A szakdolgozatom fő programja a MovieRecommendationSystem program, amely egy grafikus felületű C# alkalmazás. A program célja, hogy a felhasználó számára egy olyan filmet ajánljon, ami az ő érdeklődési körébe esik, az ő igényeinek felel meg. A felhasználó érdeklődési körének felmérése céljából a program indításkor kérdéseket tesz fel a felhasználónak, amelyek megválaszolása után tudja meghatározni az alkalmazás az ajánlott filmet. A program szintén a movies.db adatbázisból dolgozik, így ezt használja az ajánlott film kiválasztásához is.

Movie osztály

Mivel a filmek pontos osztályozásához, majd ajánlásához minél több információra van szükség, így az összes rendelkezésünkre álló adatra szükség van a filmek esetén. Így már nem csak a lebontandó mezőkhöz kapcsolódó adattagokat kell tartalmaznia a Movie osztálynak, hanem a film összes tulajdonságához szükség van egy adattagra, amelyben eltároljuk az adott információt. A lebontott adatokat táróló adattagok listák lesznek, hiszen ebben az esetben több ugyanolyan típusú információt is el kell tárolnunk egy adott film esetében, a többi adattag viszont sima változó lesz.

internal class Movie

{

public int Id { get; set; }

public string Title { get; set; }

public int Released { get; set; }

public int Runtime { get; set; }

public int GenderOfProtagonist { get; set; }

public string MainActor { get; set; }

public double TmdbScore { get; set; }

public int NumberOfRatings { get; set; }

public double Popularity { get; set; }

public long Budget { get; set; }

public long Revenue { get; set; }

/// <summary>

/// A kapcsolótáblák adatait felhasználva ezekbe az adattagokba kerülnek az egyes adatokat, pl. műfajokat azonosító integerek (id-k)

/// </summary>

public List<int> Genre { get; set; }

public List<int> Keyword { get; set; }

public List<int> Director { get; set; }

public List<int> Language { get; set; }

public List<int> ProductionCountry { get; set; }

/// <summary>

/// Az azonosító integereket (id)-kat felhasználva ezekbe a string listákba kerülnek a filmekhez kapcsolódó tényleges adatok

/// </summary>

public List<string> GenreString {get; set;}

public List<string> KeywordString {get; set;}

public List<string> LanguageString {get; set;}

public List<string> DirectorString {get; set;}

public List<string> CountryString {get; set;}

/// <summary>

/// Az előző listák felhasználásával a listákban szereplő adatok vesszővel történő összefűzése táblázatos megjelenítéshez

/// </summary>

public string GenreStringWithCommas => string.Join(", ", GenreString);

public string KeywordStringWithCommas => string.Join(", ", KeywordString);

public string LanguageStringWithCommas => string.Join(", ", LanguageString);

public string DirectorStringWithCommas => string.Join(", ", DirectorString);

public string CountryStringWithCommas => string.Join(", ", CountryString);

}

SqlConnector osztály

Mivel a program adatbázisból dolgozik, így ugyancsak meg kell teremteni a kapcsolatot az adatbázissal, ami az SqlConnector osztályban valósul meg. Emellett itt történik az adatbázis movie táblájából az egyszerű, felsorolást nem tartalmazó mezők adatainak betöltése.

Az egyszerű adatok mellett viszont szükséges a TableInserts programmal már előkészített, lebontott formába került adatok beolvasása és eltárolása is, az osztály minden ilyen adattaghoz tartalmaz egy Fillup metódust, mint pl. a FillupGenre.

/// <summary>

/// A korábbiakban feltöltött Id adattag és a Movies\_Genres kapcsolótábla segítségével a Genre adattag feltöltésre kerül, itt még a műfajokat azonosító integerekkel

/// </summary>

/// <param name="movies">A movies listában tároljuk el a filmek műfajait</param>

public void FillupGenre(ref List<Movie> movies)

{

int index;

int data;

SQLiteDataReader reader = null;

SQLiteCommand command = connection.CreateCommand();

command.CommandText = "SELECT \* FROM Movies\_Genres";

reader = command.ExecuteReader();

while (reader.Read())

{

index = Convert.ToInt32(reader["Movie\_ID"]);

data = Convert.ToInt32(reader["Genres\_ID"]);

for (int i = 0; i < movies.Count; i++)

{

if (movies[i].Id == index)

{

movies[i].Genre.Add(data);

}

}

}

reader.Close();

}

A kapcsolótáblában lévő, jelen esetben a műfajokat azonosító integerek megfelelőek arra, hogy hozzárendeljük a filmekhez a különböző adatokat, műfajokat, viszont a megjelenítésükkel nem derül ki, hogy konkrétan melyik műfajról van szó. Tehát szükség van egy metódusra, amely minden filmhez hozzárendeli az azonosítókon kívül a tényleges adatot is, aktuálisan a műfajok nevét. Az osztály minden ilyen adattaghoz tartalmaz egy ilyen metódust, mint jelen esetben a műfajokhoz a GetGenre.

