación.    * **Función:**      + Revisa si los tres elementos están presentes en la escena.      + Si están presentes, cambia el estado a DetectionComplete y muestra el slider y el botón de validación.    * **Uso en el código:** Se usa para controlar si el sistema está listo para validar la mezcla o si debe seguir esperando la detección de más objetos.   private void CheckAllTargetsDetected()      {          bool soluteDetected = false;          bool solventDetected = false;          bool calderoDetected = false;          var calderoObserver = mandatoryTarget.GetComponent<ObserverBehaviour>();          if (calderoObserver != null && (calderoObserver.TargetStatus.Status == Status.TRACKED || calderoObserver.TargetStatus.Status == Status.EXTENDED\_TRACKED))          {              calderoDetected = true;          }          foreach (var solute in possibleSolutes)          {              var soluteObserver = solute.GetComponent<ObserverBehaviour>();              if (soluteObserver != null && (soluteObserver.TargetStatus.Status == Status.TRACKED || soluteObserver.TargetStatus.Status == Status.EXTENDED\_TRACKED))              {                  soluteDetected = true;                  currentSolute = solute;                  break;              }          }          foreach (var solvent in possibleSolvents)          {              var solventObserver = solvent.GetComponent<ObserverBehaviour>();              if (solventObserver != null && (solventObserver.TargetStatus.Status == Status.TRACKED || solventObserver.TargetStatus.Status == Status.EXTENDED\_TRACKED))              {                  solventDetected = true;                  currentSolvent = solvent;                  break;              }          }          if (calderoDetected && soluteDetected && solventDetected)          {              SetState(State.DetectionComplete);              UpdatePercentageText(percentageSlider.value);          }          else          {              SetState(State.WaitingForDetection);          }      }   1. **OnSliderValueChanged(float value):**    * **Descripción:** Actualiza el texto que muestra los porcentajes de soluto y solvente cada vez que el usuario cambia el valor del slider.    * **Función:**      + Refleja en la interfaz los porcentajes que el usuario selecciona mediante el slider.    * **Uso en el código:** Se activa cada vez que el slider se ajusta, mostrando el nuevo valor en la pantalla.   private void OnSliderValueChanged(float value) => UpdatePercentageText(value);   1. **OnValidateButtonClick():**    * **Descripción:** Se ejecuta cuando el usuario presiona el botón de validar. Cambia el estado a Validating, invoca SolutionValidator y oculta la interfaz de usuario después de un tiempo.    * **Función:**      + Evalúa la combinación de soluto y solvente al invocar la lógica de validación.      + Oculta el slider y el botón de validación tras un pequeño retraso.    * **Uso en el código:** Se invoca cuando se realiza una mezcla y el usuario quiere verificar si es correcta.   private void OnValidateButtonClick()      {          float solutePercentage = percentageSlider.value;          float solventPercentage = 100 - solutePercentage;          SetState(State.Validating);          OnValidation?.Invoke(currentSolvent.name, currentSolute.name, solutePercentage, solventPercentage);          // Ocultar UI después de la validación          StartCoroutine(HideValidationUI());          // Mostrar el botón de limpiar          clearButton.gameObject.SetActive(true);      }   1. **OnClearButtonClick():**    * **Descripción:** Se ejecuta cuando se presiona el botón de limpiar, restableciendo la escena y reiniciando el proceso de detección.    * **Función:**      + Limpia todos los objetos detectados y restablece la interfaz para permitir una nueva mezcla.    * **Uso en el código:** Permite que el usuario reinicie la escena para realizar una nueva combinación.   private void OnClearButtonClick()      {          SetState(State.Cleaning);          detectedTargets.Clear();          foreach (var target in targetObjects.Values)          {              var observer = target.GetComponent<ObserverBehaviour>();              if (observer != null)              {                  observer.enabled = false;                  observer.enabled = true;              }          }          foreach (var solute in possibleSolutes)          {              solute.SetActive(true);          }          foreach (var solvent in possibleSolvents)          {              solvent.SetActive(true);          }          if (mandatoryTarget != null) mandatoryTarget.SetActive(true);          foreach (var solution in solutionObjects) solution.SetActive(false);          if (incorrectSolutionObjectWaterSalt != null) incorrectSolutionObjectWaterSalt.SetActive(false);          if (incorrectSolutionObjectAlcoholIodine != null) incorrectSolutionObjectAlcoholIodine.SetActive(false);          // Ocultar el botón de limpiar          clearButton.gameObject.SetActive(false);          SetState(State.WaitingForDetection);          StartCoroutine(CheckAllObserversAfterRestart());      }   1. **CheckAllObserversAfterRestart():**    * **Descripción:** Este método introduce una pequeña pausa antes de verificar nuevamente si todos los observadores de Vuforia están listos para la detección.    * **Función:**      + Permite reiniciar la detección de objetos después de limpiar los datos anteriores.    * **Uso en el código:** Se utiliza después de limpiar la escena para asegurarse de que Vuforia pueda comenzar a detectar nuevamente.       private IEnumerator CheckAllObserversAfterRestart()      {          yield return new WaitForSeconds(0.5f);          StartCoroutine(CheckAllTargetsDetectedWithDelay());      }   1. **Update():**    * **Descripción:** Verifica continuamente si algún objeto ha dejado de ser detectado y actualiza el estado en consecuencia.    * **Función:**      + Elimina de la lista los objetos que ya no están siendo observados por Vuforia.    * **Uso en el código:** Se invoca constantemente para asegurarse de que los objetos en la escena se estén detectando correctamente en todo momento.       private void Update()      {          List<string> targetsToRemove = new List<string>();          foreach (var targetName in detectedTargets)          {              var observer = targetObjects[targetName].GetComponent<ObserverBehaviour>();              if (observer != null && observer.TargetStatus.Status == Status.NO\_POSE)              {                  targetsToRemove.Add(targetName);              }          }          foreach (var targetName in targetsToRemove)          {              detectedTargets.Remove(targetName);          }          CheckAllTargetsDetected();      }   1. **IEnumerator CheckAllTargetsDetectedWithDelay():**    * **Descripción:** Este método introduce un pequeño retraso de 0.2 segundos antes de ejecutar la función CheckAllTargetsDetected(). Esto es útil para asegurarse de que los estados de detección de los objetos se actualicen correctamente antes de proceder con la lógica.    * **Función:**      + Espera brevemente antes de volver a verificar si los targets (caldero, soluto, solvente) están presentes.    * **Uso en el código:** Se invoca cuando un nuevo objeto es detectado, permitiendo que los estados de los objetos se estabilicen antes de ejecutar la validación.       private IEnumerator CheckAllTargetsDetectedWithDelay()      {          yield return new WaitForSeconds(0.2f);          CheckAllTargetsDetected();      }   1. **void UpdatePercentageText(float value):**    * **Descripción:** Este método actualiza el texto en pantalla que muestra el porcentaje actual de soluto y solvente en función del valor del slider. La proporción del slider se convierte en números enteros antes de mostrarse.    * **Función:**      + Muestra los porcentajes de soluto y solvente que el usuario selecciona en el slider.    * **Uso en el código:** Se llama cada vez que el valor del slider cambia, proporcionando una retroalimentación visual inmediata al usuario.       private void UpdatePercentageText(float value)      {          if (percentageText != null)          {              float solutePercentage = Mathf.Round(value);              float solventPercentage = Mathf.Round(100 - value);              percentageText.text = $"Soluto: {solutePercentage}%\nSolvente: {solventPercentage}%";          }      }   1. **IEnumerator HideValidationUI():**    * **Descripción:** Este método introduce un retraso de 2 segundos después de que el usuario presiona el botón de validar, y luego oculta el slider, el texto del porcentaje y el botón de validación.    * **Función:**      + Oculta los controles de validación (slider, texto, botón) después de que se ha completado una validación para evitar nuevas interacciones innecesarias.    * **Uso en el código:** Se invoca después de validar la mezcla, manteniendo la interfaz limpia y lista para la siguiente acción.       private IEnumerator HideValidationUI()      {          yield return new WaitForSeconds(2f);          percentageSlider.gameObject.SetActive(false);          percentageText.gameObject.SetActive(false);          validationButton.gameObject.SetActive(false);      }  **SCRIPT SolutionValidator:**  **Métodos**   * **Awake()**   + **Descripción:** Este método es llamado al inicializar el script. Configura el vínculo con el script CloudManager2 y sus eventos.   + **Función:**     - Busca una referencia a CloudManager2 usando FindFirstObjectByType para poder invocar su funcionalidad.     - Suscribe el método ValidateSolution() al evento de validación que se dispara en CloudManager2 cuando se presiona el botón de validación.   + **Uso en el código:** Inicializa la conexión con CloudManager2 y asegura que este script reciba las notificaciones de validación cuando el usuario intente validar una mezcla.       