EÖTVÖS LORÁND TUDOMÁNYEGYETEM

Informatikai kar

Programozási nyelvek és fordítóprogramok tanszék

Fájlkonverziós alkalmazás C++ nyelven

Témavezető:

Pataki Norbert

Egyetemi docens, PhD

Szerző:

Rimóczi Loránd

Programtervező informatikus BSc

Budapest, 2023

Tartalomjegyzék

1. Bevezetés

1.1 Motiváció

Az egyetemi tanulmányaim során rengeteg programozási nyelvvel, illetve azoktól független, de mégis a programozáshoz kapcsolódó tudásanyaggal ismerkedhettem meg, amely egyfelől jó dolog egy minél szélesebb látókör kialakításához, másfelől viszont kicsit össze is tud zavarni, elveszhet köztük az ember. Velem is ez történt, így amikor a szakdolgozat elkészítésének feladat a látótérbe került, nem is tudtam, pontosan mivel is szeretnék foglalkozni. Szerencsémre egy nagyon motiváló C++ előadás után úgy éreztem, még ha nehéz is, ebben a témakörben szeretnék elmélyedni. Ezután még kérdéses volt, hogy mit is lehetne készíteni, de ez is gyorsan megoldódott, eredménye pedig egy olyan program ötlete lett, melynek rálátásom szerint valódi haszna is lehet, létező hiányt pótolhat.

A fejlődő világban, ezáltal az oktatásban is minden egyre inkább számítógépen, digitálisan zajlik, kiemelten igaz volt ez a COVID idejére. Ekkor az egyetemen is minden számonkérés online környezetben zajlott. A jellemzően matematikai, valamint számításelméleti tárgyaknál a zárthelyi dolgozatokat ekkor is legkényelmesebben papírra írva lehetett megoldani, azonban ezeket utána szkennelve, fotózva kellett feltölteni. Az ezt kezelő rendszer kapcsán azonban hamar előjött a kicsi, de annál bosszantóbb hiba, mégpedig, ha a fájlformátum, amibe szkenneltünk, vagy fotóztunk nem volt elfogadott.

Ekkor a legtöbben valamilyen online fájlkonvertert kerestünk, természetesen kapkodva, és reméltük, hogy a talált oldal enged annyit konvertálni ingyenesen, ahány oldalt írtunk. Ezek után pedig néha hibázva váltogattunk a programablakok közt, hisz egyszer van a felület, ahova fel kell töltenünk, van a fájlkezelő, ahonnan jó esetben drag & drop módszerrel behúzhatjuk a fájlt, és persze van a konvertálást intéző weboldal. Hasonló szituációk természetes előkerülnek a hétköznapokban is, például amikor valamilyen hivatalos kérvényt kell adott oldalra feltölteni, a kép mellékletet persze csak PNG formátumban fogadja, míg nekünk JPG formátumban van.
A különböző online konvertálós felületek gyakran eltűnnek, mire újra használnánk, vagy a szükségesnél kevesebbet tudunk ingyenesen konvertálni, a kívánt mennyiségért már fizetni kell.

Erre szeretnék megoldást kínálni egy olyan fájlkezelő alkalmazással, amely beépítve tartalmaz konvertálási lehetőséget a népszerűbb kép, szöveg, valamint e-book formátumok között. Egyrészt így nem kell mindig keresni egy új weboldalt, és izgulni, hogy tudunk-e annyit konvertálni, amennyit szeretnénk, másrészt egy programablakban elvégezhető a konvertálás és az általános fájlkezelés is, mint például másolás, átnevezés vagy törlés.

1.2 A konvertálásról pár szóban

Sokat emlegettem a konvertálást, de jogos a kérdés, mi is ez pontosan. Mint ismert, a számítógépek mindent 1-es és 0-ás számok sorozataként tárolnak, majd ezeket feldolgozva jelenítenek meg bármit is az ember számára értelmezhető módon. Ebbe tartoznak bele a fájlok is, amik nekünk képek, zenék, szöveges dokumentumok, vagy éppen forrás kódok, a számítógépen csupa 1 és 0.

Ahhoz, hogy ezek elkülöníthetőek legyenek, a fájlformátumok speciális algoritmusokként foghatóak fel, amelyek leírják, hogy egy adott 1-es és 0-s számsorozat, hogy dekódolható emberi fogyasztásra alkalmas formátumba.
Különböző igények miatt, például inkább férjen el kisebb tárterületen, vagy inkább legyen minél részletgazdagabb, veszteségmentesebb (az analóg adatok, például hanghullám digitális formába történő mentésekor az eredeti, analóg adat egy része elvész). Mindkét esetben ugyanolyan dolgot szeretnénk tárolni, de mégis más módon, így eltérő fájlformátumot fogunk használni. Ebből ered a probléma is, hogy egy adott program, vagy weboldal kezeli-e azt a formátumot, amelyet az adott fájl a reprezentálására használ fel.

A konvertálás folyamata ezen eltérő formátumok közt biztosít átjárást, értelemszerű keretek közt (tehát zenéből azért nem tudunk képet konvertálni, sem fordítva). Ezen folyamat gyakran külön programot, bizonyos esetekben pedig komoly számítási teljesítményt is igényel, ilyen például a videó konvertálás. Persze erre a hétköznapokban meglehetősen ritkán van szükség, azonban különböző képformátumok, illetve szöveges állományok formátumai közötti váltásra annál inkább, ennek megfelelően tűztem ki célul, hogy az általam írt alkalmazás ezen formátumok közt legyen képes a konvertálást végrehajtani.

2. Felhasználói dokumentáció

2.1 Feladat

A feladat egy olyan, alap fájlkezelési képességekkel rendelkező program, amely integrált, azaz beépített módon biztosít lehetőséget bizonyos fájltípusok közti konverzióra.
Fájlkezelési szempontból a célkitűzés a fájlok átnevezésének lehetősége, fájlok törlése, valamint fájl keresés volt. Konvertálási célok e-book, illetve kép fájlok legelterjedtebb formátumaira irányultak, úgy, mint a pdf, mobi, epub és azw3 e-bookok esetén, jpg, png és bmp képek esetén.
Szempont volt továbbá, hogy mindez valamilyen grafikus felületen jelenjen meg, így könnyebbé, kényelmesebbé téve használatát.

Ezen kitűzéseket foglalta össze az eredeti terv, miszerint legyen egy olyan alkalmazás, mely egy felületen képes megvalósítani fájlkezelést és konvertálást is, így nem kell számtalan további programablakot, felületet megnyitni, amikor esetleg csak arra van szükség, hogy a feltöltendő képfájl JPEG helyett PNG formátumban legyen. Továbbá cél még az internetes eléréstől való függetlenség, amely megnehezítené a feladat végrehajtását, amennyiben nincs internet kapcsolat, vagy épp egy konvertáláshoz használt rendszer interneten keresztül érhető el.

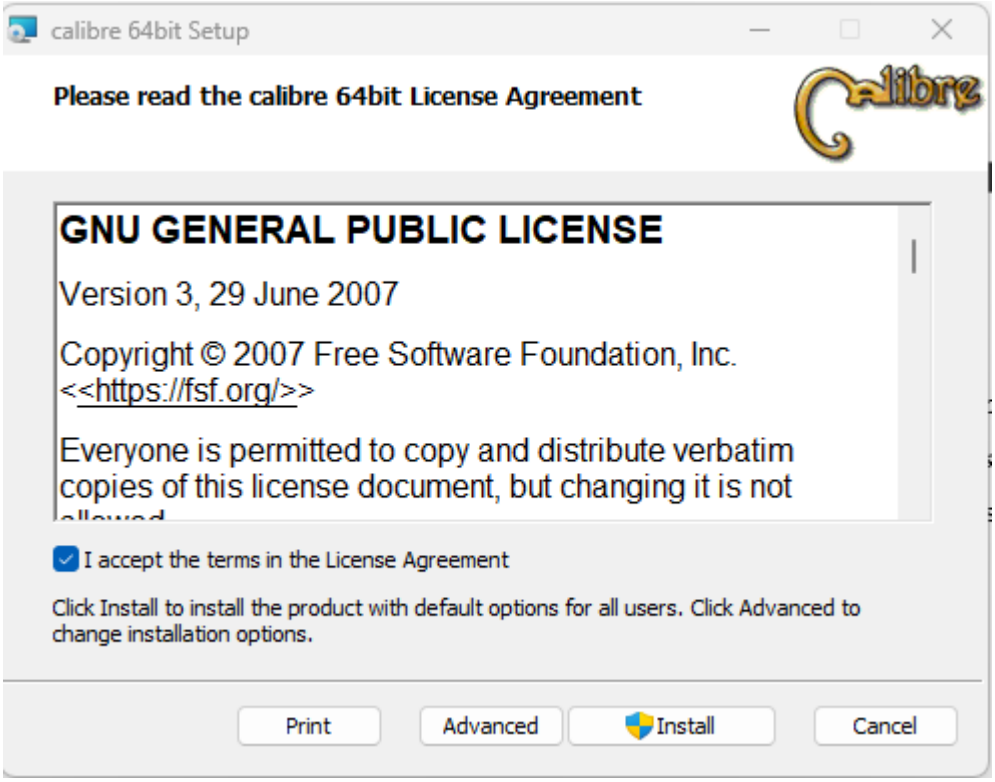
2.2 Rendszerigény, futtatás

A program lényegében bármilyen mai asztali számítógépen és laptopon futtatható, amennyiben azon Windows 10 vagy 11 64 bites, vagy linux ubuntu 64 bites operációs rendszer fut[1]. Emellett elvárás a minimum 1280x720 képernyőfelbontás, ez a program ablak beállított minimális felbontása. Természetesen nagyobb felbontású képernyőn is jól használható, kisebb esetén azonban nem minden jelenik meg helyesen.
A teljes csomag és a program megközelítőleg 650-700MB tárhelyet igényel (jelen dokumentáció írásának idejében), melyből ~80MB a program és futtatásához szükséges fájljai, ~150MB a függőségként jelenlévő Calibre telepítőjének mérete, valamint ~400MB a telepített Calibre program.

[1] A linux ubuntu operációs rendszeren való futtatáshoz a programot forráskódból build-elni kell QT Creator használatával. Részletesebb információk a Fejlesztői dokumentációban.

Fontos, hogy az e-book konvertálás használatához elengedhetetlen a Calibre e-book manager program telepítése, hiszen jelen program arra támaszkodik e-book konvertálás esetén, így enélkül a program e-bookot nem tud konvertálni.

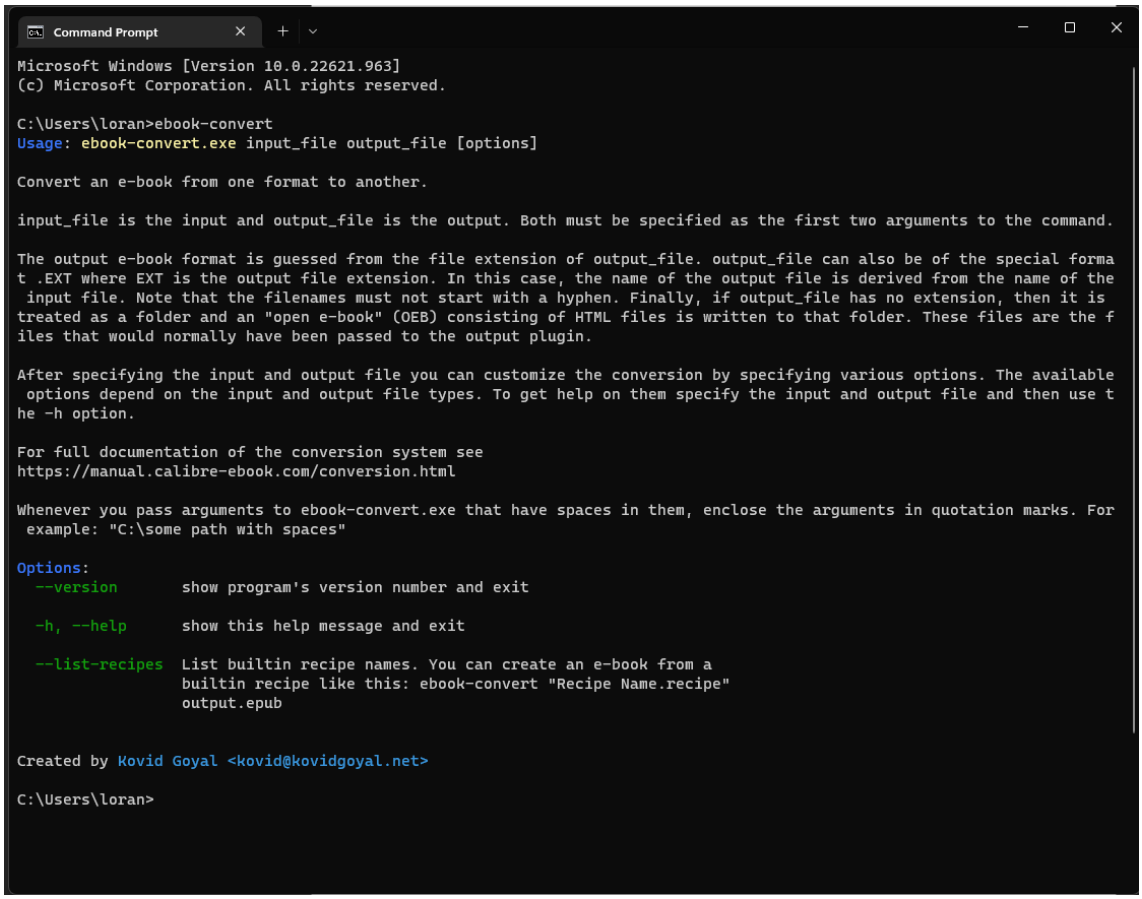
Ehhez a dokumentáció mellett megtalálható *Dependencies* mappában található *calibre-64bit-6.9.0.msi* fájl futtatásával telepíthető a Calibre program[2]. A telepítés során mindössze egy ASZF elfogadás, valamint az Install gombot kell nyomni, további beállítás, személyre szabás nem szükséges. Telepítés végén a telepítő felajánlja a Calibre futtatását, ez opcionális lehetőség.



