Este código es una implementación básica de un motor de búsqueda que utiliza el modelo de ponderación TF-IDF para calcular la relevancia de los documentos en función de una consulta de búsqueda. A continuación, se describe algunas líneas y métodos del código:

```csharp
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text.RegularExpressions;
```

Se importan los espacios de nombres necesarios para leer archivos, trabajar con listas, expresiones regulares, entre otros.

```csharp
namespace MoogleEngine
{
    class ElAprobado
    {
```

Se define un espacio de nombres y una clase ElAprobado que contiene los métodos necesarios para realizar la búsqueda.

```csharp
        static (string[], string[]) LeerArchivosDeTexto()
        {
            string rutaCarpeta = Path.Combine(Directory.GetParent(".").ToString(), "Content");
            // Buscar todos los archivos de texto en la carpeta especificada
            string[] archivos = Directory.GetFiles(rutaCarpeta, "*.txt");
            // Crear un array para el contenido de cada archivo
```

```
        string[] contenidoArchivos = new string[archivos.Length];
        // Leer el contenido de cada archivo y almacenarlo en el array correspondiente
        for (int i = 0; i < archivos.Length; i++)
        {
            contenidoArchivos[i] = File.ReadAllText(archivos[i]);
        }
        return (contenidoArchivos, archivos);
    }
```

Este método lee los archivos de texto ubicados en la carpeta "Content" y devuelve un array con el contenido de cada archivo y otro con los nombres de los archivos.

```csharp
    static string[] Normalizar(string documento)
    {
        // Crear una lista para almacenar las palabras normalizadas
        List<string> palabras = new List<string>();
        // Separar el texto en palabras utilizando expresiones regulares
        string[] palabrasTexto = Regex.Split(documento, @"\W+");
        // Normalizar cada palabra a minúscula y agregarla a la lista de palabras
        foreach (string palabra in palabrasTexto)
        {
            if (!string.IsNullOrWhiteSpace(palabra))
            {
                palabras.Add(palabra.ToLower());
            }
        }
        // Convertir la lista de palabras en un array y devolverlo
        return palabras.ToArray;
```

```
    }
```

Este método normaliza el contenido de un documento de texto, eliminando signos de puntuación y convirtiendo todas las palabras a minúsculas.

```csharp
        static string[] ObtenerPalabras(string nombreArchivo)
        {
            // Leer el contenido del archivo
            string contenido = File.ReadAllText(nombreArchivo);

            // Normalizar el texto y separarlo en palabras
            string[] palabras = Regex.Split(contenido.ToLower(), @"\W+");

            return palabras;
        }
```

Este método obtiene las palabras de un archivo de texto, normalizando el contenido del archivo y separándolo en palabras.

```csharp
        static double[] CalcularVectorTfIdf(string[] palabras, Dictionary<string, List<string>> frecuencia, string[] array)
        {
            double[] vector = new double[palabras.Length];
            for (int i = 0; i < palabras.Length; i++)
            {
                string palabra = palabras[i];
                double tf = 0;
                double idf = 0;
```

```
                int frecuenciaEnArray = array.Count(p => p == palabra);
                vector[i] = frecuenciaEnArray;
                if (frecuenciaEnArray > 0)
                {
                    tf = (double)frecuenciaEnArray / array.Length;
                    idf = Math.Log((double)array.Length / frecuencia[palabra].Count);
                }
                vector[i] = tf * idf;
            }
            return vector;
        }
```

Este método calcula el vector TF-IDF de un conjunto de palabras, utilizando la frecuencia de las palabras en el conjunto y en la colección de documentos.

```csharp
        static double[] CalcularVectorTfIdfQuery(string[] palabras, Dictionary<string, List<string>> frecuencia, string[] array)
        {
            double[] vector = new double[matriz.GetLength(1)];
            for (int i = 0; i < palabras.Length; i++)
            {
                string palabra = palabras[i];
                int index = frecuencia.Keys.ToList().IndexOf(palabra);
                vector[index] = array.Count(p => p == palabra);
            }
            return vector;
        }
```

Este método calcula el vector TF-IDF de una consulta, utilizando la frecuencia de las palabras en la consulta y en la colección de documentos.

```csharp
        static double CalcularSimilitudCoseno(double[] vector1, double[] vector2)
        {
            double productoPunto = 1;
            double magnitud1 = 1;
            double magnitud2 = 1;
            for (int i = 0; i < vector1.Length; i++)
            {
                productoPunto += vector1[i] * vector2[i];
                magnitud1 += vector1[i] * vector1[i];
                magnitud2 += vector2[i] * vector2[i];
            }
            magnitud1 = Math.Sqrt(magnitud1);
            magnitud2 = Math.Sqrt(magnitud2);
            return productoPunto / (magnitud1 * magnitud2);
        }
```

Este método calcula la similitud coseno entre dos vectores.empezando desde 1 para a la hora de la división en caso de q alguna magnitud es 0 , no de error el programa al no poder realizar la división.

```csharp
        static string[] ObtenerPalabrasQuery(string query)
        {
            // Normalizar el texto y separarlo en palabras
            string[] palabras = Regex.Split(query.ToLower(), @"\W+");
```

```
        return palabras;
    }
```