private void Awake()      {          cloudManager = FindFirstObjectByType<CloudManager2>();          cloudManager.OnValidation += ValidateSolution;      }   * **OnDestroy()**   + **Descripción:** Este método es invocado cuando el objeto que contiene el script está a punto de ser destruido o desactivado.   + **Función:**     - Elimina la suscripción al evento de validación de CloudManager2 para evitar posibles errores o referencias nulas.   + **Uso en el código:** Previene fugas de memoria o comportamientos inesperados al asegurarse de que este script ya no escuche eventos cuando sea destruido.wake()       private void OnDestroy()      {          cloudManager.OnValidation -= ValidateSolution;      }   * **ValidateSolution(string solvent, string solute, float solutePercentage, float solventPercentage)**   + **Descripción:** Este método es llamado por CloudManager2 cuando el usuario presiona el botón de validación. Evalúa la combinación de soluto y solvente, además de los porcentajes, para determinar si es una mezcla válida.   + **Función:**     - Comprueba si la combinación de soluto y solvente, junto con los porcentajes, es correcta.     - Si la combinación es correcta, muestra la solución adecuada.     - Si la combinación es incorrecta, muestra un objeto de error.   + **Uso en el código:** Se utiliza para verificar las combinaciones específicas de solutos y solventes, asegurando que el usuario obtenga retroalimentación (correcta o incorrecta) sobre la mezcla que realizó.   private void ValidateSolution(string solvent, string solute, float solutePercentage, float solventPercentage)      {          GameObject resultObject = null;          if (solvent == "water" && solute == "salt")          {              if (solutePercentage >= 1 && solutePercentage <= 10)                  resultObject = cloudManager.solutionObjects[0];              else                  resultObject = waterSaltError;          }          else if (solvent == "alcohol" && solute == "iodine")          {              if (solutePercentage >= 2 && solutePercentage <= 12)                  resultObject = cloudManager.solutionObjects[1];              else                  resultObject = alcoholIodineError;          }          if (resultObject != null)              StartCoroutine(FusionEffect(cloudManager.currentSolute, cloudManager.currentSolvent, cloudManager.mandatoryTarget, resultObject));      }   * **FusionEffect(GameObject solute, GameObject solvent, GameObject cauldron, GameObject resultObject)**   + **Descripción:** Este método implementa una animación visual para simular la mezcla de los solutos y solventes en el caldero, desplazando los objetos hacia el centro y mostrando el resultado de la fusión.   + **Función:**     - Anima los objetos de soluto y solvente para que se muevan hacia el caldero durante un tiempo determinado.     - Una vez completada la animación, desactiva los objetos de soluto y solvente, y activa el objeto de resultado (correcto o incorrecto).     - Muestra el botón de limpiar para que el usuario pueda reiniciar la mezcla.   + **Uso en el código:** Crea un efecto visual de fusión que le da al usuario una retroalimentación visual del proceso de combinación de los objetos.       private IEnumerator FusionEffect(GameObject solute, GameObject solvent, GameObject cauldron, GameObject resultObject)      {          float fusionDuration = 2f;          float elapsedTime = 0f;          Vector3 soluteInitialPosition = solute.transform.position;          Vector3 solventInitialPosition = solvent.transform.position;          Vector3 cauldronPosition = cauldron.transform.position;          while (elapsedTime < fusionDuration)          {              float t = elapsedTime / fusionDuration;              solute.transform.position = Vector3.Lerp(soluteInitialPosition, cauldronPosition, t);              solvent.transform.position = Vector3.Lerp(solventInitialPosition, cauldronPosition, t);              elapsedTime += Time.deltaTime;              yield return null;          }          solute.SetActive(false);          solvent.SetActive(false);          cauldron.SetActive(false);          resultObject.transform.position = cauldronPosition;          resultObject.SetActive(true);          clearButton.gameObject.SetActive(true);      } Diagrama de secuencia Este diagrama de secuencia ilustra la interacción entre el usuario y los componentes clave del sistema, como CloudManager2, SolutionValidator, y la interfaz de usuario (UI - Slider). El usuario escanea los códigos QR de los elementos químicos, lo que permite a CloudManager2 detectar los objetos y mostrar el slider en la interfaz. Tras ajustar las proporciones en el slider, CloudManager2 envía los datos a SolutionValidator para validar la mezcla, y finalmente se muestra el resultado al usuario.  **Diagrama  Descripción generada automáticamente** |