MŰSZAKI INTÉZET
GÉPÉSZMÉRNÖKI ALAP
Mérnök Informatika specializáció

SZAKDOLGOZAT

feladatlap

Deák Tamás *(YWJXE1)*

részére

A diplomadolgozat címe:

Maradék anyagkezelés lemezmegmunkáló szoftverben

Feladatkiírás:
A kiindulási adatok, a megoldandó szakmai feladat és/vagy probléma, illetve az elvégzendő feladatok rövid (3-4 soros) megfogalmazása, műveltető megfogalmazásban.

.....

.....

Közreműködő tanszék: *Műszaki Intézet /Gépszerkezettani Tanszék*

Külső konzulens: *Veres Tibor Ügyféltámogatási vezető*, 2335. Taksony, Vezér utca 17.

Belső konzulens: *Maradász István*, MATE, Műszaki Intézet

A dolgozat beadási határideje: 2024. év 11. hó 5.nap

Kelt: Budapest, 2024. év 10. hó 24. nap

Jóváhagyom		Átvettem
(tanszékvezető)	(szakfelelős)	(hallgató)

A dolgozat készítőjének külső konzulense nyilatkozom arról, hogy a hallgató az előre egyeztetett konzultációkon megjelent.

Kelt: Budapest, 2024. év 10. hó 24. nap

(külső konzulens)

SZAKDOLGOZAT

Hallgató neve
Deák Tamás

ÉV 2024

Magyar Agrár- és Élettudományi Egyetem
Szent István Campus
Mérnökinformatikai központ
Gépészmérnöki alapképzési szak

Maradék anyag kezelés a lemezmegmunkáló szoftverben.

Belső konzulens: Madarász István
Mestertanár

Belső konzulens intézete/tanszéke: **Műszaki Intézet /Gépszerkezettani Tanszék**

Külső konzulens: Veres Tibor
Ügyféltámogatási vezető

Készítette: **Deák Tamás**

Budapest
év 2024

Tartalomjegyzék

1 Bevezetés és célkitűzések

A modern lemezmegmunkálás egyik folyamata amikor az alkatrészek terítékeit kivágjuk a lemez táblából. Ezen terítékeket szeretnénk úgy elhelyezni az elő gyártmányt képző lemeztáblán, hogy a lehető legjobb kihasználtságot érjük el. A mai lemezmegmunkáló szoftverek egyik fő funkciója, hogy ezt a lemez kihasználtságot hatékony kiosztó algoritmusok segítségével maximalizálják. Viszont a kiosztás során előfordulhat, hogy jelentős szabad terület marad a táblán, amit később esetleg még szeretnénk felhasználni. Ezért igényünk, hogy legyen lehetőségünk ezt a maradék területet elsőnek valamilyen raktár kezelési megoldással leltározni. Majd újra felhasználni, ha az ugyan ilyen anyagmegnevezésű és lemezvastagságú alkatrész terítéket kell megmunkálnunk.

Ezen szakdolgozatban a RADAN nevű CAD-CAM szoftver segítségével mutatok be egy megoldást a maradék anyag kezelésre, amivel nem csak a raktár kezelést megoldására mutatok példát, hanem az esetleg felmerülő selejtek automatikus újra kiosztására is. A selejtekkel kiosztás szakaszában még nem lehet tervezni. Viszont, ha a lemezvágó gépen lefutott a vágási program akkor már képesek lehetünk meghatározni, hogy hány darab lett jó alkatrész a táblán és ebből a selejt szám is adódik. Egyszerűen a darab lejelentéssel eltudjuk küldeni az adatokat egy adatbázisnak és már is automatikusan kitudjuk újra osztani az esetleges alkatrész terítékeket.

Ha már újra vágásra van szükség a jobb kihasználtság végett nyugodtan oszthatunk ki egy még érintettlen nagy táblára is pár alkatrészt vagy akár már a folyamatban eddigre keletkező esetleges maradék táblára is. Fontos viszont, hogy a maradék táblára csak abban az esetben tervezhetünk újabb kiosztást, ha ez a tábla rendelkezésünkre áll. Ha még nem futott le a lézergépen akkor a raktár készletből még hiányzó táblára kellene megmunkáló programot írnunk, ami talán sosem futna le a nem megfelelő sorrendiség miatt.