1. ábra, Calibre telepítő

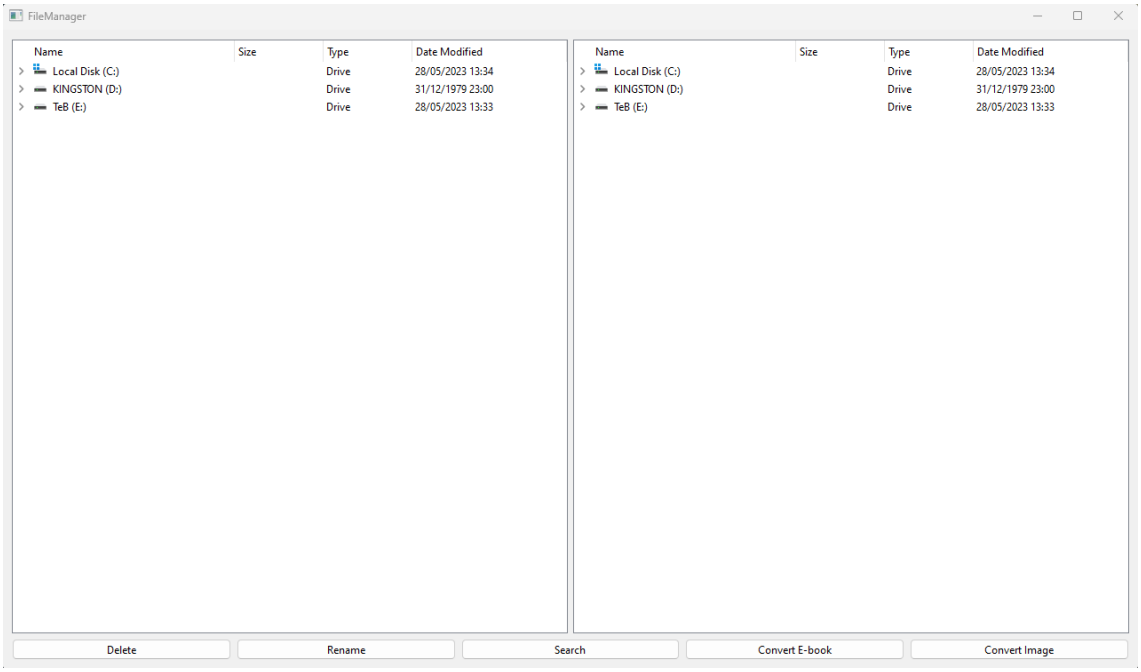
A fájlkezelő és konvertáló alkalmazás szempontjából helyes telepítés ellenőrizhető, a parancssorban kiadott ebook-convert paranccsal, ahogy a 2. ábrán látható. Amennyiben a parancs nem található, hibaüzenet fogad, úgy a telepítés megismétlendő, mivel enélkül a program nem tud e-book konverziót végezni.

[2] Ez Windows operációs rendszer esetében igaz, linux operációs rendszer esetén a hivatalos oldalon (calibre - Download for Linux (calibre-ebook.com)) található leírás alapján telepíthető.



2. ábra, Az ebook-convert parancs eredménye

Amennyiben ezen előfeltételek teljesültek, a program a *FileManagerProgram* mappában található, *FileManager.exe* fájl indításával futtatható. A mappában a program futásához szükséges állományok vannak, így a hosszabb távon történő használat kényelmesebb módja a fájlhoz történő parancsikon létrehozása a START menüben, vagy asztalon. Ez opcionális, a felhasználó preferenciája szerint tetszőlegesen alkalmazható eljárás.



3. ábra, A program felülete Window operációs rendszeren

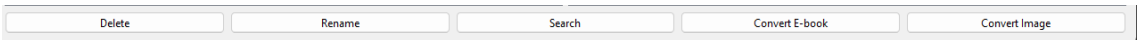
A *SourceCode* mappában találhatóak a program forrás fájljai, fordításukhoz a QT 6.4-es verziójára van szükség. Erről részletesebben a fejlesztői dokumentáció fejezetben írok.

2.3 A program funkciói, használata

A programot megnyitva a 3. ábrán látható ablakot kapjuk, ezen belül van a bal- és a jobb oldali panel, valamint az 5 funkció gomb. A felépítés letisztult: a baloldali panelen a meghajtók és a mappák láthatóak, míg a jobboldalin a baloldali meghajtóhoz, mappához tartozó fájlok. Fontos, itt csak a konkrét meghajtó/mappa fájljai vannak felsorolva, vagyis, ha például egy mappán belüli mappában lévő fájlra van szükség, akkor a baloldali panelen ki kell jelölni azt a mappát. A bal panelen a mappák és meghajtók mellett kis nyilak láthatóak, ezekkel lehet bennük lépkedni, illetve az egérrel történő duplakattintásra is „lenyílnak”.

A paneleken a felső sorban található függőleges vonalak egérrel történő mozgatásával a tulajdonságokat megjelenítő oszlopok szélessége állítható.

A 4. ábrán látható gombok a rájuk írt műveletek végrehajtásáért felelősek, kizárólag fájlok esetén, tehát például mappát nem lehet törölni, átnevezni, fájlt azonban igen. Ezen műveletek működését később részletesen kifejtem.

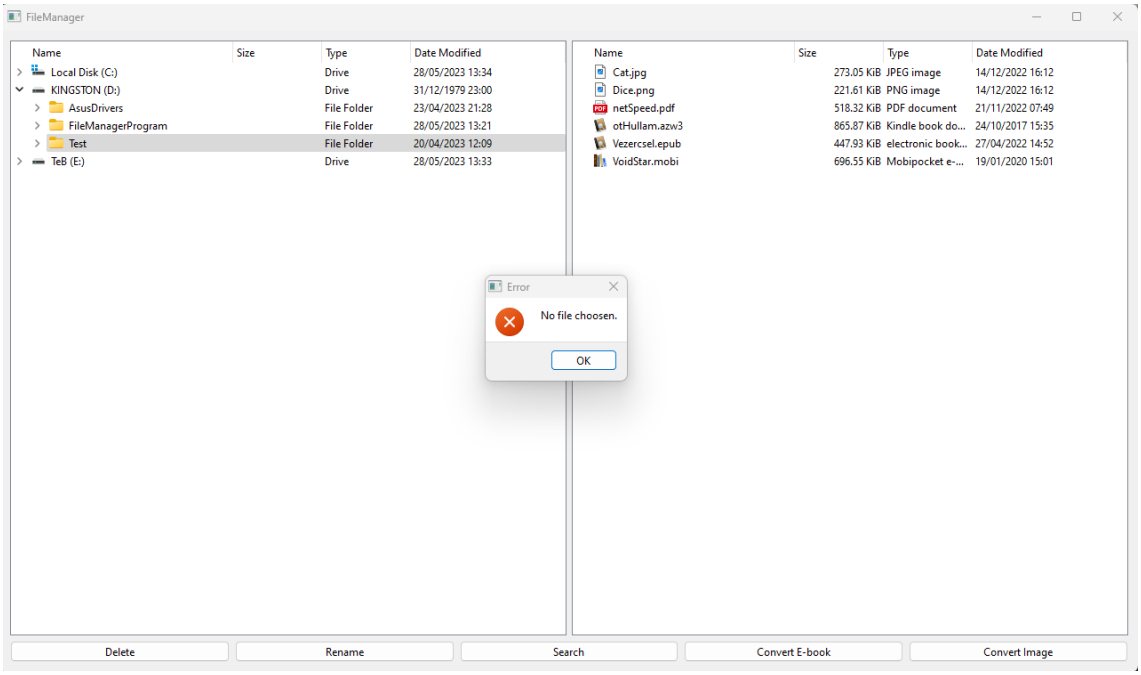


4. ábra, A gombok

A keresés kivétel minden funkció, törlés (Delete), átnevezés (Rename), e-book konvertálás (Convert E-book), és kép konvertálás (Convert Image) csak akkor használható helyesen, azaz csak akkor fogja az elvárt műveletet végezni, ha a jobboldali panelen van kijelölt fájl. Természetesen akármelyik funkció akármilyen fájllal nem működik, hiszen például videófájlt nem lehet e-book-ként konvertálni.

2.3.1 Delete gomb

A Delete gomb használatával a jobboldali panelen kijelölt fájl törlésre kerül. Ha nincs kijelölt fájl, vagy valami miatt a törlés nem sikerül, az 5. ábrán látható módon hibaüzenetet kapunk.



5. ábra, Hibaüzenet fájlkijelölés hiányában

2.3.2 Rename gomb

A Rename gomb használatával a jobboldali panelen kijelölt fájlt nevezhetjük át. A program kezeli a fájl kiterjesztését/formátumát, így a felugró ablakban elegendő a kívánt új név megadása. Windows-on kis- és nagybetű közt nincs különbségtétel, így például

egy Cat.png-t átnevezve cat.png-re a fájl neve nem változik. Amennyiben a jobboldali panelen nem volt kijelölés, vagy valami hiba miatt az átnevezés sikertelen, hibaüzenetet kapunk.

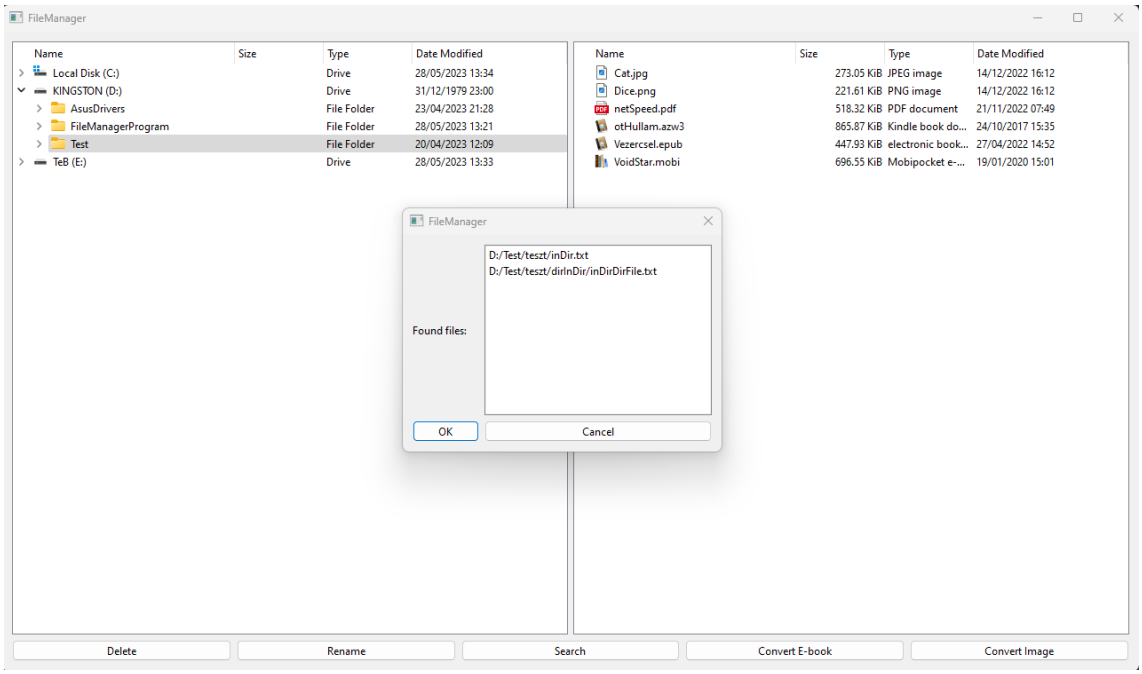
2.3.3 Search gomb

A Search gomb segítségével keresést indíthatunk. Ha a jobboldali panelen van fájl kijelölve, akkor az azt a fájlt tartalmazó mappából, ha ez nincs, de a baloldali panelen van mappa vagy meghajtó kijelölve, akkor azt használva, ha pedig egyik panelen sincs kijelölve semmi, akkor alapértelmezetten a rendszerszintű gyökértől, Windows esetén ez jellemzően a C:/, linux Ubuntu esetén jellemzően „/” útvonaltól kezdve indul el a keresés. A keresés rekurzívan, vagyis az induló mappától „befelé”, az abban a mappában lévő mappákban és így tovább, keres. Például, ha egymás mellett van A és B mappa, és B mappában indítunk keresést, akkor csak a B mappa teljes tartalmát fogja átvizsgálni, az A mappáét nem.

A kereséshez nem fontos megadni a fájl kiterjesztését, elegendő csak a keresett fájl nevét, vagy annak egy részletét megadni, a program ezután minden olyan fájlt figyelembe vesz, amelynek a teljes neve, vagy nevének egy részlete megegyezik a bemenettel.

A keresés végeztével, amennyiben egy találat van, úgy automatikusan az a fájl megjelenítésre kerül a jobb oldali panelen.

Ha több találat van, akkor a találatok egy felugró ablakban kerülnek listázásra, az elérési útjukkal. Itt egyet kiválasztva, majd az OK gombra kattintva a kiválasztott fájl megjelenítésre kerül a jobboldali panelen. Ez látható a 6. ábrán.



6. ábra, A megtalált fájlok listája

Ha a fájlkeresés eredménytelen, hiba/figyelmeztető üzenet jelenik, mely szerint nem sikerült a keresés, de meg lehet próbálni másik meghajtón, vagy hierarchiában felsőbb szinten lévő mappából indítva azt.