/// <summary>

/// A korábbiakban megkapott integerek felhasználásával hozzárendeljük a műfajok nevét is a filmekhez

/// </summary>

/// <param name="movies">A movies lista megfelelő adattagjában fogjuk eltárolni a műfajok tényleges neveit is</param>

/// <param name="tableID">A műfajok integer azonosítót tároljuk benne</param>

/// <param name="tableData">A műfajok neveit tároljuk benne</param>

public void GetGenre(ref List<Movie> movies, List<int> tableID, List<string> tableData)

{

SQLiteDataReader reader = null;

SQLiteCommand command = connection.CreateCommand();

command.CommandText = "SELECT \* FROM Genres";

reader = command.ExecuteReader();

while (reader.Read())

{

tableID.Add(Convert.ToInt32(reader["ID"]));

tableData.Add(reader["Genre\_Name"].ToString());

}

reader.Close();

Algorithms alg = new Algorithms();

int mode = (int)Mode.Genre; //enumot használva beállítjuk a mode változó értékét Genre-re, amit castolással integerré alakítunk

alg.tableFiller(ref movies, tableID, tableData, mode); //a tableFiller metódusnak átadjuk a műfajok azonosítóit és a műfajokat is, a mode változó értékébőúl pedig tudni fogja hogy a movies lista melyik adattagját kell feltölteni a kapott adatokkal

}

Algorithms osztály

A Movie típusú movies lista azon adattagjait, amelyek nem sima változók, hanem listák, inicializálni kell, ez az Algorithms osztályban történik.

A korábbiakban említett tableFiller metódus is ebbe az osztályba került, ez dönti el, hogy a movies lista melyik adattagját töltsük fel. Az algoritmus használja a szintén az osztályban lévő enum típusú Mode-ot, melynek a swich case eseteinek azonosításában van szerepe.

public void tableFiller(ref List<Movie> movies, List<int> tableID, List<string> tableData, int mode)

{

switch(mode)

{

case (int)Mode.Genre:

GenreToString(ref movies, tableID, tableData);

break;

case (int)Mode.Keyword:

KeywordToString(ref movies, tableID, tableData);

break;

case (int)Mode.Language:

LanguageToString(ref movies, tableID, tableData);

break;

case (int)Mode.Director:

DirectorToString(ref movies, tableID, tableData);

break;

case (int)Mode.Country:

CountryToString(ref movies, tableID, tableData);

break;

}

}

A tableFiller minden adattag esetében másik metódust hív meg, a műfajok feltöltése esetében a GenreToString metódust.

/// <summary>

/// A GenreString adattag tényleges feltöltése az egyes filmek esetén

/// </summary>

/// <param name="movies">A movie lista GenreString adattagjába kerülnek az adatok</param>

/// <param name="tableID">A Genres táblából származó műfaj azonosítók (id-k)</param>

/// <param name="tableData">A Genres táblából származó műfajok tényleges nevei</param>

public void GenreToString(ref List<Movie> movies, List<int> tableID, List<string> tableData)

{

for (int i = 0; i < movies.Count; i++)

{

for (int j = 0; j < movies[i].Genre.Count; j++)

{

for (int k = 0; k < tableID.Count; k++)

{

if (movies[i].Genre[j] == tableID[k])

{

movies[i].GenreString.Add(tableData[k]);

}

}

}

}

tableID.Clear();

tableData.Clear();

}

A filmeken végigmegyünk, majd az egyes filmek minden egyes műfajának adatait a Genres adattagból (amik egyelőre id-k) összehasonlítjuk a tableID lista minden elemével. Ha egyezés van akkor mivel a tableID és tableData listákban párhuzamos indexeléssel vannak letárolva az adatok, ezért az az adott filmhez tartozó GenreString adattaghoz hozzáadjuk a tableData lista adott indexű elemét.

HandmadeLanguageDecTree osztály

A program tartalmaz egy programrészt egy kézzel készített, a film nyelvére vonatkozó döntési fa megjelenítésére is. Ennek megvalósítására ebben az osztályban kerül sor.

A döntési fa kirajzolásának alapja a ParseNode metódus, amely csomópontonként dolgozza fel a fát tartalmazó json forrásfájlt.