Ezen eljárás kivitelezéshez, rendkívül fontos, hogy a megfelelő megnevezésű kiosztás a megfelelő tábla névre kerüljön. Általános esetben „végtelen” mennyiségben határozzuk meg a raktárban lévő tábláinkat, hogy a szabványos méretek miatt ne egyesével kelljen felvinni az adatokat. Viszont a valóságban ezek a táblák pár mm-el nagyobbak szoktak lenni mind két irányba, mert a darabolás gyártási hibával készül. Ha viszont ezeket a táblákat kg mennyiségben vesszük akkor könnyen lehet, hogy az éves leltárnál a termelés volumenétől függően akár több tonna különbség is összejöhet. Sajnos a szakdolgozatom keretein belül ezen probléma megoldásával csak a maradék táblák méretein belül foglalkozok.

A szakdolgozatom azt a célt tűzi ki maga elé, hogy megoldást nyújtson a maradék táblák raktározására és lehetőséget adjon az automatikus szoftver használatra. Ezen automatizációknak következő feladatok elvégzésére kell képesnek lenniük:

- Lehetőség a maradéktábla automatikus lementésére a CNC gépen történt futtatás után.
- A már lementett maradék táblát betölteni a RADAN szoftver, anyagtáblák listájába.
- Készítsen automatikus CNC kódot, amit továbbít a szerszámgépre.

2 Szakirodalmi áttekintő

Lemez alkatrészek. A lemezalkatrészek fő meghatározó jellemzői, hogy az egyik irányban lényegesen kisebb a kiterjedése és a felülete sokkal nagyobb, mint a kerülete. Lemez alkatrészeket napjainkban is sok helyen alkalmaznak. Mint például autók-, számítógéphát-, szellőzőrendszer burkolatai. De lemez alkatrészekből épülnek fel különböző tartó szerkezek, lépcsők is.

A lemez alkatrészeket egyik megmunkálási módja amikor hajlítjuk egy ív mentén az alkatrészt. Ebben az esetben a hajlításnak két oldala lesz. Az egyiknek kisebb rádiusza lesz, itt nyomó igénybevételnek van kitéve az anyag. Ekkor beszélhetünk zömülésről. A hajlítás másik oldala lesz a külső nagyobb rádiuszú. Itt az anyag húzó igénybevételnek van kitéve. A két oldal között viszont létezik egy úgy nevezett semlegesszál. A semlegesszálra ott alakul ki, ahol a húzó és a nyomó igénybevétel nulla. Általában s-el jelölik. A semleges szál azért fontos, mert ez határozza meg, hogy mekkora alkatrészt kellene kivágnunk a lemez táblából, hogy majd a hajlítás után méret helyes legyen az alkatrész. A semleges szállal számolt alkatrész befoglaló geometriáját hívjuk terítéknek. (Fledrich G, et. al. 2017)

A megmunkálást, a hajlítást követően az alkatrész a rugalmas alakváltozást is elszenvedi. Ami azt jelenti, hogy túl kell hajítani az alkatrészt, hogy a megmunkálást követően a véggeometria megfelelő legyen számunkra. Ennek a túlhajtásnak az értékét tapasztalati úton vagyunk képesek meghatározni.

„A hajlításkor a lemez szálai részben nyomásra, részben húzásra vannak igénybe véve. A hajlítóbélyeg az anyag felé eső részét nyomásra a külső hajlító prizma felé eső rést pedig húzásra veszi igénybe. A két feszültség választó vonala a semleges réteg, amelyben a feszültség értéke nulla. Mivel itt nem lép fel erő, a semleges réteg eredeti hosszúsága változatlan.” (Fledrich G, et. al. 2017 p 140.)