Este método obtiene las palabras de una consulta, normalizando el contenido de la consulta y separándolo en palabras.

```csharp
    static double[,] CalcularMatrizTfIdf(string[] contenido, string[] nombres)
    {
        // Crear la matriz de TF-IDF
        Dictionary<string, List<string>> frecuencia = new Dictionary<string, List<string>>();
        for (int i = 0; i < nombres.Length; i++)
        {
            string[] palabrasEnDocumento = Normalizar(File.ReadAllText(nombres[i]));
            foreach (string palabra in palabrasEnDocumento)
            {
                if (!frecuencia.ContainsKey(palabra))
                {
                    frecuencia[palabra] = new List<string>();
                }
                if (!frecuencia[palabra].Contains(nombres[i]))
                {
                    frecuencia[palabra].Add(nombres[i]);
                }
            }
        }

        double[,] matriz = new double[nombres.Length, frecuencia.Count];
        for (int i = 0; i < nombres.Length; i++)
```

```
            {
                double[] vectorDocumento = CalcularVectorTfIdf(frecuencia.Keys.ToArray(), frecuencia, Normalizar(contenido[i]));
                for (int j = 0; j < frecuencia.Count; j++)
                {
                    matriz[i, j] = vectorDocumento[j];
                }
            }

            return matriz;
        }
```

Este método `CalcularMatrizTfIdf(string[] contenido, string[] nombres)` recibe dos arreglos: `contenido` y `nombres`. El arreglo `nombres` contiene los nombres de los documentos y el arreglo `contenido` contiene el contenido de cada documento.

- `Dictionary<string, List<string>> frecuencia = new Dictionary<string, List<string>>();`: Se crea un diccionario `frecuencia` que se utilizará para almacenar la frecuencia de cada palabra en cada documento.

- `for (int i = 0; i < nombres.Length; i++)`: Se itera sobre cada documento.

- `string[] palabrasEnDocumento = Normalizar(File.ReadAllText(nombres[i]));`: Se lee el contenido del documento correspondiente y se normaliza para obtener una lista de palabras.

- `foreach (string palabra in palabrasEnDocumento)`: Se itera sobre cada palabra del documento.

- `if (!frecuencia.ContainsKey(palabra))`: Si la palabra no está en el diccionario `frecuencia`, se agrega como una nueva clave con una lista vacía como valor.

- `if (!frecuencia[palabra].Contains(nombres[i]))`: Si el nombre del documento no está en la lista de documentos asociada a la palabra en el diccionario `frecuencia`, se agrega a la lista.

- `double[,] matriz = new double[nombres.Length, frecuencia.Count];`: Se crea una matriz `matriz` de tamaño `nombres.Length` por `frecuencia.Count` que contendrá los valores de TF-IDF.

- `double[] vectorDocumento = CalcularVectorTfIdf(frecuencia.Keys.ToArray(), frecuencia, Normalizar(contenido[i]));`: Se calcula el vector de TF-IDF para el documento actual llamando al método `CalcularVectorTfIdf()`.

- `for (int j = 0; j < frecuencia.Count; j++)`: Se itera sobre cada término en el diccionario de frecuencia.

- `matriz[i, j] = vectorDocumento[j];`: Se asigna el valor de TF-IDF correspondiente a cada término en la matriz de TF-IDF para el documento actual.

- `return matriz;`: Se devuelve la matriz de TF-IDF resultante.

```

Este código realiza una búsqueda de palabras clave en una lista de documentos utilizando la técnica de TF-IDF y la similitud de coseno.

```
// Leer los archivos de texto de la carpeta "content"
    (string[] contenidoTupla, string[] nombresTupla) = LeerArchivosDeTexto();
    contenido= contenidoTupla;
    nombres= nombresTupla;
    // Calcular la matriz de TF-IDF
      matriz = CalcularMatrizTfIdf(contenido, nombres);

}

public static SearchItem[] Busqueda(string query)
{
```
```

```
// Realizar la consulta
Dictionary<string, List<string>> frecuencia = new Dictionary<string, List<string>>();
double[] similitudes = CalcularSimilitudQuery(query, nombres, contenido, matriz, frecuencia);

// Ordenar los resultados por similitud descendente
//Array.Sort(similitudes, nombres);
Array.Sort(similitudes);
Array.Reverse(nombres);
Array.Reverse(similitudes);

SearchItem[] resultado = new SearchItem[similitudes.Length];

// Mostrar los documentos más similares a la consulta
            for (int i = 0, j = 0; i < nombres.Length; i++)
        {
            System.Console.WriteLine(similitudes[i]);
              if (similitudes[i] >= 0) // Se agregó la condición j < 10 para mostrar solo las 10 primeras palabras
               {
            string[] palabras = ObtenerPalabras(nombres[i]);
            string primerasPalabras = String.Join(" ", palabras.Take(10)); // Se agregó esta línea para obtener las 10 primeras palabras

            resultado[j] = new SearchItem(nombres[i], primerasPalabras, (float) similitudes[i]); // Se agregó el parámetro primerasPalabras para guardar las 10 primeras palabras
            j++;
          }
return resultado;
}
```

- `Busqueda(string query)`: Este método recibe una consulta, calcula la similitud de coseno entre la consulta y los documentos, los ordena por similitud descendente y devuelve un arreglo de objetos `SearchItem` que contiene información sobre los documentos más similares a la consulta. Cada objeto `SearchItem` contiene el nombre del documento, las primeras 10 palabras del contenido y la similitud de coseno correspondiente.