2.3.4 Convert Ebook gomb

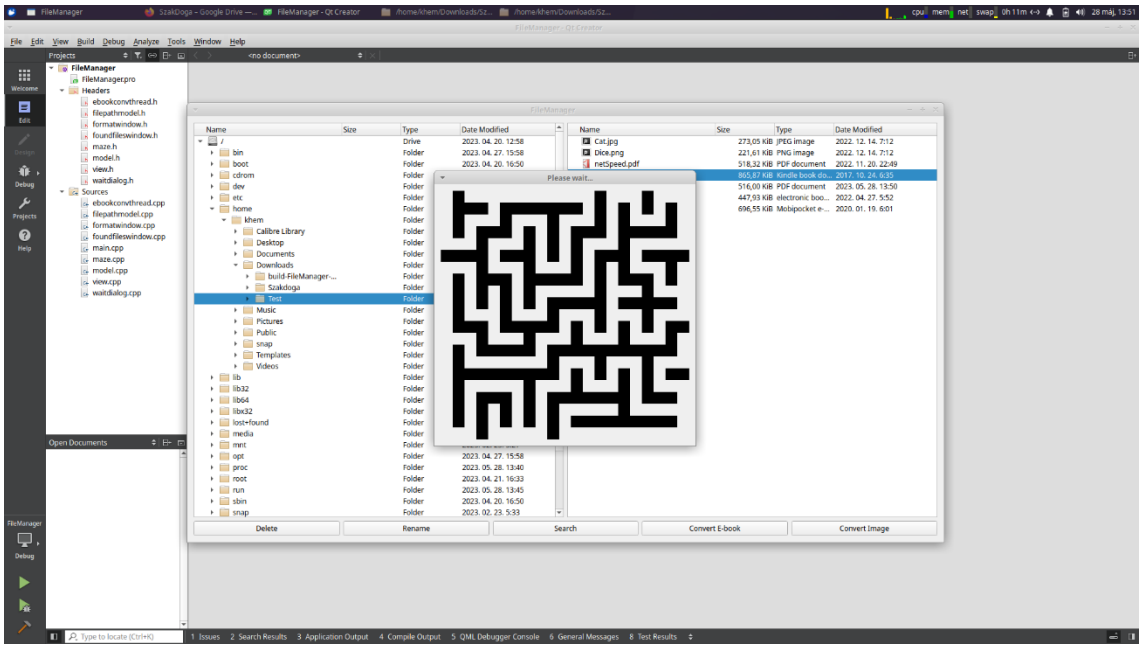
A Convert E-book gombot használva, amennyibe a kijelölt fájl egy megfelelő formátumban lévő e-book, lehetőség nyílik a konvertálásra másik formátumra.
A Delete és Rename gombok működéséhez hasonlóan itt is kötelező kijelölt fájlnak lennie, azonban ez még önmagában nem elég. Fontos, hogy a kijelölt fájl a támogatott fájlformátumok közüli, e-book formátum legyen.

Konvertáláskor csak a kívánt formátumot kell kiválasztani, az új fájl neve egyezni fog a bemeneti fájl nevével, csak a formátuma és a kiterjesztése változik.

Támogatott e-book formátumok konvertáláshoz: epub, mobi, azw3, pdf. Mind bemeneti, konvertálandó, mind kimeneti, kívánt fájl esetében ezen 4 formátummal lehet dolgozni.
Megfelelő bevitel esetén a program a korábban telepített Calibre program hívásával elvégzi a kívánt konverziót, egyéb esetben tájékoztató vagy hibaüzenetet ad. Tájékoztató

üzenet például, ha a bemeneti és a kívánt kimeneti formátum megegyezik, ilyenkor nincs értelme végrehajtani konverziót.
Hiba lehet még, ha nem támogatott kiterjesztésű fájlt akarunk konvertálni.

Mivel az e-book konvertálás időigényes is lehet, így a program egy animációt jelenít meg a konvertálás befejezéséig, amelyen másodpercenként egy labirintust generál a képernyőn. Ez az animáció automatikusan bezáródik, amint a konvertálás befejeződött. Az animációról egy kép az alábbi, 7. ábrán látható, amely linux xubuntu-n készült, de ugyanígy jelenik meg Windows esetén is.



7. ábra, Várakozó animáció, linux

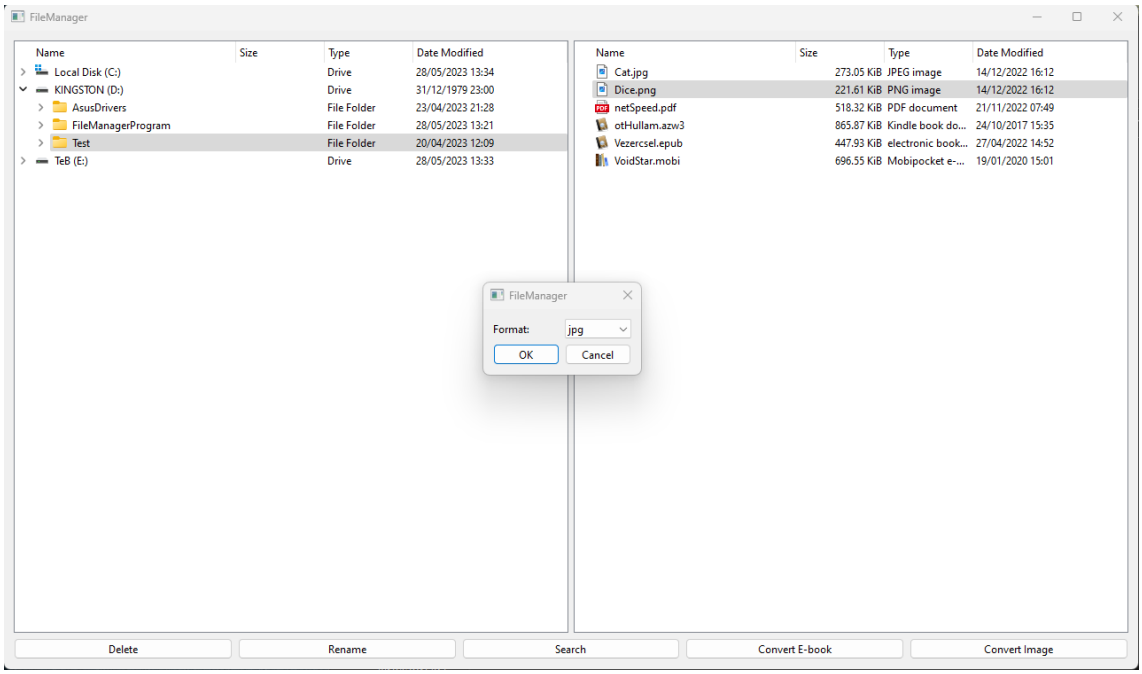
Amennyiben a Calibre előzetesen nem került telepítésre, úgy a program az e-book konvertálást nem tudja elvégezni.

2.3.5 Convert Image gomb

A convert Image gombot használva képfájlokat konvertálhatunk. Alapvetően a Convert E-book-hoz hasonló a működése, viszont itt nincs szükség előretelepített külső alkalmazásra, valamint természetesen kép fájl formátumokkal dolgozik.

Konvertáláskor csak a kívánt formátumot kell kiválasztani, az új fájl neve egyezni fog a bemeneti fájl nevével, csak a formátum változik. Erre példa a 8. ábra.

Támogatott formátumok: jpg, png, bmp. Itt is mindhárom formátum elfogadott be- és kimenet is egyben. Bemenettel egyező formátumra történő konvertálás esetén a program tájékoztat, hogy a kívánt formátumban a fájl már létezik.



8. ábra, Hibás formátum

3. Fejlesztői dokumentáció

3.1 Feladat elemzése

A feladat tehát, egy olyan, GUI-val (Graphical User Interface) rendelkező alkalmazás létrehozása, ahol egyrészt a számítógépen található fájlok megjelenítésre kerülnek, másrészt ezen fájlokkal alapvető fájlműveleteket, valamint az e-book és kép fájlok esetén konvertálást tudunk végrehajtani.

Ezen kitűzések alapján a feladat megvalósítását négy fő részre bontottam, ezek sorrendben történő megoldásával haladtam a végcél felé. Ezen négy fő rész: GUI felületben fájlok megjelenítése, alapvető fájlkezelési funkciók, e-book konvertálás, kép konvertálás. Ezek természetesen, mint kiderült, egyáltalán nem egyenlő nehézségű részek, ez meglátszott a részekhez szükséges kutatási és fejlesztési idő igényeken is.

3.2 A feladat megoldása

Az első nagy kérdés a programmal kapcsolatban az volt, hogy mégis, hogyan lesz grafikus felülete? Erre kérdésre a válasz a QT, hisz C++-t szerettem volna használni, és akár keresést indítottam, akár ilyesmiben már jártas embereket kérdeztem, mind ezt javasolták. Emellett az sem elhanyagolható tény, hogy bár számomra ez egy teljesen új dolog volt, amivel meg kellett ismerkedjek, meg kellett tanuljam (legalábbis egy kis szeletét), az igen széleskörű elterjedésének köszönhetően rengeteg segédanyag, minta, valamint már feltett és megoldott hiba- és válaszgyűjtemény várt rám.

Ezzel egyidőben szintén komoly kérdés volt, hogy hogyan is érhetem el a számítógépen tárolt fájlokat, azokat hogyan rendezhetem valamilyen struktúrába és egyáltalán, hogyan kerülhetek velük interakcióba. Erre a választ magában a QT-ben találtam meg, a QFileSystemModel osztályt használtam erre a célra, ezt később bővebben kifejtem.

E kettő után már csak két jelentős kérdés marad, az egyik az ebookok konvertálása, a másik a képeké. Elsőre a könyvekhez egy Aspose.Words nevű C++ könyvtárcsomagot akartam használni, azonban ezt több megakadást követően leváltottam egy elterjedt, nyílt forráskódú e-book manager-re, melynek használata parancssoros módon is lehetséges, így könnyebb elérni a saját programomból. Természetesen ezeket is kifejtem bővebben.

A képekhez végül szintén a QT volt segítségemre, a QImage osztállyal, így végül is a saját programomon kívüli, külső megoldásra egyedül az e-book konverzió esetén kellett támaszkodnom. A következő, a megoldás sorrendjében szereplő alfejezeteknél természetesen erről is írok bővebben.

3.2.1 A fejlesztéshez használt eszközök

A fejlesztés során Windows 11 64 bites rendszeren futtatott QT Creator 8.0.1 (Community)-t használtam, QT 6.4.0 framework verzióval, valamint Desktop Qt 6.4.0 MinGW 64-bit kit-tel a fordításhoz. Ezek mellett használtam még a Calibre 6.9.0 64-bit verzióját, ez felel az e-book konvertálásért. A Calibre ezen verziójának telepítője a Dependencies mappában található, az aktuális verzió a Calibre honlapjáról ingyenes letölthető (a szakdolgozat írásának idején a Calibre teljes backward compatibility-t (visszafelé kompatibilitást a korábbi verziókkal), teljes ingyenességet és nyílt forráskódot biztosít).
Linux Ubuntu-n történő fordításhoz és futtatáshoz egy VirtualBox virtuális gépre telepített Xubuntu 22.04 LTS verziót (ez egy Ubuntu változat, XFCE asztali felülettel) használtam, QT Creator 10.0.0, QT 6.4.0 keretrendszerrel és Desktop QT GCC 64bit kit-el. Itt a Calibre-t a hivatalos holnapon található leírás alapján telepítettem, ennek verziója a 6.17 volt.

3.2.2 A QT keretrendszer

A QT egy a fejlesztők számára szánt keretrendszer, melynek célja, hogy egyrészt a grafikus felületek létrehozását könnyítse, másrészt nagy hangsúlyt fektet a cross-platform lehetőségek megvalósítására, azaz, hogy olyan alkalmazásokat lehessen készíteni, melyek függetlenek az operációs rendszertől, így ugyanaz az alkalmazás legyen futtatható mind windows-os, linux-os, vagy éppen mac-es környezetben. [1]
Üzleti felhasználása fizetős, azonban van egy nyílt licenc-es mód is, ahol nem üzleti célokra felhasználva ingyenesen hozzáférhető.

A QT több programozás nyelvvel is használható, ezek közül a legnagyobbak a C++, valamint a Python.
Rendelkezik saját IDE-vel (Integrated Developmnet Enviroment), amely QT Creator néven fut. Telepítéskor választható, hogy szeretnénk-e ezt is telepíteni, mivel a keretrendszer használatához nem kötelező. Jómagam telepítettem és használtam, hiszen

sok, a fejlesztés megkönnyítő funkció található benne, de több fórumon és hasonló oldalon olvastam olyanoktól, akik más IDE-t, például Visual Studio-t használtak.

A QT rengeteg saját típust és osztályt definiál, ezek a program fejlesztése során szabadon használhatóak. A már említett QT Creator ezeknek a használatában is segítséget tud nyújtani. Kódolásban kevésbé jártas személyeknek is célszerű lehet a használata, mivel van egy grafikus, drag &drop módszerrel működő UI (User Interface) felülete, ahol például, ha egy gombot szeretnénk a programunkhoz hozzáadni, elég azt az eszköztárból egér segítségével a kívánt helyre húzni. Itt a tulajdonságai, valamint működése is szerkeszthető, de a konkrét működést, például, hogy mit csináljon, ha ráklikkelünk, mindenképpen a választott programozási nyelven kell megírnunk. Ennek a felületalkotási lehetőségnek nagy előnye a gyorsaság, viszont hátránya a dinamikusság, mivel az így létrehozott felület kevésbé változtatható a dinamikusan, a program futása során. Emiatt ezt a módszert célszerű akkor használni, ha a tervezett program alapvetően statikus, azaz a program nézethez tartozó elemek, mint gomb, panel, menü, stb, nem változnak folyamatosan a program futása során. Amennyiben ennek ellenkezője igaz, célszerű lehet meggondolni, hogy magukat a kezelő és megjelenítő elemeket is kódolva adjuk hozzá a programhoz, így biztosítva a dinamikusságát.