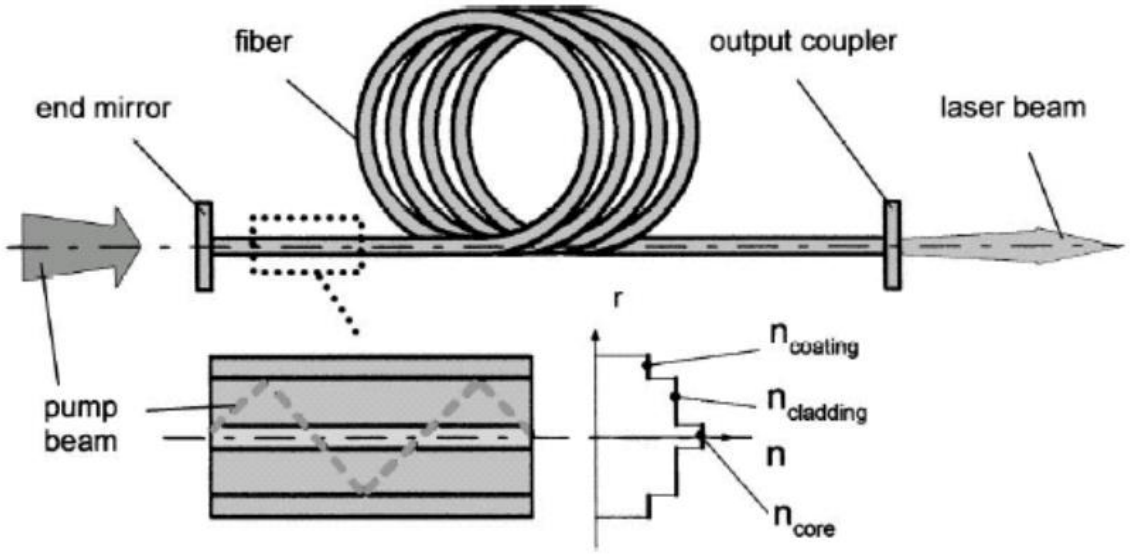
Az említett hajítás a külső felületen húzásra még a belső felületen nyomási igénybevételnek van kitéve. Belül zömül az anyag, még kívül kirepedhet. Viszont a lemez vastagságon belül létezik egy olyan vonal menti geometria, ahol ki egyenlítődnek ezek az igénybevételek és itt a geometria nem változik. Ezt nevezik semleges szálnak, ami az alkatrész eredeti hosszát megőrzi.

2.1 Lézermegmunkálás

A lézereres lemezvágásnál lokális energiabevitel történik, ami megolvassza az anyagot és a magas nyomású gázok fújják a vágórésből az anyagot. „A lézerrezonátor egy üvegcsőből és a két végét lezáró tükörből áll. A csőben gázkeverék van, amely CO_2 (5%), N_2 (15%), He (80%)

gázok elegye, ahol a CO_2 biztosítja a lézerátmenetet. A N_2 gáz a gerjesztésben segít, a He pedig az alap-energiaszintre való visszajutásban és a hőelvezetésben játszik szerepet.” (Markovits, 2018, p. 3).

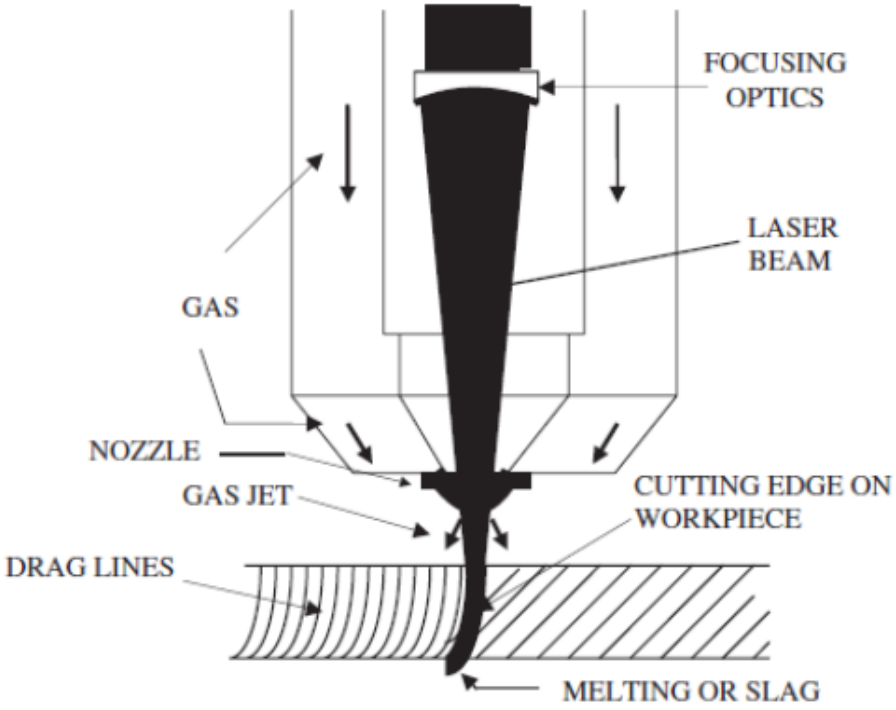
Ezt a lézerfényt a szerszámgépeken egyedi tükör rendszerrel vezették el a generálási helytől a vágási felülethez. Az ügy nevezett Fiber gépek azért terjedtek el napjainkban mert a bonyolult tükörrendszert felcserélte és a hely igényét nagy mértékben lecsökkentette az üvegszál használata. Ezt szemlélteti a következő ábra.



