Proyecto final:

En la primera entrega se nos insta a escoger un capitulo de los simpsons del cual basados en su trama haremos un juego codificado desde 0 utilizando las herramientas de qt creator y la teoría dada a lo largo del curso

Como equipo escogimos el capitulo 25 de la sexta temporada y el primer capitulo de la séptima; en el capitulo la trama que queremos desarrollar es el momento del atentado al señor burns creando un juego de pistas y tareas para resolver un misterio, en este caso el intento de asesinato del señor burns (que realmente fue un accidente por parte de la hija menor de homero) durante los diferentes escenario se planea poner pequeños minijuegos que ayudaran al personaje que controlamos a obtener las pistas necesarias para descubir la verdad detrás del accidente que conmociona a Springfield.

En cuanto a sprites hemos tenido dificultades para hallar algunos personajes necesarios para la trama, como el mismo señor burns además de algunos escenarios necesarios para el juego

# Diseño y Estructura del Juego: "¿Quién mató al Sr. Burns?"

## 1. Introducción

Este documento describe el diseño y la estructura básica de un juego basado en el episodio "¿Quién mató al Sr. Burns?" de Los Simpsons. El juego se implementará en Qt Creator y está diseñado para incluir elementos de interacción, búsqueda de pistas, y un sistema de diálogo con sospechosos que ayuda a resolver el misterio.

## 2. Estructura de Niveles del Juego

El juego consta de tres niveles principales, cada uno con objetivos específicos para avanzar en la investigación:

1. \*\*Nivel 1 - Escena del Crimen\*\*: El jugador, interpretando a Lisa Simpson, debe recolectar pistas en la planta nuclear, examinando diferentes ubicaciones en la escena.  
2. \*\*Nivel 2 - Entrevista a Sospechosos\*\*: Basado en las pistas recogidas, Lisa interroga a varios sospechosos (Smithers, Moe, Homer, etc.) para recopilar más información.  
3. \*\*Nivel 3 - Resolución del Caso\*\*: Con las pistas recolectadas, Lisa evalúa la información y deduce quién es el culpable a través de un rompecabezas lógico o seleccionando entre varias opciones.

## 3. Funcionalidades Clave

- \*\*Recolección de Pistas\*\*: Las pistas se almacenan en una lista dinámica (`vector`) con información específica.  
- \*\*Interacción con Personajes\*\*: Se utiliza un `map` para almacenar y gestionar las respuestas de cada sospechoso.  
- \*\*Sistema de Progreso\*\*: Se implementa una clase específica para gestionar el progreso del jugador.

# Diseño de Clases del Juego "¿Quién mató al Sr. Burns?"

## Clase Juego

Atributos:  
- privado:  
 - vector<Pista> inventarioPistas: Almacena todas las pistas recolectadas en el juego.  
 - int nivelActual: Controla el nivel en el que se encuentra el jugador.  
 - int puntosVerdad: Lleva el conteo de puntos obtenidos por respuestas correctas y pistas clave.  
  
Métodos:  
- público:  
 - Juego(): Constructor que inicializa los valores del juego.  
 - void iniciarNivel(int nivel): Configura y cambia al nivel seleccionado.  
 - void cargarEscena(string nombreEscena): Carga la escena específica según el nivel actual.  
 - void mostrarInventario(): Muestra todas las pistas en el inventario.  
 - void actualizarPuntosVerdad(int puntos): Suma puntos al progreso de verdad del jugador.  
 - void guardarJuego(): Guarda el estado actual del juego.  
 - void cargarJuego(): Restaura el estado del juego guardado.  
 - void reiniciarNivel(): Reinicia el nivel actual para un nuevo intento.

## Clase Nivel (Abstracta)

Atributos:  
- protegido:  
 - string objetivo: Describe el objetivo principal del nivel.  
  
Métodos:  
- público:  
 - Nivel(): Constructor abstracto.  
 - virtual void iniciarNivel() = 0: Método abstracto que inicializa el nivel.  
 - virtual void terminarNivel() = 0: Método abstracto que termina el nivel.  
 - void mostrarObjetivo(): Muestra el objetivo actual del nivel.

## Clase NivelEscenaCrimen

Atributos:  
- privado:  
 - vector<Pista> pistasNivel: Almacena las pistas disponibles en el nivel.  
 - vector<ObjetoInteractivo\*> objetosInteractivos: Guarda los objetos en la escena para interacción.  
  
Métodos:  
- público:  
 - NivelEscenaCrimen(): Constructor que inicializa los atributos del nivel.  
 - void recolectarPistas(): Almacena las pistas encontradas en inventarioPistas.  
 - void mostrarPistasRecogidas(): Muestra la lista de pistas recogidas en el nivel.  
 - bool verificarInteraccion(string objeto): Verifica si un objeto interactivo se puede inspeccionar.