A feladatom megoldása során több QT-beli osztályra is támaszkodtam, ezek lentebb kerülnek kifejtésre, ahol a konkrét program funkciók működését veszem sorra.

3.2.3 Calibre e-book manager

Természetesen nem kerülhető ki a Calibre, hiszen a program egyik lényegi funkciója támaszkodik rá. [2] A Calibre egy nyílt forráskódú (azaz a program kódja szabadon hozzáférhető, szerkeszthető, felhasználható), ingyenes e-book kezelő program. Bár hivatalos információ nincs róla, könnyen feltételezhető és belátható, hogy ezen program ez e-book kezelők között a legnagyobbak közt szerepel, ezt nagy mértékben elősegíti a fentebb említett ingyenesség mellett a cross-platform tulajdonsága (azaz, hogy különböző operációs rendszereken ugyanúgy elérhető és használható), a számtalan e-book formátum támogatása, valamint természetesen a nagy mértékű személyre szabás lehetősége és a bővítmények kimerítetlen tárháza. A program rendelkezik grafikus felhasználói felülettel és parancssoros móddal is, amely fontos tényezőként játszott szerepet a programom megírásakor, hiszen külső programként kívántam futtatni a Calibre-t a konvertáláshoz,

amit, mint kiderült sokkal egyszerűbb és mivel nem kell grafikus felületet megnyitni, így gyorsabb parancssori parancsként megtenni. Beépített olvasó funkcióval is rendelkezik, így közvetlenül a számítógépen olvasható bármilyen meglévő e-book.

Az eddigi, e-book-ok világában eltöltött időm, valamint ez idő alatt szerzett tapasztalataim alapján kijelenthető, hogy egy-két kifejezetten speciális művelettől eltekintve, bárminemű műveletet, feladatot szeretnénk végezni e-book(ok)-kal, azt ezzel az alkalmazással meg lehet tenni, el lehet végezni.

Érdekességképp: ilyen nem megoldható feladat például az amazon kindle paperwhite e-book olvasón a Collection-ök (gyűjtemények) kezelése, lokális fájlokkal. Ez betudható annak, hogy ez az e-book olvasó alapvetően úgy lett megalkotva, hogy az amazon-on keresztül vásárolt könyveket olvasson rajta az ember, így a szoftvere, valamint a formátumok támogatottsága is zárt, véges. Emiatt a zártság miatt érthető, hogy nem lehet csak úgy belenyúlni külső programmal.

A Calibre letöltésre elérhető a hivatalos weboldalon, ahol természetesen rengeteg leírást is találni a különféle használati módokról, valamint magának a programnak a működéséről.

A telepítése egyszerű, lényegében csak az ÁSZF-et (általános szerződési feltételek) kell elfogadni, majd a telepítés gombra nyomni. A telepítő felrakja a grafikus és parancssoros módokat is, így azok külön telepítés, konfigurálás stb. nélkül azonnal használhatóak. Bár ezen lépések Windows esetén érvényesek, Linux esetén sem sokkal hosszabb a művelet. Bővítmények utólag telepíthetőek, ehhez internetelérés szükséges.

A szakdolgozatom számára fontos, ebook-convert parancssori utasítást a programom ehhez funkciójának leírásához tartozó részben bővebben kifejtem.

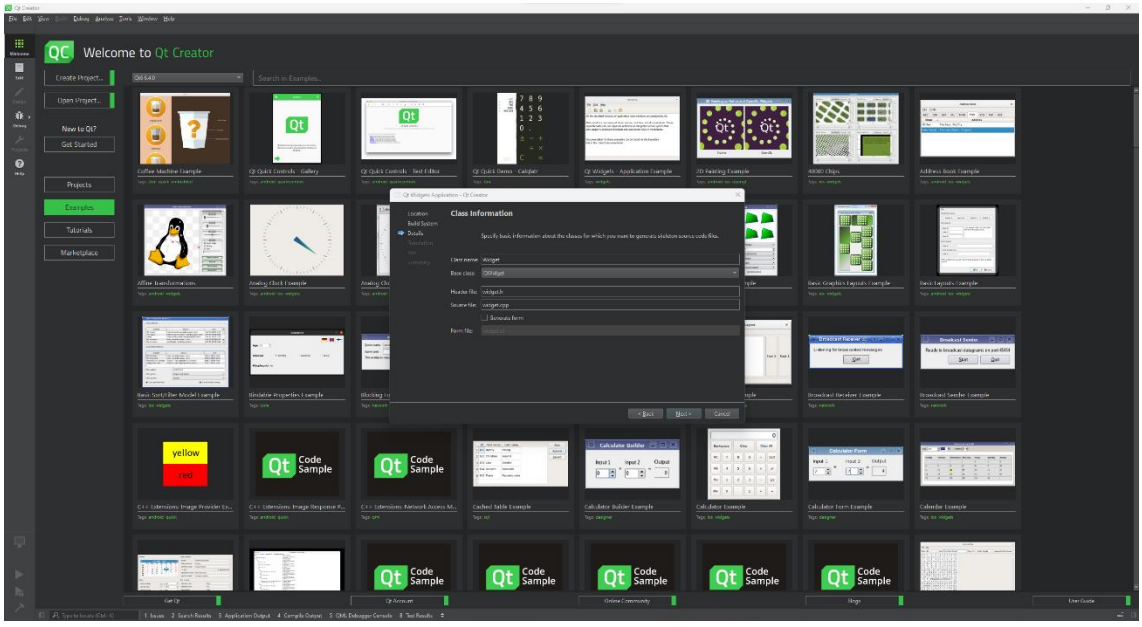
3.2.4 QT Project – FileManager

Elsőként a program létrehozássakor QT Creatorban kezdtem egy project-et FileManager néven, így ez lett a programom neve is.

Új project létrehozásakor a QT Creator több template (minta/sablon) lehetőséget is felkínál, itt értelemszerűen az Application (QT) és QT Widgets Application párost választottam, hiszen egyrészt C++-t használok, így a QT for Python nekem nem lenne jó, illetve grafikus felületet akartam, így a többi lehetőség, miszerint parancssori applikáció, vagy mobil applikáció nekem kiesnek a lehetőségekből.

Ezután lehet még több dolgot beállítani a project (és a program) számára. A build system esetemben a qmake lett, ez QT telepítéssel mindig elérhető, így nem kell külön keresni, hogy egy másik, adott build system (ez felel a program build-eléséért, azaz, hogy a forráskódból futtatható program legyen) elérhető-e épp a rendszeren, vagy elég friss verzió-e.

Az egyik, talán legfontosabb beállítás a harmadik lépésnél jön elő, ahol létrehozhatunk egy alap osztályt, amit számaztathatunk is, ahogy én tettem a QWidget-ből. A lista alsó felén opcionális lehetőség, hogy ui fájlt is hozzon-e létre, én ezt nem jelöltem be. Ha így tettem volna, akkor kaptam volna azt a ui fájlt a programhoz, ahol grafikus felületen lehet szerkeszteni a program kinézetét.
Ezek után még kérdéses a KIT, amit választani kell, ez nálam a QT-vel telepített, fentebb is írt Desktop Qt 6.4.0 MinGW 64-bit, Linuxos fordítás és futtatás esetén pedig a Desktop Qt 6.4 GCC 64bit.



9. ábra, QT Creator project létrehozás

Ekkor a QT Creator, ami a 9. ábrán látható, a paraméterezés után legenerálja a fájlokat project-be szervezve, ahol végül kaptam egy Hello World jellegű programot, amit fordítva és futtatva (build-elve) egy üres programablakot kaptam, ami bár nem egy nagy látványosság, jelzi, hogy minden beállítás, telepítés stb. jó, tud fordulni és futni is a program. Számomra ez amolyan nulladik teszt, ami inkább a fejlesztői környezetet ellenőrzi.

3.2.5 UI szerkesztés, vagy kódolt grafika?

Ahogy látható a QT Project - FileManager fejezetben, mindössze néhány kattintással, a project létrehozásával egyidőben elkészültek a forrásfájlok (amikhez persze igény esetén továbbiak is létrehozhatóak, ha strukturáltság, modularitás igényli), valamint az üres főablak. Amennyiben nem ott hozom létre, utólag kellett volna a szükséges kódsorokat hozzáadni, amelyek ennyire váz/sablon esetében lényegében csak másolás kézzel, ami időigényes lehet. Emellett így könnyen elkerülhetőek vagy épp felderíthetőek alapvető hibák, amik ennél a lépésnél jellemzően még a fejlesztési környezet beállításainak hibája lehet, ami könnyebben kibukik, ha maga a meglévő kód a QT fejlesztői által beállított alapkódok, így azok több ellenőrzésen átmentek, mint amit épp csak odaírok kézzel, így kizárva, hogy a kódból ered a hiba. Ilyen érdekes hiba volt például, amikor a project-et tartalmazó mappa nevében SPACE, vagy ékezetes karakterek voltak, ekkor ugyanis nemhogy nem fordult a kód, de még az alap include-okra is hibát jelezett az IDE, mondván, hogy nem találja a fájlokat. Egy ehhez hasonló, fájlnevekkel kapcsolatos hiba később még sok fejtörést okozott a fájlkezelési műveleteknél, de erről ott részletesebben írok.

Kérdés volt számomra, hogy végül a programomhoz szükséges további felületi elemeket kódként írva, vagy a UI szerkesztővel rakjam fel a felületre. Korábban már pedzegettem, hogy itt leginkább a dinamikusság (kód módszer) és a gyorsaság, esetleg kisebb hibalehetőség (UI szerkesztő) állnak egymással szemben. Átgondolva, hogy mit is szeretnék elérni a programommal, hogy hogyan működjön, hogyan nézzen ki, elsőre a dinamikusság rovására a UI módszert választottam, majd a fejlesztés közben felmerült kisebb-nagyobb problémák után a programot átdolgoztam úgy, hogy a megjelenítést is teljesen kódként írtam meg. Bár ez a változás alapvetően használat közben nem érzéklehető, a fejlesztés folyamán többször bebizonyosodott, hogy ez volt a jó döntés, hiszen sokkal több megoldást lehetett találni kódként egy-egy hibára, mint UI tulajdonság

állításként.

A program kapott egy minimum méretet 1280x720 pixelben meghatározva, ennél kisebbre nem lehet állítani az ablakot, nagyobbra azonban igen. Ebből is ered ezen felbontás követelményként való megjelenése is. Ehhez az értékhez az ihlet részint saját használati szokásból ered, hiszen nagyjából ekkorában nyílik meg a Windows Fájlkezelő is, amit általában pont jó méretűnek látok, valamint megkérdezve néhány, számítógépet kifejezetten felhasználóként használó ismerőst, ezen kis tézis megerősítést nyert.

Ezek mellett fontos információ még, hogy az alkalmazás view-model struktúrában valósult meg, azaz alapvetően külön osztály felel a megjelenítésért, és külön osztály a műveletek végrehajtásáért. Ez alól kivétel az e-book konvertáláskor megjelenő animáció. Emellett több kisebb saját származtatott osztály is készült, különböző részfeladatok megoldására.

3.2.6 Az osztályok gyors felsorolása

Az alábbiakban röviden bemutatom a programot alkotó osztályokat, feladatukat.

A *View* osztály mondhatni az alapja a programnak, ez felel a teljes felület megjelenítésért, a QWidget-ből származik. Létrehoz egy saját példányt a *Model* osztályból, így azzal signal-ok segítségével kommunikálva hajt végre utasításokat eseményekre, például kattintásra, gombnyomásra reagálva. Legtöbbször ez azt jelenti, hogy az esemény hatására signal-t küld, amire a Model megfelelő metódusa végrehajtódik, majd ha van megjeleníteni való, például egy sikeres konvertálásról a felhasználó tájékoztatása, akkor azt megjeleníti.

A *Model* osztály, ahogyan említve volt, a működésért, a logikért felelős a QObjectból származik. Szintén küld signal-okat, a *View* felé, hogy bizonyos eredmények, legyen az hiba, vagy egy sikeres konvertálás, az megjelenjen a felhasználó felé.

Az *EbookConvThread* osztály a nevéből is láthatóan e-book konvertáláshoz van használva, a QThread-ből származik. Feladata, hogy külön szálon futva meghívja az „ebook-convert” parancsot a megfelelő paraméterekkel. Mivel a konvertálás befejezésére

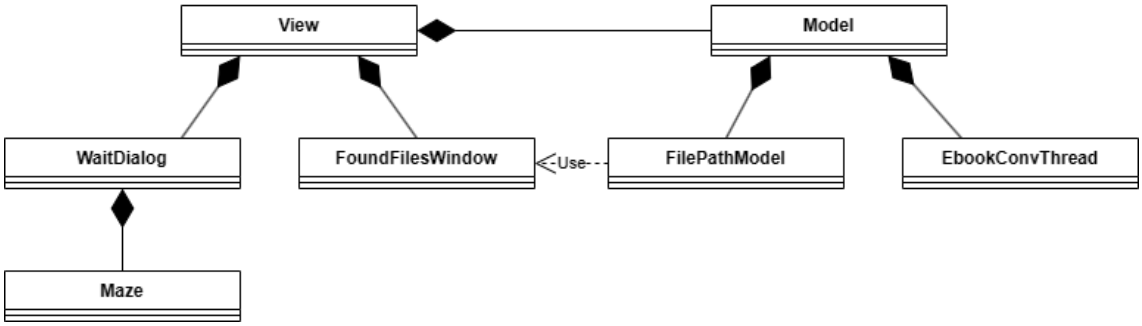
várni kell, így célszerű volt külön szálra helyezni, hisz a program fő szála így folyamatosan futhat tovább.