1. ábra Tükörrendszer megoldása.

Forrás: Hügel(2000)

A vágás minősége és vágott felület is fontos paraméter a lézer vágott lemez alkatrészeket tekintve. Még egy 1mm-es alkatrész vágott oldalán a kerületi felület nem jelentős mértékű így ott nem lehet észrevenni, de egy 25mm vastag anyagon, ahol pontosan látszik, hogy más-más felületi érdesség jelenik meg a vágási magasságtól függően.



2. ábra Lézeres vágás vetületi képe

Forrás: (Dubey - Vinod 2008)

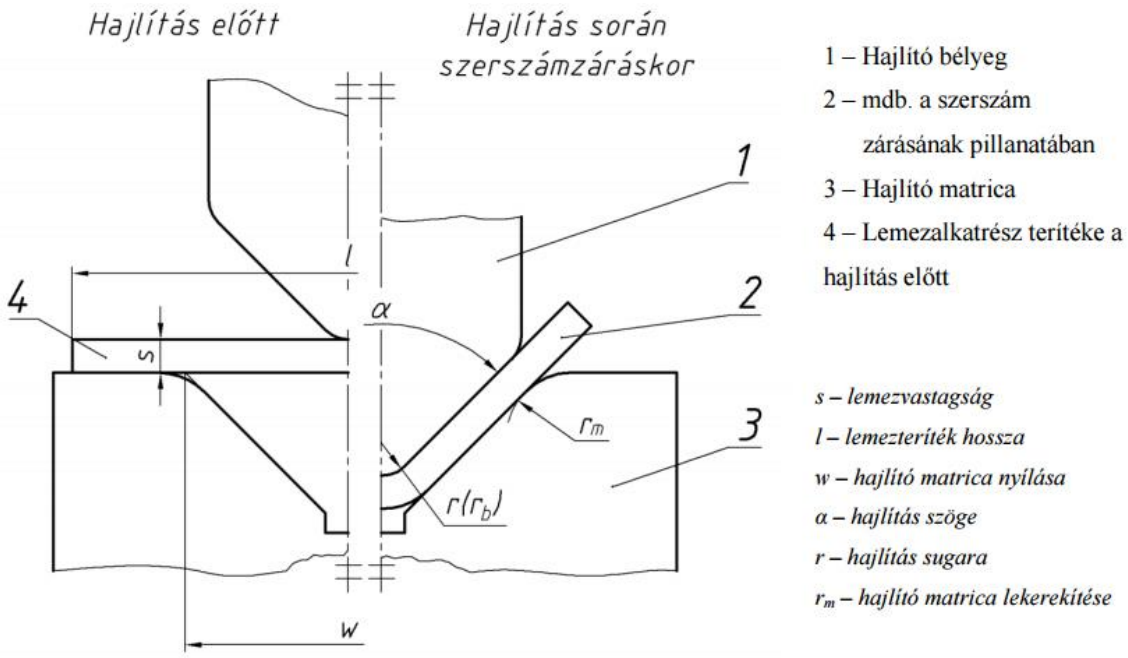
Az említett vágási paraméterek több mindentől is függenek. „The quality of cut solely depends on the setting of process parameters like cutting speed, focal point, laser power, assist gas pressure etc.(Senthilkumar, 2014, pp. 44.-48.)

TRUMPF Werkzeugmaschinen GmbH + Co. vállalat Operator's manual – TRUMPF LASERCELL 1005 07/2004-es kiadású kézikönyvének 3. fejezetében olvasni lehet egy modern lézergép felépítéséről. A lézervágó gépek az esetek nagy többségében CNC szerszámgépként működnek. Ezekre a gépekre igaz, hogy számítógép által vezérelt mozgásra és sebességre képesek. A szánokat léptető motorok mozgatják és mikrométeres pontoságra is képesek. A CNC Computer Numerical Controll rövidítése. Régebben lyukszalaggal működtek ezek a gépek, de a modern technikának köszönhetően a gépek többsége (létezhet olyan, amit még nem alakítottak át, de mára már nem gyártanak ilyet) már vezérlővel rendelkezik. Ezek a vezérlők képesek .NC fájl-t olvasni. A .NC fájlok iso G kódot tartalmaznak, amik főként koordinátákat, vágási sebességeket tartalmaz. Vannak úgy nevezett M kódok is benne, amik gép funkciókat kapcsolnak. Léteznek egyedi vezérlőre szabott fájlok is, amik általában a vezérlő saját fájl kiterjesztése. Ilyen például a Sinumerik 840D vezérlő is. Egy ilyen fejlettebb vezérlővel már külön alprogramokat és függvényeket is lehet írni, ami a rugalmas programozás hasznos eszköze lehet. (TRUMPF LASERCELL 1005 Edition: 07/2004)