/// <summary>

/// Rekurzív metódus, amellyel a JSON fájlból a döntési fa csomópontonként kerül feldolgozásra

/// </summary>

/// <param name="dot">A string lista, amiből majd a dot fájl készül</param>

/// <param name="node">Az éppen aktuálisan feldolgozott csomópont</param>

/// <param name="parentName">Az adott csomópont szülője</param>

static void ParseNode(List<string> dot, JToken node, string parentName)

{

if (node.Type == JTokenType.Object)

{

var conditionArray = node["condition"] as JArray;

if (conditionArray != null)

{

foreach (var conditionObject in conditionArray)

{

var condition = conditionObject["condition"]?.ToString(); // Lekéri a feltétel szövegét

var outcome = conditionObject["outcome"]; // Lekéri a kimeneti objektumot

if (condition != null && outcome != null)

{

if (outcome.Type == JTokenType.Object) // Ha a kimenetel objektum, akkor egy új csomópont

{

string nodeName = Guid.NewGuid().ToString();

dot.Add($" \"{parentName}\" -> \"{nodeName}\" [label=\"{condition}\"];");

dot.Add($" \"{nodeName}\" [label=\"{outcome["node"]}\"];");

ParseNode(dot, outcome, nodeName);

}

else if (outcome.Type == JTokenType.Array) // Ha a kimenet egy tömb, akkor minden elemet külön kezel

{

foreach (var item in outcome)

{

string leafName = Guid.NewGuid().ToString();

dot.Add($" \"{parentName}\" -> \"{leafName}\" [label=\"{condition}\"];");

dot.Add($" \"{leafName}\" [label=\"{item}\", shape=box];");

}

}

else

{

string leafName = Guid.NewGuid().ToString();

dot.Add($" \"{parentName}\" -> \"{leafName}\" [label=\"{condition}\"];");

dot.Add($" \"{leafName}\" [label=\"{outcome}\", shape=box];");

}

}

}

}

}

else if (node.Type == JTokenType.Array) //Ha az aktuális csomópont tömb, akkor szintén minden elemet külön dolgoz fel

{

foreach (var item in node)

{

ParseNode(dot, item, parentName);

}

}

else

{

string leafName = Guid.NewGuid().ToString();

dot.Add($" \"{parentName}\" -> \"{leafName}\" [label=\"\"];");

dot.Add($" \"{leafName}\" [label=\"{node}\", shape=box];");

}

}

A ParseNode metódus a MainJson metódusból kerül meghívásra. Ebben a metódusban történnek az alapvető változó deklarációk, a json forrásfájl beolvasása, json objektummá való átalakítása, illetve a dot string típusú lista létrehozása is, amiből majd kezdetben a .dot fájl készül, később pedig ebből a megjelenítendő .png kép.

/// <summary>

/// A döntési fa felépítéséhez szükséges változók létrehozása, függvények meghívása

/// </summary>

public void MainJson()

{

string json = CreateJsonPath();

json = File.ReadAllText(json);

JObject jsonObj = JObject.Parse(json);

var dot = new List<string>

{

"digraph DecisionTree {"

};

dot.Add($" \"root\" [label=\"{jsonObj["decision\_tree"]["root"]["node"]}\"];");

ParseNode(dot, jsonObj["decision\_tree"]["root"], "root");

dot.Add("}");

File.WriteAllLines("decision\_tree.dot", dot); //A dot string lista tartalmából készít egy .dot fájlt

ProcessStartInfo startInfo = new ProcessStartInfo //Graphviz parancsok amelyek segítségével a .dot fájlból .png kép készül

{

FileName = "dot",

Arguments = "-Tpng decision\_tree.dot -o decision\_tree.png",

RedirectStandardOutput = true,

UseShellExecute = false,

CreateNoWindow = true

};

using (Process process = Process.Start(startInfo)) //A folyamat indítása

{

process.WaitForExit();

}

PictureBox pictureBox = new PictureBox //PictureBox használata a kép megjelenítéséhez

{

Dock = DockStyle.Fill,

Image = Image.FromFile("decision\_tree.png"),

SizeMode = PictureBoxSizeMode.Zoom

};

this.Controls.Add(pictureBox);

}

Az osztály tartalmaz még egy CreateJsonPath metódust, ez a forrásfájl helyét határozza meg, amely szükséges a beolvasáshoz.

Main osztály

Végül a program fő osztálya a Main, ami a program fő ablakát jeleníti meg.

Az osztály fő metódusa a Main\_Load, ami akkor fut le amikor az ablak megjelenítésre kerül, tehát mivel ez a fő ablak, így gyakorlatilag a program indulásakor. Ebben a metódusban történik a kapcsolódás az adatbázissal, a korábban bemutatott metódusok meghívása, az adatok kiolvasása az adatbázisból, illetve a korábban lebontott mezők adatainak tényleges hozzárendelése a filmekhez. A táblázatos megjelenítés is ebben az osztályban valósul meg, illetve a korábbiakban említett kézzel készített döntési fa megjelenítése is.

A Main osztály másik metódusa a CreatePath építi fel az adatbázis eléréséhez szükséges utat, ezzel elkerülhető az abszolut útvonal használata. Ez a metódus is a Main\_Load-ból kerül meghívásra.