A *FilePathModel* osztály, amely a QAbstractListModel osztályból származik, a fájlkeresénél kerül felhasználásra, ahol egy egyedi modellre volt szükségem, amiben a talált fájlok elérést tárolom.

A *FoundFilesWindow*, amely a QDialog-ból származik, az előbbi *FilePathModel* tartalmát jeleníti meg. Akkor jelenítem meg, ha 2 vagy több fájlt talált a keresés, ekkor a megjelenő listából választva, majd OK gombot nyomva a választott fájl a főablak jobb oldali panelén jelenik meg, ahonnan a funkciók elérhetőek.

A *Maze* osztály, a QWidget-ből származtatom, feladata labirintus generálása és kirajzolása. Ezt egy QTimer-nek köszönhetően másodpercenként végzi, így amolyan várakozó animációként működik, amíg az 5-10, esetenként ennél több másodpercig tartó e-book konvertálás befejeződik. Itt jó, hogy van az *EbookConvThread*, hiszen ha a főprogramban várnák a befejezést, a főprogram futása addig teljesen leállna.

A QDialog-ból származó *WaitDialog* feladata a *Maze* animáció megjelenítése, ami olyan szempontból előnyös, hogy a QDialog-nál van lehetőség azt beállítani, hogy amíg aktív, látszik, addig másik ablakra ne lehessen kattintani, vagyis muszáj legyen rendesen végig várni a konvertálást.



10. ábra, Osztályok kapcsolata osztálydiagramon

3.2.7 Fájlok megjelenítése, a két panel, QFileSystemModel

Alap fájlkezelő, valamint fájlokkal dolgozó program lévén mondhatjuk, hogy első lépés, hogy a program érje el a fájlokat, tudja azokat megjeleníteni. Már adott az eszköz (QT), van egy alap, üres program ablak is, és egy feladat, miszerint jelenjenek meg a fájlok.

Ezen ismeretekkel felvértezve már kezdődhetett az információgyűjtés, hogy ez hogyan lehetséges. Bár elsőnek a kereséseket elrontottam, és mindig arra találtam megoldásokat, hogy QT Creator-on belül a forrásfájlokat hogyan lehet megnézni, hamar belefutottam egy tutorial (oktató) videóba, ahol a QFileSystemModel-t használva hozott létre a videó készítője egy alap fájlmegjelenítő programot. [3] Lényegében azt csinálta meg, amit én szerettem volna, így a videót alapul véve hoztam létre a saját programom számára is a két panelt az ablakon, ahol a baloldalin a meghajtók és mappák látszanak, míg a jobboldalin a bal oldalt kijelöl meghajtóhoz, mappához tartozó fájlok.

Maga a QFileSystemModel leírása is azt foglalja magába, hogy az osztály célja, hogy hozzáférést biztosítson a programot futtató számítógép fájlrendszeréhez, ezáltal a fájlokhoz is, valamint biztosít alapvető műveleteket fájlok, mappák törléséhez, átnevezéséhez, vagy épp azokról valamilyen információ megszerzéséhez. Itt könnyen látható, hogy mivel terveztem is fájl átnevezést és törlést is hozzáadni a programomhoz, kézenfekvő ezen osztály használata, több problémámra is megoldást nyújt.
Jelen feladatnál számomra azonban még csak az kellett, hogy a fájlok megjelenjenek.
Elsőnek hozzáadtam a programhoz a két QTreeView panelt, amelynek a nevéből is látszik, hogy fa szerkezetben tárolt modellek megjelenítésére alkalmasak. Ez azért is kényelmes, mert a QFileSystemModel pont egy ilyen, fa szerkezetű modellel tér vissza, ahol az elemek a fájlok, mappák lesznek.

Ezek mellé még egy kis szűrést raktam be, mivel ugye van egy „mappás” oldal, és van egy „fájlos” oldal, így a két panelhez van két modell, ahol az egyiknél megjelenítésre szűrve van, hogy mappák látszódjanak, míg a másiknál értelemszerűen fájlok.

```
void Model::setDirModel(QString sPath)
{
	dirModel = new QFileSystemModel(this);
	dirModel->setFilter(QDir::NoDotAndDotDot | QDir::AllDirs);
	dirModel->setRootPath(sPath);
}

void Model::setFileModel(QString sPath)
{
	fileModel = new QFileSystemModel(this);
	fileModel->setFilter(QDir::NoDotAndDotDot | QDir::Files);
	fileModel->setRootPath(sPath);
}
```

A fenti kód tartalmazza ezen beállításokat a két modell számára. A *Model* osztály rendelkezik két getter metódussal is ezen modellekhez, így amikor a program indulásakor létrejön a *View* osztály, a getterekkel be is állítja a megfelelő modelleket a QTreeView-khoz.

Fontos, hogy a QFileSystemModel számára kötelező egy kiindulási pont, amit root-nak, azaz gyökérnek neveznek. Innen indulva építi fel a modellt, amiből az is látható, hogy ha nem a lehető „legmesszebbről” indítom, akkor bizony kihagyhat mappákat, fájlokat. A modell rekurzívan épül fel, ezért a gyökérként beállított pontról csak „befelé” vizsgál, vagyis, ha például egy A nevű mappát adunk meg neki, akkor a modell számára csak az A mappa tartalma lesz releváns, azokkal fogunk tudni dolgozni.

Jelen esetben „” üres QString van megadva, így a QT keretrendszer automatikusan állítja a gyökeret, Windows esetén ez a meghajtókat jelenti (azaz ahány meghajtó van a gépen, legyen az HDD, SSD, pendrive), minden meghajtó megjelenik a listában, Linux esetén pedig „/”-t jelenti, ami lényegében a rendszerlemez gyökere. A kiemelt kódrészlet a Model metódusai, amelyek a konstruktorban meghívásra kerülnek, azaz amikor a program elindul, és a *View* osztály példányosít egy *Model*-t, rögtön lesz mit megjeleníteni.

Pozitív adalék a megjelenítéshez, hogy a QT az operációs rendszer irányából a mappák és fájlok mellé betölti az azokhoz tartozó alapértelmezett ikonokat is, azaz a mappáknak a rendszer által használt kis mappa ikont, vagy docx fájlokhoz a Word ikonját

(amennyiben a Word telepítve van, és be van állítva az adott fájlkiterjesztéshez, mint alapértelmezett program).

3.2.8 Fájlok jobb oldalon

Miután a két panel kész, még fennmaradt a probléma, hogy bár mindkét oldalon lehet „lenyitni”, nem jelennek meg a fájlok jobboldalon. Az elképzelés ennek működésére az volt, hogy ha kijelölök egy meghajtót, vagy mappát a baloldalon, akkor annak a fájltartalma jelenjen meg jobb oldalon.

```
void Model::updateFileModel(QModelIndex index)
{
    sPath = dirModel->fileInfo(index).absoluteFilePath();
    QModelIndex indexBack = fileModel->setRootPath(sPath);
    emit modelUpdated(indexBack);
}
```

A kódrészletben ennek megoldása látható, ahol, ha a baloldalon megtörténik egy kattintás, akkor lekérdezem annak az elemnek az abszolút útvonalát, egy signal-lal elküldöm a *Model*-nek, majd a jobb oldal számára annak a mappának az útvonala kerül beállításra, mint forrás, majd egy másik signal visszajelez a *View*-nak, ahol megtörténik a nézet módosítása.

Példa: C:/Users/user/Documents/beadando.pdf, ahol a / jelek a mappákat választják el, a végén pedig egy fájl található, de jelen esetben a C:/Users/user/Documents egy teljesen helyes, jól használható útvonal lehet.

Ehhez szükség volt még a QT által biztosított connect()-re, amely azt teszi lehetővé, hogy eseményeket, signalokat kössünk össze slot-okkal, metódusokkal. Aaza, ha egy esemény történik, az signal-t vált ki (vagy mi küldünk egyet az emit signal-lal), amire utána a connect() segítségével kapcsolhatunk valamilyen végrehajtást.
Erre példa:

```
connect(dirView, &QTreeView::clicked, this, &View::dirViewClicked);
```

Itt a dirView nevű QTreeView példányomnál lett kötve a QTreeView által a clicked signal, ami értelmszerűen egy kattintásra váltódik ki, a View osztályom dirViewClicked

metódusa[3], amely egy signal-lal továbbítja a *Model* felé a kattintott elem QFileSystemModel-beli indexét, ami alapján a *Model* végrehajtja a másik QFileSystemModel változtatását, hogy a megfelelő fájlok jelenjenek meg.

Így annyi történik, hogy mivel „szerencsémre" két modell, és két nézet van, így egyszerűen veszem a mappa nézetből a kijelölt mappa, meghajtó abszolút útvonalát (abszolút útvonal, amikor az útvonal mindenképp a számítógép fájlrendszerének gyökeréből indítva ad teljes elérési útvonalat), majd ezt, mint gyökér állítom be a fájlokat megjelenítő modellnek, ami frissül a nézetben, így a kijelölt mappa tartalma látszik a jobboldalon.
A nézetben a frissítés:

```
void View::setNewRoot(QModelIndex index)
{
    fileView->setRootIndex(index);
}
```

3.2.9 Az első funkció, fájlok megnyitása

Miután sikerült eljutnom oda, hogy már van programablakom, és azon megjelennek a fájlok, elértem a következő nagyobb mérföldkőhöz, az alapvető fájlkezelési funkciókhoz. Sokaknak több minden is beugorhat egyszerre, mint például törlés, átnevezés, azonban én egy még, mondhatni „alapvetőbbel" kezdtem, mégpedig a fájlok megnyitásával. Szintén a hétköznapi életből vett ötletem, hogy dupla klikk-re lehessen megnyitni a fájlokat, hiszen ez évek óta amolyan standard-ként (alapértelmezettként) van jelen hasonló alkalmazásokban. Meglepő módon, ennek a kivitelezése sokkal több keresést igényelt, mint amennyire végül „bonyolult" lett, hisz valójában az eredmény két sor kód lett, amik ráadásul nem is olyan összetettek.

Elsőnek persze meg kellett oldani, hogy egyáltalán a program reagáljon, ha a dupla kattintás megtörténik a megfelelő felületen. Itt a már megismert connect()-et használtam újra (és még elég sokszor a program további részén), ahol a megfelelő eseményt, azaz a dupla klikket kötöttem egy metódushoz.

[3] QT4 idején még külön működésű volt a slot és metódus mechanizmus, ezt azonban a QT5-ben már eltörölték, így a kettő egyenértékűnek számít. Jelnleg a legfrissebb a QT6.5.

```
connect(fileView, &QTreeView::doubleClicked, this,
&View::fileViewDoubleClicked);
```

Itt könnyítés volt, hogy a doubleClicked() paraméterként megkap egy QModelIndex-et, ami konkrétan a klikkelt fájl teljes elérési útvonala (ez lényegében a gyökértől vett, mappákon át vezető útvonal, amelyet követve elérhetjük a fájlt).
Mivel lényegében minden bemenő információ megvan, következhet maga a fájl megnyitás. Itt arra gondoltam, hogy a legjobb lenne, ha ugyanúgy működne, mint a Windows fájlkezelőnél, azaz egyszerűen megnyitja az operációs rendszer alapértelmezett programjával.
Az eredmény a következő lett:

```
void Model::fileOpen(const QModelIndex index)
{
    QString oPath = fileModel->fileInfo(index).absoluteFilePath();
    QDesktopServices::openUrl(QUrl::fromLocalFile(oPath));
}
```

Első lépésben kell a fájl abszolút elérési útvonala, majd ezt átadom a QDesktopServices osztály openUrl static metódusának, amely végrehajtja a kapott „címmel” a fájl megnyitását.
Ez az osztály, a QFileSystemModel-hez hasonlóan, közvetlenül az operációs rendszer nyújtotta szolgáltatásokhoz biztosít bizonyos mértékű hozzáférést.
Korábban, említettem egy hibát a nevekkel kapcsolatban, ami itt még visszaköszönt, ugyanis eredetileg rövidebb volt a megnyitásért felelős kódsor:

```
QDesktopServices::openUrl(QUrl(sPath));
```

Azonban ez nem tudott megnyitni olyan fájlokat, amelyek SPACE karakter, vagy ékezetes betűket tartalmaztak, kizárólag az angol ABC betűiből, számokból és egyszerű írásjelekből (pl.: -; _) álló névvel rendelkező fájlokkal működött. Itt hibakeresés gyanánt kiírattam a QString típusú változóból a fájl elérési útját a konzolra, azonban ez nem sokat segített, hiszen helyes volt, később derült ki, hogy a kód, amit elsőre írtam, elég általános változat, és kifejezetten URL link (uniform resource locator, a webböngészés során

használt szabványos cím) típusú címekkel (amelyekben értelemszerűen sem SPACE, sem ékezetes karakterek nincsenek) tud operálni, míg az utólag a sorhoz hozzáadott **fromLocalFile** kezeli a többi karaktert is, így a fájl nevében szereplő speciális karakter már nem akadály a megnyitáshoz. [4] Természetesen ettől függetlenül kell, hogy adott fájlhoz az adott számítógépen legyen telepítve program, ami támogatja, és meg tudja nyitni, különben akárcsak más fájlkezelők, ez sem fogja tudni megnyitni a fájlt.

3.2.10 Funkció gombok közös elemei

Amikor elértem az első, gombbal működtetett funkcióhoz, kellett egy megoldás a fájl elérési útjának megszerzésére, hiszen ezentúl a gombokhoz lesznek társítva metódusok, így nem kapom meg a korábban paraméterként látott QModelIndex-et, ami a kijelölt fájl, vagy mappa elérést tartalmazta, erre pedig mindegyik további funkció esetén szükségem lesz, hiszen a fájlokkal szeretnék valamilyen műveletet végrehajtani.