Az említett út és pálya adatokat lehet a szerszámgép vezérlőjén megírni. De modern számítógépes szoftverekkel hatékonyabb megmunkálást lehet végezni. Ezeket a megoldásokat CAM szoftvereknek nevezzük. A CAM Computer Aided Manufactoring rövidítése. A számítógéppel segített gyártás előnye, hogy még a vezérlőn készülő program programozási ideje alatt a szerszámgép képtelen működni, gyártani. Addig egy CAM szoftverrel könnyedén képesek vagyunk arra, hogy amíg a szerszámgép dolgozik addig mi előre elkészítsük a megmunkálásokat. Ezen megmunkálásokkal képesek lehetünk futási időt megközelítőleg meghatározni, ha tudjuk a szerszámgép gyorsulását, lassulását, a lyukasztási idejét és egyéb paramétereket. A CAM szoftverek másik nagy előnye, hogy jobb átláthatóságot nyújt a felhasználó számára. A legtöbb CAM szoftvernek van grafikus felülete, ahol a szerszám pályát, alkatrészeket, ráállásokat, leállásokat grafikusan is képesek lehetünk ellenőrizni. Addig egy vezérlőn írt szerszám pálya megjelenítés általában kevesebb információt tartalmaz.

A modern üzemekben a hajlítás CNC hajlító géppel végzik. Ezek a gépek nevéből adódóan lemez hajlításra valók. Két szerszámmal dolgozik ezek fent és lent helyezkednek el.

1. Bélyeg: Ez a hajlítóeszköz az, amelyik a lemez egyik oldalát befolyásolja. A hajlító bélyeg olyan kemény és erős anyagból készül, amely ellenáll a nyomásnak és a deformációnak.
2. Matrica: Ez a hajlítóeszköz az, amelyik a lemez másik oldalát befolyásolja. A hajlító matrica általában az alakot adja a hajlítandó lemeznek.



3. ábra: hajlítás elve

Forrás: https://www.cnc.hu/

Ahogy a lemez a hajlító bélyeg és a hajlító matrica között mozog, a megfelelő alakot veszi fel. (Fledrich G, et. al. 2017)

A hajlítás során az anyagok különböző részei különböző igénybevételeknek vannak kitéve, például nyomásnak és húzásnak. Ez a hajlító bélyeg és a hajlító matrica alkalmazása során jön létre, amelyek nyomást és húzást gyakorolnak az anyag különböző területeire.

Emellett az anyagok deformációja során meg kell vizsgálni a hajlításnak kitett anyagok zömülését és nyúlását. A zömülés az anyag vastagságának növekedése a hajlítás folyamán, míg a nyúlás a hajlított rész hosszának változása.

Az anyagok viselkedése a hajlítás során számos tényezőtől függ, beleértve az anyag típusát, vastagságát, hajlítási sugarát és a hajlítás sebességét. Ezeket a tényezőket figyelembe kell venni a hajlítási folyamat optimalizálása és az alkatrész minőségének biztosítása érdekében.

A már említett hajlításoknál fellépő nyúlások és zömülések miatt a lemez alkatrész méreteit nehéz meghatározni. A semleges szál megközelítő számításával is az eredeti befoglaló méretek kismértékben, de változnak. Mivel mi szeretnék ezt az alkatrészt egy táblából kivágni így pontos méretekre lenne szükségünk. A geometriát, ami a semleges szállal számolt módosított geometriája az alkatrésznek, terítéknek hívjuk.

A megmunkálás egyik fő fázisa amikor meghatározzuk a kivágandó alkatrészek helyzetét a nyers táblán. CNC megmunkálást feltételezve ezt elvégezhetjük általában a szerszámgép vezérlő egységén és egy erre a célra írt CAM szoftverrel is.

Az alkatrészek bizonyos területet fednek le a tábla felületéből. Minél jobban fedik le az alkatrészek a táblán annál jobb lesz a tábla kihasználtsága.