## Clase NivelEntrevista

Atributos:  
- privado:  
 - vector<Sospechoso> sospechosos: Almacena los personajes entrevistables en el nivel.  
  
Métodos:  
- público:  
 - NivelEntrevista(): Constructor.  
 - void iniciarDialogo(Sospechoso\* sospechoso): Inicia el diálogo con un sospechoso.  
 - void mostrarOpcionesDialogo(): Muestra las preguntas disponibles.  
 - void analizarRespuesta(int indicePregunta, string respuestaJugador): Analiza la respuesta del sospechoso.

## Clase NivelResolucion

Atributos:  
- privado:  
 - vector<Pista> pistasClaves: Pistas específicas necesarias para resolver el caso.  
  
Métodos:  
- público:  
 - NivelResolucion(): Constructor.  
 - void resolverCaso(): Permite al jugador seleccionar al culpable.  
 - void verificarConclusiones(): Confirma si las conclusiones son correctas.  
 - void mostrarResumenPistas(): Muestra un resumen de las pistas recolectadas relevantes.

## Clase Pista

Atributos:  
- privado:  
 - string descripcion: Descripción de la pista.  
 - bool esRelevante: Indica si la pista es clave.  
  
Métodos:  
- público:  
 - Pista(string descripcion, bool esRelevante): Constructor.  
 - string obtenerDescripcion(): Retorna la descripción de la pista.  
 - void marcarComoRelevante(): Marca la pista como relevante.

## Clase Sospechoso

Atributos:  
- privado:  
 - string nombre: Nombre del sospechoso.  
 - map<int, string> respuestas: Almacena las respuestas del sospechoso a las preguntas.  
  
Métodos:  
- público:  
 - Sospechoso(string nombre): Constructor.  
 - string darRespuesta(int pregunta): Devuelve la respuesta del sospechoso.  
 - void mostrarInformacion(): Muestra la información del sospechoso.  
 - void marcarComoSospechosoPrincipal(bool esPrincipal): Marca al sospechoso como principal.

## Clase Ubicacion

Atributos:  
- privado:  
 - string nombre: Nombre de la ubicación.  
 - bool visitada: Indica si ya ha sido visitada.  
  
Métodos:  
- público:  
 - Ubicacion(string nombre): Constructor.  
 - void explorarUbicacion(): Permite explorar la ubicación.  
 - void marcarComoVisitada(): Marca la ubicación como visitada.  
 - vector<string> mostrarObjetosInteractivos(): Muestra objetos disponibles.

## Clase ObjetoInteractivo (Abstracta)

Atributos:  
- privado:  
 - string nombre: Nombre del objeto.  
  
Métodos:  
- público:  
 - ObjetoInteractivo(string nombre): Constructor.  
 - virtual void interactuar() = 0: Método abstracto para interactuar con el objeto.  
 - void resaltar(): Resalta el objeto al acercarse.  
 - void guardarEnInventario(): Almacena el objeto en el inventario.

## Clase Dialogo

Atributos:  
- privado:  
 - map<int, string> respuestasCorrectas: Relación de preguntas con respuestas correctas.  
  
Métodos:  
- público:  
 - Dialogo(): Constructor.  
 - void mostrarOpciones(): Muestra las opciones de preguntas disponibles.  
 - bool verificarRespuesta(int idPregunta, string respuestaJugador): Compara la respuesta del jugador con la correcta.  
 - void mostrarSiguientePregunta(): Muestra la siguiente pregunta en el diálogo.  
 - void guardarRespuesta(): Almacena la respuesta dada por el jugador para análisis posterior.

Actualización de estado

Tras realizar una sesión de retroalimentación durante las clases teóricas, se decidió reformar la estructura del diagrama UML y las relaciones entre las clases ya establecidas. A continuación, se detallan los cambios realizados:

**Cambios Realizados:**

* **Reestructuración del Diagrama UML**: Se actualizó la estructura y las relaciones entre las clases en el diagrama UML para reflejar mejor la lógica del juego.
* **Eliminación de la Clase Ubicación**: Se decidió que cada nivel tendrá su ubicación específica predefinida, eliminando la necesidad de una clase separada.
* **Modificaciones de Variables**: Se realizaron ajustes en las variables de algunas clases para mejorar la cohesión y funcionalidad.
* **Creación de Nuevas Clases**: Se añadieron las clases Personaje e Inventario para una mejor gestión de los elementos del juego.
* **Nuevas Relaciones Entre Clases**:
  + Se estableció una relación entre Inventario y Pista.
  + Se definieron relaciones entre Pista y Sospechoso.
  + Se establecieron vínculos entre Sospechoso, NivelEntrevista y NivelResolucion.

En los próximos días, los miembros del equipo se dedicarán a la implementación y ejecución de las clases con sus respectivos atributos y métodos, asegurándose de que las clases funcionen en conjunto. Dado que esta es una versión de prueba, los atributos y métodos están sujetos a cambios según sea necesario.