Szerencsére, ez nem volt olyan nehéz, mint elsőre gondoltam. A QTreeView-nak, van egy metódusa, a currentIndex(), ami pont egy QModelIndex-szel tér vissza, ha van kijelölt elem, vagyis nekem csak egy ilyen típusú változó kell, amibe ezt elmenthetem, a *Model* számára egy signal-al elküldhetem.

Ezután egy ellenőrzést is végre lehet hajtani, az index-nek, ami most egy QModelIndex, lehet validitását nézni, ami igaz, ha van egy kijelölt fájl, és hamis egyébként, amikor is egy a felhasználó felé küldött hibaüzenetet követően ki is lépek a metódusból, így nem kerül lefutásra olyan rész, ami támaszkodna egy adatra, ami nem helyes.

Ezen ellenőrzés minden funkciónál használható lett számomra. Ez alól amolyan kivételként jelenik meg a fájlkeresés, viszont ott csak annyi a módosítás, hogy egyrészt lehet kijelölt mappa is, de az sem kötelező, bármilyen kijelölés nélkül is el lehet indítani. Erről részletesebben a keresésről szóló fejezetben.
Az index ellenőrzés:

```
void Model::indexCheck(QModelIndex index)
{
    if (!index.isValid()){
        emit errorOccurred("No file choosen.");
        return;
    }
}
```

A paraméterként kapott (signal-al érkezett) index-et ellenőrzi,, hogy az index megfelel-e művelet végrehajtáshoz. Ez az üzenet látható a felhasználó dokumentáció fejezetben, az 5. ábrán.

Közös elemként említeném még, hogy a műveletek végrehajtása során általában van még egy-két pont, ahol hiba történhet, így minden gombhoz megírt metódus tartalmaz további ellenőrző lépéseket, ahol hiba esetén hibaüzenetet küldve a felhasználónak a program kilép az adott metódus futásából. Ezekben az üzenetekhez egy közös metódus tartozik, amelynek paramétere maga az üzenet. Ez a metódus egy emit errorOccurred(„ÜZENET”) signal-al váltható ki, vagyis mindenhol, ahol ezt használom, egyedi üzenetet adhatok meg, ezen kívül azonban teljesen ugyanúgy fog megjelenni minden hiba esetén. Működéskor egy kis üzenet ablakban jeleníti meg a szöveget a program ablak felett, átvéve a fókuszt, így amíg az üzenettel nem foglalkozunk, a főprogramablakban nem tudunk semmit csinálni.

Bizonyos gombok metódusai között vannak további hasonlóságok vagy egyező műveletek, azonban azok már csak kettő, esetleg három gombot érintenek, így ezekről az érintett gombok fejezeteinél írok részletesebben.

Továbbá, bár mindenhol a *View* és a *Model* közt signal-okkal zajlik a kommunikáció, valamint ezeket és a végrehajtandó metódusokat connect()-ek kötik össze, ezek ismétlődő leírásától a továbbiakban eltekintek, hiszen semmi új nem lesz bennük.

3.2.11 Fájl törlése

Első, gombbal működtettet funkcióként a fájlok törlését valósítottam meg. Bár az előző fejezetben leírtam közös funkciókat, a fejlesztés során azonban természetesen nem így került kidolgozásra, hanem valamelyik gomb metódusánál elkészült egy megoldás, majd

azután ültettem át a többibe, ahogy azok készültek. Itt a törlésnél lett meg a QModelIndex probléma megoldása, ahol ugye a gond az volt, hogy bár most már nem a treeView paneleken történnek az események, mégis szükség van a kijelölt (ha van ilyen) fájl elérésére.

Mivel ez most már rendelkezésre áll, maga a fájl törlése a QFileSystemModel::remove(QModelIndex) metódusával könnyen elvégezhető. Természetesen előtte ellenőrizni kell, hogy az index helyes-e. A törlés metódusa visszatér egy true (igaz) értékkel, ha a törlés sikerült, így egy ellenőrzést itt is el tudtam végezni, valamint megírni, hogyha nem sikerült, akkor erről a felhasználó kapjon egy üzenetet.

3.2.12 Fájl átnevezése

Itt a QInputDialog osztályt használtam, abból hoztam létre egy példányt, az új fájlnév beviteléhez. A példány könnyen paraméterezhető, és ahogy a neve is mutatja, kifejezetten a felhasználó felől történő adatbevitelre való. A fentebb tárgyalt kezdeti index megszerzése és ellenőrzése után itt már el kellett mentsem a fájl teljes elérését, ugyanis az átnevezést a QFile osztály segítségével végzem.

Az abszolút útvonal segítségével létrehozok egy QFile objektumot, jelen esetben a fájlnév megadásával. Ezután jön pár sor string művelet, ahonnan külön kiszedem csak a mappáig az útvonalat, majd a fájl nevét is szétszedem a névre és a kiterjesztésre. [5] Erre azért van szükség, mert elegendő csak a kívánt új nevet beírni, ezután ezen QString változókkal a fájl teljes elérése újra összerakható: csak a mappáig útvonal + az új fájlnév + az eddigi kiterjesztés. Itt egy érdekes hiba volt, hogy eleinte az átnevezett fájl mindig egy mappával kijjebb került, a nevéhez pedig hozzáadódott az eredeti mappájának a neve. Lázas kutatás után kiderült, hogy a section() QString metódus, amit a bontáshoz használtam, a paraméterként kapott elválasztó elemet (pl.: / jel), a keresett helyen, ami nekem ugye az utolsó előfordulása a stringben, hisz az az eredeti mappa útvonala, levágja. Miután erre rájöttem a végső képlet ez lett: csak a mappáig útvonal + / jel + az új fájlnév + az eddigi kiterjesztés. Ez látható az alábbi kódrészletben is:

```
QFile file(sPath);
if(file.exists()){
    // get only the path for the file, to create absolute path with
this and new name
    QString fPath = sPath.section("/",0,-2);
    // full file name
    QString sFile = sPath.section("/",-1,-1);
    //file extension only
    QString sFileFormat = sFile.section(".",-1,-1);
    // important: the section method cuts the last "/", which then
leads the file to one folder up after rename
    QString newFileName = fPath + "/" + newName + "." +
sFileFormat;

    if(!file.rename(newFileName)){
        emit errorOccurred("Failed to rename file.");
    }
```

Elsőnek a QFile létrehozása, majd ennek ellenőrzése, utána a fájl elérési útvonalának szétszedése, majd az új elérési útvonal létrehozása. A newFileName nevű QString-be kerül, ami azért lett így elnevezve, mert már így is volt a metóduson belül két változó sPath és fPath, amik ugye útvonalat tartalmaznak, így követhetőbbnek tartottam, hogy a fájl új „neve” -ként hivatkozzak a teljes új elérési útvonalra.

Természetesen üres nevet nem lehet fájlnak adni, ezt is le ellenőrzőm, ha ez rendben van, akkor a QFile::rename() metódusával elvégzem az átnevezést. Amennyiben ez valamiért meghiúsul, erről is üzenetet kap a felhasználó.

3.2.13 Fájlkeresés

Utolsó funkció a fájlkezelési mérföldkőben, a fájlkeresés név alapján.
Korábban volt (és később is lesz), hogy a metódusok végrehajtásához kellett a kijelölt fájl, amivel a műveletet végzem. Itt jelentős különbség, hogy nincs konkrét fájl, hiszen a keresés azt feltételezi, hogy nem tudom, hol van, feltéve, hogy egyáltalán létezik a fájl.

Itt előjött az a kérdés is, hogy a fájlok, mappák között hogyan végezzek keresést. Erre a megoldást a QDirIterator osztály adta, amely a korábbi elnevezésekhez hűen, nevéből egyértelműen jelzi, hogy mappákon és elemieken iterál (veszi sorba azokat). [6] Így a probléma megoldható akképp, hogy indítok egy iterálást, majd minden elemnél megvizsgálom, hogy az elem neve egyezik-e a keresett fájl nevével. [7] Az iterátor

rekurzívan halad, vagyis a kiindulási mappából befelé, azonban kijjebb nem. Így fel is merült a kérdés, hogy honnan lenne logikus indítani a keresést. Itt arra a megoldásra jutottam, hogy ha van kijelölt fájl, akkor a keresés annak a fájlnak a mappájából induljon, ha fájl nincs kijelölve, de mappa/meghajtó igen, akkor onnan induljon, ha pedig semmi nincs kijelölve, ami fájlkeresés esetén helyes induló állapot, akkor alapértelmezettként a rendszer gyökérből indul a keresés.

Másik felmerülő kérdés volt számomra, hogy a fájlt csak név alapján, vagy kiterjesztéssel keressem-e. Itt némi próbálkozás és átdolgozás után végül egy regex-es módszer lett kidolgozva, ahol nem számít a kiterjesztés, de még a teljes név sem kell. A keresés során a felhasználó által megadott szöveget keresi a program az iteráció során minden fájl nevében úgy, hogy nem számít a kis- vagy nagybetű, illetve elég, ha a névben részben szerepel a megadott szöveg. Például, ha a megadott szöveg az „alma”, akkor az „alma.pdf”, de az almafa.jpg” is helyes, mindkettő érvényes találat, de az „alm.txt” már nem. Ehhez a QRegularExpression osztályt használtam, amiből egy példányt létrehozva, elég annak megadni a bemeneti szöveget, valamit regex szabályokat, például: QRegularExpression::CaseInsensitiveOption, amivel ugye a kis- és nagybetű közti különbség nem fog számítani.

Itt előjött még egy elég nagy probléma számomra, hogy ha például több mappa mélységű hierarchiában sikerül több megfelelő fájlt is találni, az hogyan jelenjen meg.
Sok kutatás után végül a megoldás a következő lett: készítettem egy saját, származtatott osztályt, a *filePathModel*-t. Ez egy, a már használatban lévő QFileSystemModel-ekhez hasonló, de kissé azoktól eltérő osztály lett, ami a QAbstractItemModel-ből származik. Ez lehetővé tette, hogy az össze talált fájlt, abszolút eléréssel összegyűjtsem. A keresés során elsőként mindet egy QList-be gyűjtöm, ami QString-eket tárol, ennek könnyű az append() metódusát használni, majd ezt megadva a *filePathModel* konstruktorának egy, a már ismert fileModel és dirModel-ekhez hasonló modellt kaptam, ami a fa szerkezettel ellentétben sima listaként működik, azaz nem kell QTreeView hozzá, elég egy QListView a megjelenítéshez.
Ehhez csináltam még egy származtatott osztályt, a *foundFilesWindow*-t, ami a QDialog-ból származik. Ezen megjelenik a lista a talált fájlokkal, ahol tetszőlegesen egyet ki lehet jelölni, majd, ha az OK gombra kattintunk, akkor azon lista elem alapján kerül beállításra

a főprogram ablakban a jobboldali panel, mintha csak baloldalt a megfelelő mappára kattintottunk volna.

Ha csak egy találat van, akkor a fenti lista és ablak nem fog megjelenni, hisz feltételezhető, hogy úgyis az az egy fájl kerülne kiválasztásra, így automatikusan ez kerül beállításra a nézetben is, egy értesítő üzenet társaságában, amely tájékoztat, hogy egy fájlt talált a keresés.

Amennyiben a keresés nem vezetett eredményre, látható, hogy két eset miatt történhet: az egyik, hogy a fájl ténylegesen nem létezik, a másik az, hogy a keresés nem jó helyről indult. Ilyen példát írtam a felhasználói dokumentáció fejezet 2.3.3-as alfejezetében, ahol A és B mappa egymás mellett szerepelnek. Ekkor, ha egy B mappa béli fájlt keresünk, de a keresés A mappából indul, akkor a keresés rekurzivitása miatt a fájlt nem fogjuk megtalálni.

Ezért a beállított értesítő, hibaüzenetben is szerepel, hogy bár a fájlt nem sikerült megtalálni, a felhasználó próbálja meg a keresést egy „külsőbb” mappából, meghajtóról elindítani.

```
QDirIterator it(sDir, QDir::Files | QDir::NoDotAndDotDot,
QDirIterator::Subdirectories);
QList<QString> found;
QRegularExpression re(fileName,
QRegularExpression::CaseInsensitiveOption);
while (it.hasNext()) {
    it.next();
    if (re.match(it.fileName()).hasMatch()) {
        found.append(it.filePath());
    }
}

if(found.isEmpty()){
    emit errorOccurred("Could not find the requested file.\nPlease
check file name, or start search for an outer directory.");
}
else if(found.size()==1){

    emit foundFile(fileModel-
>setRootPath(found.first().section("/",0,-2)));
    emit jobDone("One file has been found.");
}
else{
    //create a list or qmodel to make a list on view layer
    FilePathModel* foundFilesModel = new FilePathModel(found);

    emit foundFiles(foundFilesModel);
}
```

A kódon a keresés menete látható, a fentebb leírt módon, azaz a megadott kifejezés keresése a fájlok nevében. Látható egy elágazás, amely attól függ, hogy 0, 1 vagy több fájlt sikerült találni.

3.2.14 E-book konvertálás

A korábban már emlegetett mérföldkövek közül elértem az utolsóhoz, ami a konvertálásokat foglalja magába.
Itt már elértem, hogy van grafikus felületem, van módszerem a fájlok elérésére, valamint alapvető ellenőrzési metódusaim is. Így a fő megoldandó probléma maga a konverzió elvégzése.