A fiber lézer a legmodernebb lézer technológiák közé tartozik. Akár 50kW-os teljesítményre is képes így sokkal hatékonyabb, mint a plazma gépek. Fontos megjegyezni, hogy méretben is jelentősen kisebb helyigénye van így a szerszámgépek mérete nem nő jelentős mértékben. A lézergépekre jellemző, hogy viszonylag kis vágóréssel dolgoznak, ami kis mértékű anyagvesztéssel is jár. A vágórés általában 0.2-0.9mm között mozog technológiától és anyagvastagságtól: Természetesen nagyon vastag anyagoknál lehet 1- 1,5 mm is. A plazmára viszont jellemző, hogy sokkal vastagabb vágóréssel dolgozik így a kezdő érték itt 1mm fölött van, ami anyagvastagság függvényében csak nőni tud. A fiber lezer gépek mellett szól még az is, hogy még a Plazma gépek csak vágni tudnak, addig a lézer gépeken lehet állítani a vágás erősségét így akár fólia égetést vagy jó minőségű gravírozást is végre tudunk hajtani velük.

2.2 Számítógéppel támogatott tervezés és hajlítás a lemezmegmunkálásban

A CAD (Computer-Aided Design) rendszerek segítségével a tervezők részletes 3D-modelljekeket hozhatnak létre az alkatrészekről, amelyek pontosan megjelenítik a tervezett

geometriát és tulajdonságokat. Ezek a modellek lehetővé teszik a tervezők számára, hogy vizsgálják az alkatrészeket a tervezési fázisban, és optimalizálják azokat az előrejelzést terhelési és hajlítási feltételek alapján.

A CAM (Computer-Aided Manufacturing) rendszerek lehetővé teszik az alkatrészek gyártásának tervezését és vezérlését a hajlító gépeken. Ezek a rendszerek képesek automatikusan generálni a hajlítási műveletekhez szükséges CNC kódot a CAD modell alapján. Ez lehetővé teszi a gyártók számára, hogy hatékonyan programozzák és irányítsák a hajlító gépeket, minimalizálva a hiba lehetőségét és optimalizálva a termelékenységet.

A CAD/CAM rendszerek továbbá lehetővé teszik az alkatrészek kihasználtságának maximalizálását a lemezeken, valamint az optimális vágási sorrendek kialakítását, hogy minimalizálják az anyagpazarlást és növeljék a tábla kihasználtságát.

2.3 Szoftver testreszabás

A RADAN nevű CAD CAM szoftver lemezalkatrészek megmunkálására ad megoldást. Stanc, Lézer, Plazma és kombi gép egyaránt programozhatók ezen szoftverrel. Piacvezető alkatrész kiosztó algoritmusával a lehető legjobb tábla kiosztást képes eredményezni. Ez lehetővé teszi a felhasználók számára, hogy a lehető legkevesebb maradék anyaggal dolgozzon, ami nem csak hulladék mennyiségét, de hatékonyságot is növeli. A kiosztás mellett testre szabhatóható Lézer és Plazma gépek esetén a rá/le állások értékei, a vágási kondíció, vágási sebesség és a kompenzációséért. Ezeken túl testre szabhatjuk a vágási sorrendet és ezt követően CNC kódot is készíthetünk, amit a szerszámgépen lehet futtain.

A C# (C Sharp) egy modern, objektumorientált programozási nyelv, amelyet a Microsoft fejlesztett ki a .NET keretrendszer részeként. A C# egy erős típusú nyelv, amely lehetővé teszi a hatékony és strukturált alkalmazások fejlesztését különböző platformokra, beleértve a Windows, az iOS és az Android operációs rendszereket is.

Az objektumorientált programozás (OOP) egy programozási paradigmát jelent, amely az adatok és a hozzájuk kapcsolódó műveletek közötti szoros kapcsolatot hangsúlyozza. Az OOP arra összpontosít, hogy az adatokat objektumokként reprezentálja, amelyek tartalmazhatnak adatokat (mezők) és azokon végrehajtható műveleteket (metódusok). Az OOP nagyban hozzájárul a kód újrafelhasználhatóságához, a fejlesztési idő csökkentéséhez és az alkalmazások strukturáltabbá tételéhez. (Albahari, 2023 p 1.-2.)