Természetesen ezen probléma megoldása is sok keresgéléssel indult, ahol az első számomra is jónak tűnő megoldásnak az Aspose.Words for C++ [8] nevű osztály

könyvtár volt, amely leírás alapján dokumentumokkal való műveletekre, köztük konvertálásra készült. Alapvetően a használatát egyszerűnek írják, ahol a megfelelő include használata után pár sor kóddal végrehajtható egy konvertálás. Sajnos ez végül mégsem működött, sőt, az is kiderült, hogy a rengeteg támogatott formátumból valójában csak kettő használható kimenetként, így nem is igazán felelt meg a céljaimnak.

Mindenképpen olyan megoldást akartam, ami internet eléréstől, valamint távoli kiszolgálótól független, azaz mindenképpen lokálisan lehessen elvégezni a konvertálást. Ehhez azt a tanácsot kaptam, hogy keressek olyan programot, ami lehetőleg parancssoros módon használható, mert ha az telepítve van, akkor az én programomból meg lehet hívni azt a programot a konvertálás elvégzésére. Emellé én egy olyan kikötést tettem még, hogy a külső program viszonylag könnyen telepíthető legyen egy átlagos felhasználó számára is, ne legyen felé túlzott elvárás egy nagyon összetett, bonyolult telepítéssel.

Ennek eredményeképp esett a választás a Calibre e-book manager programra. Ahogy a 3.2.3-as alfejezetben is írtam, ez egy ingyenes hozzáférhető, nyílt forráskódú, sok leírással és segítséggel rendelkező, könnyen használható program, amit kifejezetten e-book-ok kezelésére hoztak létre. A program rendelkezik GUI-val (graphical user interface, azaz grafikus felhasználói felület), de ami most számomra lényeges, hogy parancssorból is használható a megfelelő utasításokkal, valamint a telepítése is egyszerű (a Dependencies mappában található egy telepítő[4] is a teljesen offline módon történő telepítéshez).

Szintén, ahogy fentebb említettem, a telepítés sikeressége könnyen ellenőrizhető, ha a parancssorban az **ebook-convert** parancsot kiadjuk. Amennyiben sikeres, úgy a 2.2 alfejezetben található, 2. ábrán látható eredményt kell kapjuk.
Ez lesz az a parancs, amit a saját programom is ki fog adni, természetesen ekkor már paraméterezve lesz, hiszen kell egy bemeneti e-book fájl, amit konvertál és kell egy kimenet fájlnév és kiterjesztés is, amiből az ebook-convert tudni fogja, hogy milyen formátumra konvertáljon.

[4] A telepítő Windows 10 vagy 11 operációs rendszert használó számítógépen működik, Linux esetén a hivatalos oldalon található telepítési lépések követése javasolt: calibre - Download for Linux (calibre-ebook.com)

Miután így elkészültem a tervvel, egyszerűen vettem a korábbi funkcióktól a fájl ellenőrzésére vonatkozó részeket, valamint a fájlnév tagolást is.
Ezek után történik egy formátum bekérés, ahol a következő formátumok közül lehet választani: epub, mobi, azw3, pdf. Mind a négy formátum elfogadott bemenetként, valamint kimenetként is, azzal az egy feltétellel, hogy a bemeneti formátum és a kívánt kimeneti formátum eltér. Ennek az ellenőrzését egy olyan QStringList-el végzem, amelynek elemei a fenti formátumok szöveges felsorolása, és amelyet a kiválaztott opcióval hasonlítok össze, így megvizsgálva, hogy alapvetően van-e értelme a konvertálásnak. [9]

Mivel az ebook-convert helyben dolgozik, azaz manuális esetben, ahol a parancssor van megnyitva, így az e-book-ok konvertálásakor elegendő a felhasználó által a kívánt formátum kiválasztása, utána már a programon belül kinyerhető egyrészt a forrás mappa, másrészt a fájlnév is, illetve megkonstruálható az új fájlnév. Így, a forrás mappát, a régi nevet és az új nevet megadva arhumentumként az „ebook-convert” a forrás fájl mellé készíti el a konvertált példányt az új formátummal.

Az ebook-convert futtatásához a QT QProcess osztályát használtam. Szintén beszédes név, rövid leírásában ez szerepel: a QProcess osztály arra használható, hogy külső programot futtassunk, és azzal kommunikáljunk. Vagyis pontosan azt tudja, amire szükségem volt. Itt fontos megjegyezni, hogy előtte találtam QT független, C++ kódokat is, azonban azokhoz nagyon sok helyen jelezték, hogy nem igazán biztonságos, valamint nem is teljesen stabil a működésük. Alapvetően mindegyik a system() hívást végezte, ahol string paraméterként lehet megadni a futtatandó programot, és annak argumentumait. Ezt kipróbáltam, de nekem eleinte egyáltalán nem is futott le, majd némi javítás után hol igen, hol nem, így ezt hagytam, és ezután találtam rá a QProcess-re. [10]

```
void ebookConvThread::run()
{
    QProcess process;
    process.setWorkingDirectory(dir);
    process.start(prog, args);
    process.waitForFinished(-1);
    emit processFinished();
}
```

A fenti kódrészleten látható, hogy hogyan is történik a külső program meghívása. Elsőnek készítek egy példányt a QProcess-ből, process néven. Ezután beállítom neki azt a mappát, ahol futnia kell, amit ugye már korábban a fájl ellenőrzésekor megszereztem. Ezután a start() metódusnak megadva a program nevét és az argumentumokat a külső program indul is. Az argumentumok, azaz a régi név és új név, valamint a program neve QStringList-ként és QString-ként megadhatóak. A waitForFinished(-1) pedig értelemszerűen megvárja, amíg a külső program futása véget ér. [11]

Látható a kódrészleten még egy csavar: az egész külső program hívás, egy külön szálon történik. Erre a következő miatt volt szükség: a Qprocess::waitForFinished(-1) valóban szépen megvárja, amíg a process futása véget ér, azonban ezt úgy teszi, hogy addig a process-t tartalmazó szál futását megállítja. Ami a főprogram átmentei megállítását jelenti, azaz addig lényegében lefagy a program.

Ennek kiküszöbölése céljából csináltam egy QThread-ből származtatott osztályt, az *ebookConvThread*-t, aminek a run() metódusát felülírva egyszerűen egy, a fő programszáltól különböző, új szálra rakhatom át az utasításokat, a várakozást. Igaz, hogy ekkor ez a szál fog átmenetileg leállni, azonban ez nem befolyásolja a főprogramot, ami így használható marad, nem fagy le. A run() metódus a konvertálás végeztével küld egy signal-t, aminek hatására a főprogram lezárja ezt az új szálat és megszünteti.

Ez azonban egy másik hibalehetőséget vetett fel bennem, ha már nem áll le a főprogram, mi történik, ha a konvertálás előtt a forrás fájlt töröljük? Azért, hogy zavaró műveletet ne lehessen végezni a konvertálás közben, az az ötlet támadt, hogy mi lenne, ha lenne egy várakozó animáció, amíg a konvertálás befejeződik?

Röviden: így született meg a *Maze* és a *waitDialog* osztály. Bővebben: amikor a főprogram Model osztálya példányosítja és elindítja az *ebookConvThread*-et küld egy signal-t a *View* felé, ami hatására a *View* létrehoz egy példányt a *waitDialog*-ból, ami a főablak elé megjelenve, blokkolja azt, miközben egy *Maze* példányt hoz létre, ami labirintust generál és rajzol a képernyőre másodpercenként. Ezt addig csinálja, amíg a konvertálás be nem fejeződik. Ha a szál felől érkezik a befejezést jelző signal a *Model*-hez, a *Model* két signal-t is küld megint *View*-nak, egyrészt, hogy leállítható és

eltüntethető az animációs ablak, másrészt, hogy kiírható a konvertálás befejezéséről tájékoztató üzenet.
A *Maze* működéséről kicsit részletesebben a következő alfejezetben.

A sikeres végrehajtás után az új formátumra konvertált e-book megjelenik a jobboldali panelen, a kiindulási e-book alatt/fölött, függően attól, hogy az új kiterjesztés az ABC szerint hol helyezkedik el a forrás kiterjesztéséhez képest (a fájlkezelő ABC szerint rendezi a fájlokat, mappákat).

3.2.15 Maze animáció

A *Maze* osztály a QWidget-ből származik és lényegében olyan, mint egy önálló program a programon belül.

A konstruktorban beállítok néhány értéket a megjelenítéshez, méreteket, majd meghívok egy generateMaze() nevű metódust, ami jelenleg egy rekurzív algoritmust használok a labirintus generálásra. Kipróbáltam a Prim algoritmust is, azonban személy s nekem jobban tetszik a rekurzív által generált eredmény. A 11. ábrán a rekurzív, a 12. ábrán a Prim algoritmus alapján generált labirintus látható.
Számomra kissé zavaró a Prim algoritmus által generált labirintusban az ismétlődő „rekeszek” jelenléte, így végül a rekurzív algoritmust választottam.

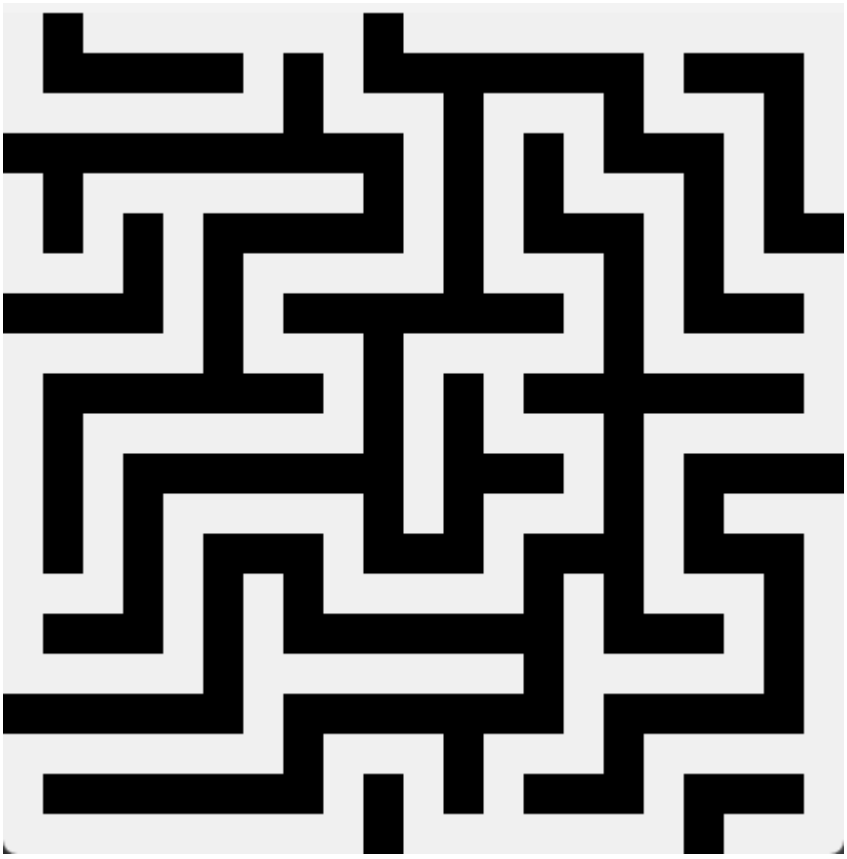
Fő különbség a két algoritmus közt, hogy a rekurzív alapvetően véletlenszerűen választ egy kiindulási pontot, majd a pont szomszédain rekurzívan (tehát annak a szomszédain, és azoknak a szomszédain..) haladva véletlenszerűen választ ösvényt vagy falat. Ezzel a módszerrel jellemzően olyan labirintus generál, amire a hosszabb, kanyargós ösvények jellemzőek, gyakran egy megoldással.
Ezzel szemben a Prim algoritmus minimális feszítőfákat generál. Kezdő mező után annak az összes szomszédját megjelöli, mint lehetséges útvonal, majd közülük véletlenszerűen egyet választ, ezt összekötve az aktuális cellával megjelöli, mint a labirintus része. Az így létrehozott labirintusban az útvonalak egyenletesebbek, nem hosszú és kanyargósak, mint a rekurzív esetén.

Visszatérve a működésre, konstruktorban elindul egy QTimer is, aminek a „timeout” event-jére rá van kötve a generateMaze(), míg időköznek 1000ms-t, azaz 1 másodpercet

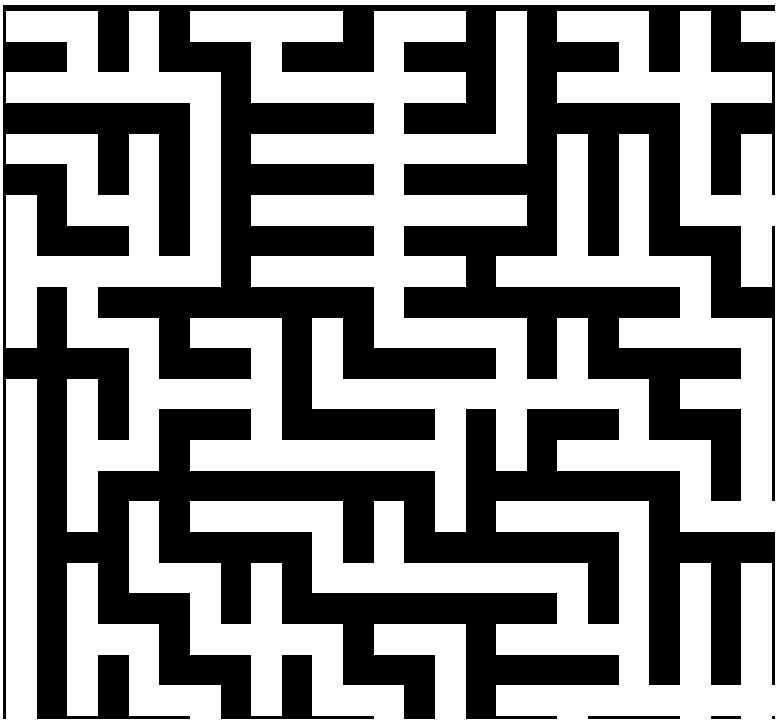
adtam meg, így másodpercenként generál egy új labirintust, így létrehozva egy amolyan animációt.