Az API egy olyan interfész, amely lehetővé teszi a szoftver más alkalmazásokkal és szolgáltatásokkal való kommunikációját és integrációját. Először is, fontos megérteni az API célját és funkcióit. Az API a fejlesztők számára biztosít egy közös módszert a szoftverek funkcióinak elérésére és felhasználására. Ezáltal lehetőséget nyújt olyan egyedi alkalmazások

és szkriptek fejlesztésére, amelyek további funkciókat és automatizációt hoznak létre a szoftverben rendszerben

A Visual Studio egy integrált fejlesztőkörnyezet (IDE), amelyet a Microsoft fejlesztett ki és tart karban. Ez az eszköz nagyon sokoldalú, és különféle platformokon és nyelveken támogatja a szoftverfejlesztést. Többek között C#-ot is. A Visual Studio számos funkciót kínál, amelyek lehetővé teszik a fejlesztők számára a hatékonyabb kódolást, tesztelést, hibakeresést és csapatmunkát.

A CSV (Comma-Separated Values) egy olyan adatformátum, amelyet széles körben alkalmaznak az adatok tárolására és átvitelére szövegalapú formában. A CSV formátum egyszerű és könnyen értelmezhető, mivel az adatokat szövegfájlban tárolja, és a mezőket vesszővel választja el egymástól. Az alapvető CSV formátum lehetővé teszi az adatok egyszerű tárolását, például adatbázisokból vagy táblázatkezelő programokból történő exportálásra, illetve adatok importálására más szoftverekbe vagy rendszerekbe. A CSV fájlok könnyen olvashatók és szerkeszthetők szövegszerkesztők vagy táblázatkezelő programok segítségével, ami növeli az általános használhatóságukat.

A Form a .NET keretrendszer egy olyan eleme, amivel ablakokat lehet létrehozni. Ezekben adatokat megjeleníteni és eseményt kiváltó folyamatokat kezelni.

Az XML (Extensible Markup Language) egy általános célú, hierarchikus adatstruktúra-leíró nyelv, amelyet az adatok tárolására és átvitelére használnak. Az XML-t elsősorban a webes és különböző alkalmazások közötti adatcsere egyszerűsítésére fejlesztették ki. XML-t gyakran használnak konfigurációs fájlok, adatbázis-mentések és egyéb strukturált adatokat tároló fájlok formátumaként. Egy tipikus XML dokumentum három fő részből áll (Molnár et al., 2018, p.17):

- XML Deklaráció: Meghatározza az XML verzióját és az esetleges karakterkódolást.
- Gyökérelem: Az összes többi elem ezen belül helyezkedik el.
- Elemtagok: Az adatokat hordozó címkék és attribútumok.

A C# programozási nyelvben a metódusok (methods) és függvények (functions) a program logikájának és funkcionalitásának alapvető építőkövei. Mindkettő kódblokkot jelent, amely meghatározott műveleteket végez, de a C# esetében gyakran metódusnak nevezik őket, függetlenül attól, hogy visszaadnak-e értéket vagy sem.

Az adatbázis egy strukturált gyűjteménye az adatoknak, amelyeket rendszerezett módon tárolnak és kezelnek annak érdekében, hogy azokat könnyen elérhetővé, kezelhetővé és