Mivel származtatott osztály, így bármikor hívható a QWidget-beli update(), amely hatására a saját paintEvent() metódusom reagál, ami újra rajzolja.

A tényleges megjelenítésért a QDialog osztályból származó waitDialog a felelős, ami lehetővé teszi azt, hogy amíg nem érkezett meg az e-book konvertálás befejezését jelző signal, addig a főprogram ablakkal ne lehessen interakciót kezdeni.



11. ábra, Labirintus rekurzív algoritmussal



12. ábra, Labirintus Prim algoritmussal

3.2.16 Kép konvertálás

Az e-book konvertálás után elérkeztem az utolsó célkitűzésemhez, a képkonvertáláshoz. Ennél a feladatnál viszonylag keveset kellett kutatnom, hogy megtaláljam a QT QImage osztályát, ami szintén a nevéből adódóan képek kezelésére biztosít lehetőséget. Számomra a kitűzött formátumok a JPEG, PNG és BMP voltak. Szerencsémre ezen formátumokat olvasási és írási műveletekkel is támogatja az osztály, így könnyen megoldható a konvertálás.

A már megszokott fájl (index) ellenőrzés, valamint név tagolást követően, az e-book konvertálásnál látott módon kiválasztható, majd ellenőrzésre kerül a kívánt formátum. Amennyiben minden bemeneti adat helyes, úgy az eredeti képből létrehozok egy QImage-t, amely utána az új kiterjesztéssel menthető. Természetesen az új név létrehozása után a QImage képként való mentésekor ellenőrzést is végzek, a QImage::save() metódus ugyanis egy logikai értékkel, igazzal vagy hamissal tér vissza sikeresség, vagy sikertelenség esetén. Az új kép, az e-book-hoz hasonló módon, az eredeti mellett jelenik meg a fájlok közt.

3.2.17 A kész program build-elése, deploy-olása

A kész program az IDE-ből való build-elés után a forráskódot tartalmazó mappa mellé generált build mappába kerül. Azonban ekkor még dll-ek (a program futásához szükséges állományok) hiányoznak, amit a következőképp lehet orvosolni: Windows start menüből el kell indítani a QT parancssort, ami nálam QT 6.4.0 (MinGW…) néven elérhető, majd ebben a konzol ablakban kell kiadni a **windeployqt program.exe** parancsot, a program.exe helyére a fájl teljes elérési útját helyettesítve. Ez bemásolja az összes függőségét a programnak a mappába, így ha ez a mappa kerül át egy másik Windows PC-re, a program azonnal futtatható (persze az e-book konvertálás nem fog működni, ha a Calibre nincs telepítve).

Linux esetén a probléma kicsit árnyaltabb, nem találtam hivatalos QT tool-t ezen művelet végrehajtásához, azonban találtam egy github repository-t [12], ahol a feltöltött tool elvileg ez a feladatot látja el linuxon. Sajnos azonban nekem nem működött, a github-on is a lényegi frissítések elég régiek.

3.3 Tesztelés

Jelen program esetén a tesztelés manuálisan zajlott.
A program fejlesztése során ügyeltem a „bolondbiztosságra”, ilyen például, hogy nem támogatott formátumot ne lehessen kiválasztani konvertáláshoz, vagy átnevezéskor ne lehessen üres nevet adni a fájlnak. Ezen gondolatmenet alapján logikus volt, hogy a programot felhasználói oldalról tesztelem, hibás és helyes bemenetekkel, gombnyomásokkal. Természetesen a fejlesztés folyamán voltak debug jellegű, kiíratásos ellenőrzések tesztelések is, amik főképp a fájlok elérési útját, nevét foglalták magukba, hiszen string műveletek végzése miatt ezek voltak a legkényesebb részek, ahogy ez fentebb olvasható, például a 3.2.12 fájl átnevezés alfejezetben, ahol a fájl útvonal bontásánál kiesett egy / jel.

A tesztelés során papíron kidolgozott teszteket hajtottam végre, amelyek a program elvárt működéséből és lehetőségeiből bármikor rekonstruálhatóak is. A legtöbb esetben a különböző funkcióknál az adott funkció használatához szükséges előfeltételek tesztelése történt, így ellenőrizve, hogy ha a szükséges előfeltételek teljesülnek, akkor a program a vártnak megfelelően működik, míg hibás előfeltétel esetén az adott funkcióhoz tartozó

kódrészlet, metódus nem fut le, helyette hiba vagy tájékoztató üzenetet küld, majd a program bármilyen összeomlás, működést akadályozó hiba nélkül fut tovább.

3.3.1 Fájl törlésének tesztelése

Fájl törlésénél a tesztelés gyorsan elvégezhető, amennyiben nincs kijelölt fájl, úgy hibaüzenet fogad, hogy nincs kijelölt fájl. Amennyiben van, úgy az adott fájlnak a nézetből el kell tűnnie, mivel törlődött, ennek teljes ellenőrzésére a fájlt figyeltem a Windows fájlkezelőből is.

3.3.2 Fájl átnevezésének tesztelése

Ahogy a korábbi, a funkció fejlesztését leíró részben is írtam, itt fordult elő egy elég érdekes hiba, miközben manuálisan teszteltem. Alapvetően, ha nincs kijelölt fájl, akkor arra érkezik hibaüzenet, ha nem adunk meg semmit új névnek, azaz üresen hagyjuk, arra is érkezik, míg, ha bármit beírunk, ami szövegként kezelhető, akkor az lesz a fájl új neve. Ennek ellenére eleinte ez nem teljesen működött így, tehát egy alaposabb tesztelésre volt szükség. Erre a megoldás egy meglehetősen hagyományos, egyszerű módszer, std::cout-ra (standard kimenet) kiírattam a programmal, hogy mi a kapott név, mi a beolvasott új név, mi a létrehozott teljes új név, ami elérési utat is tartalmaz. Ezzel fény is derült a hibára, hogy amikor az elején a teljes elérési utas nevet bontottam, akkor az elérési út utolsó / jele levágásra került, így hiányzott, amikor a teljes új nevet összeállítottam. Ezen hiba javítása után már az elvártnak megfelelően működött az átnevezés.

3.3.3 Keresés tesztelése

A keresés tesztelése érdekes volt, itt ugyanis az volt a megoldás, hogy létrehoztam egy teszt mappa és fájl struktúrát egy külső meghajtón, pendrive-on, majd ezen környezetben hajtottam végre a tesztelést. Értelemszerűen, amennyiben a keresendő fájlnév üres, keresés nem végezhető.

A tesztelés másik esete, hogy kifejezésként, azaz QString-ként kezelhető (lényegében bármilyen karakter sorozat) bemenet esetén, ha rossz, vagy ha jó helyről indul keresés, mindkét esetben az elvárt eredmény, rendre hibaüzenet, vagy a fájl(ok) megjelenítése történjen meg. A jó helyről való keresés tesztelése is bontható, hogy eredendően a fájl mappájában indul a keresés, vagy egy külsőbb mappából, amikor is már a rekurzivitás is

láthatóvá válik, azaz, hogy egy mappában lévő mappán belül is megtalálja a keresett fájlt (ez értelemszerűen egy megtalálható fájl esetén ellenőrizhető könnyen, vagy ha több hasonló fájl eltérő mappákban van, akkor a találati listán látható).

3.3.4 Az e-book konvertálás tesztelése

Az e-book konvertálás tesztelésénél kiderült, hogy kifejezetten jó választás volt a Calibre, mint teljes e-book manager program, hisz így a konvertálások végeztével maguk az e-book-ok megnyithatóak ezzel a programmal, ellenőrizhető az is, hogy jól zajlott le a konvertálás (igaz, magát a konvertálást a Calibre végzi, így ez részint annak is tesztje).

Itt negatív teszt, ha a forrással megegyező formátumra akarunk konvertálni, vagy ha a forrás nem támogatott formátum, esetleg nincs fájl kijelölve. Ezekre rendre hibaüzenet az elvárt viselkedés.

Ezek után már csak a pozitív teszt maradt, amikor megfelelő, támogatott formátumú e-book fájl van kijelölve és eltérő kívánt formátum került kiválasztásra. Ekkor a külső konvertálás meghívásra kerül, megjelenik az animáció (és nem lehet a főablakra kattintani), és jellemzően pár másodperc után, amikor a konvertálás befejeződött, az animáció eltűnik, értesítő üzenet a konvertálásról és visszatérünk a főablakhoz, ahol megjelenik a fájl nézetben az új formátumú e-book.

3.3.5 Kép konvertálás tesztelése

Ezen funkció tesztelése, felépítésének köszönhetően lényegében megegyezik az e-book konvertálás tesztelésével, annyi módosítással, hogy itt más kiterjesztések támogatottak és nincs várakozó animáció (a képfájlok konvertálása jellemzően sokkal gyorsabb, mint az e-book-oké).
Ha minden helyes, akkor a konvertálás végeztével a sikerességről tájékoztató üzenetet dob a program.

3.4 További fejlesztési lehetőségek

Ahogy látható, ebben a programban bőven maradt lehetőség a fejlesztésre.

Grafikus oldalról komoly fejlesztés lehet a kinézet átdolgozása, esetleg a jelenlegi mappa/fájl nézetek helyett két teljes nézet, mint ami például a Total Commander fájlkezelő esetén van.

Funkcionális oldalról egyrészt lehetséges további fájlkezelési funkciók hozzáadása, másrészt pedig a konvertálások esetén több formátum támogatása. Ugyanígy a konvertálásnál lehetőség, hogy a komplett Calibre program helyett például csak a konvertálást végző egységet kód formájában beépíteni a programba, így nem kellene külső programhívást végezni. Persze itt ellenérv is lehet, hogy a Calibre telepítéssel kapunk egy programot, ami megnyitni is képes az e-book-okat.

4. Összefoglalás

A dolgozatom témája és célja tehát egy olyan fájlkezelő és konvertáló alkalmazás, amelyben alapvető fájlműveletek, például átnevezés, keresés, lehetséges bizonyos fájltípusok, e-book-ok és képek népszerű formátumai közötti konverzió.

A célként kitűzött feladat megoldása közben lehetőségem nyílt megismerkedni a QT keretrendszerrel, mely többek között a C++ programozási nyelv használatával biztosít lehetőséget grafikus kezelő felülettel rendelkező alkalmazások fejlesztésére. Megtapasztaltam az önálló munkavégezést és problémamegoldást egy, az eddigieknél sokkal nagyobb léptékű, összetettebb feladattal, így a munka világában is hasznos tapasztalatra tehettem szert. A megoldáshoz hozzájárultak továbbá az egyetemi tanulmányaim során szerzett ismertek, amelyek közül kiemelhető a C++ programozási nyelv alap szintű ismerte, valamint az objektum orientált programozás elmélete, hiszen a QT irányából végig különböző osztályokat, és azok metódusait használtam, alkalmaztam. A feladat megoldása során fellépő problémák, hibák szintén segítették a szakmai irányú fejlődésemet, sokat tanultam belőlük.

4.1 Köszönetnyilvánítás

Szeretném megköszönni a sok segítséget konzulensemnek, Pataki Norbert tanár úrnak, aki a dolgozat elkészülte alatt folyamatosan elérhető volt, ha megakadtam iránymutatást, tippet adott, miközben végig tanácsokkal látott el az egész szakdolgozat teendőivel kapcsolatban.

Forrásjegyzék

[1] „QT 6 documentation,” [Online]. Available: https://doc.qt.io/qt-6/. [Hozzáférés dátuma: 15 10 2022].

[2] „Calibre,” [Online]. Available: https://calibre-ebook.com/. [Hozzáférés dátuma: 10 12 2022].

[3] „Youtube tutorial QFileSystemMOdel,” [Online]. Available: https://www.youtube.com/watch?v=92biLZST6Vg. [Hozzáférés dátuma: 31 10 2022].

[4] „Stackoverflow for upenUrl,” [Online]. Available: https://stackoverflow.com/questions/27061701/qdesktopservicesopenurl-fails-if-path-contains-spaces-after-apt-get-upgrad. [Hozzáférés dátuma: 26 10 2022].

[5] „Stackoverflow for file rename,” [Online]. Available: https://stackoverflow.com/questions/3648839/get-filename-from-qfile. [Hozzáférés dátuma: 29 11 2022].

[6] „QT 5.15 documentation,” [Online]. Available: https://doc.qt.io/qt-5.15/. [Hozzáférés dátuma: 14 11 2022].

[7] „Stackoverflow for file search,” [Online]. Available: https://stackoverflow.com/questions/19310412/findsearch-particular-named-file-from-qfilesystemmodel. [Hozzáférés dátuma: 30 11 2022].

[8] „Aspose.Words for C++,” [Online]. Available: https://products.aspose.com/words/cpp/. [Hozzáférés dátuma: 18 11 2022].

[9] „Stackoverflow for elemnt in list/ebook,” [Online]. Available: https://stackoverflow.com/questions/24139428/check-if-element-is-in-the-list-contains. [Hozzáférés dátuma: 10 12 2022].

[10] „Stackoverflow for CMD from qt,” [Online]. Available: https://stackoverflow.com/questions/33865731/how-to-run-a-windows-cmd-command-using-qt. [Hozzáférés dátuma: 10 12 2022].

[11] „Stackoverflow for QPorcess wait,” [Online]. Available: https://stackoverflow.com/questions/14504201/qprocess-and-shell-destroyed-while-process-is-still-running. [Hozzáférés dátuma: 10 12 2022].

[12] „GitHub - linuxdeployqt,” [Online]. Available: https://github.com/probonopd/linuxdeployqt. [Hozzáférés dátuma: 20 05 2023].