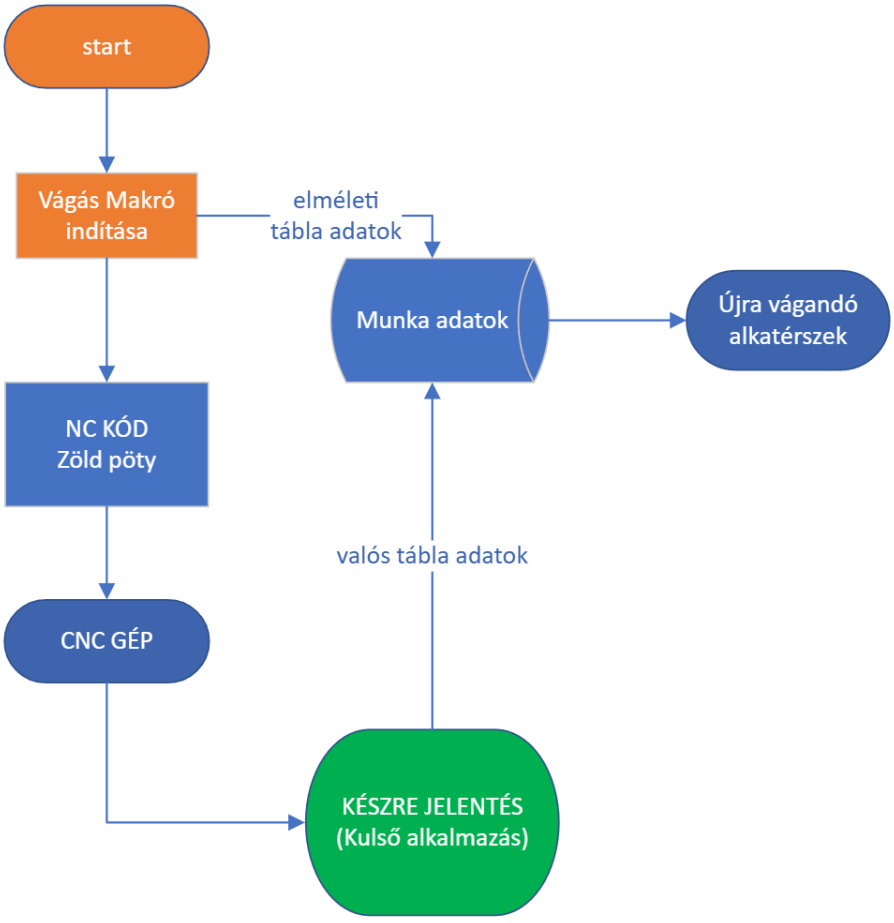
elemezhetővé tegyék. Az adatbázisok lehetővé teszik az adatok hatékony tárolását, visszakeresését és módosítását, valamint biztosítják az adatok integritását és biztonságát.

3 Folyamat tervezés

A fejlesztésem célja, hogy olyan folyamatot fejlesztek ki, amivel megoldható, hatékonyabb maradékanyag kezelés, selejtes alkatrészek újbóli kiosztása és a maradék táblák újra felhasználása.

Azért hasznos kidolgoznom ezt az eljárást mert a lemez megmunkálás folyamatában a legnagyobb költség az alapanyag, a lemez tábla. Ahogy az alkatrészeket kiosszuk a táblákra, egy bizonyos alkatrész közt kell tartanunk, hogy ne érjenek egymásba az alkatrészek és ha a pontozásokkor keletkezik olvadék ne, az alkatrészre kerüljön. Ezek az alkatrész közök és az alkatrészek furatai lesz majd a hulladék anyag. Ha az alkatrészek úgy vannak kiosztva, hogy marad nagyobb szabad felület akkor azt egy tábla levágással maradéktábla ként képesek lehetek raktározni. Ha ezt a maradék táblát újra fel tudom használni akkor lényegesen kevesebb veszteséggel tudom kezelni a már említett költséges alapanyag felhasználást.

3.1 Teljes folyamat



4. ábra:
Teljes folyamatábra

A folyamatot a RADAN szoftver segítségével fogom végre hajtani. A RADANban is vannak API-ok amivel képes vagyok testre szabni a szoftvert. A RADAN Projekt fálja és tábla fájljai mind xml kódúak. Ezt C#-ban könnyedén lehet olvasni és felhasználni, mint adat forrás. A fő adatokat az egyszerűsítés és nyomon követhetőség érdekében a Projekt nevével vagy táblaterv nevével ellátott CSV-ben mentem majd le.

- Első lépés:

Megoldani az alkatrészek automatikus betöltését a szoftverben:

Ezt egy CSV adatbázisból fogom betölteni a RADANba.

adatok: DXF elérési útvonala, anyag megnevezése, lemez vastagsága és darabszáma.

- Második lépés:

Felhasználható táblák automatikus betöltése a szoftverben

Ezt is egy CSV adatbázisból fogom betölteni. Ebben az adatbázisban fog szerepelni a raktáron lévő lemeztáblák listája.

adatok:

stock id, anyag megnevezés, lemez vastagság, befoglaló x méret, befoglaló y métert, felhasználható darabszám, lefoglalt darabszám.

- Harmadik lépés:

Alkatrészek kiosztása automatikusan, nc kód küldése.

Ebben a folyamatban már a szoftverben lévő Projekt file és mappa szerkezet biztosítja számomra az adatokat. Az elkészült táblakiosztásokat mappa és file figyeléssel egyszerűen lehet majd listázni és olvasni a Projektre vagy az aktuális táblakiosztásra vonatkozó adatokat.

Az nc kódokat hálózaton a megfelelő mappába kiküldöm így a szerszámgépen elérhetővé válik a folyamat végeztével.

- Negyedik lépés:

Alkatrész lejelentő alkalmazás létrehozása.

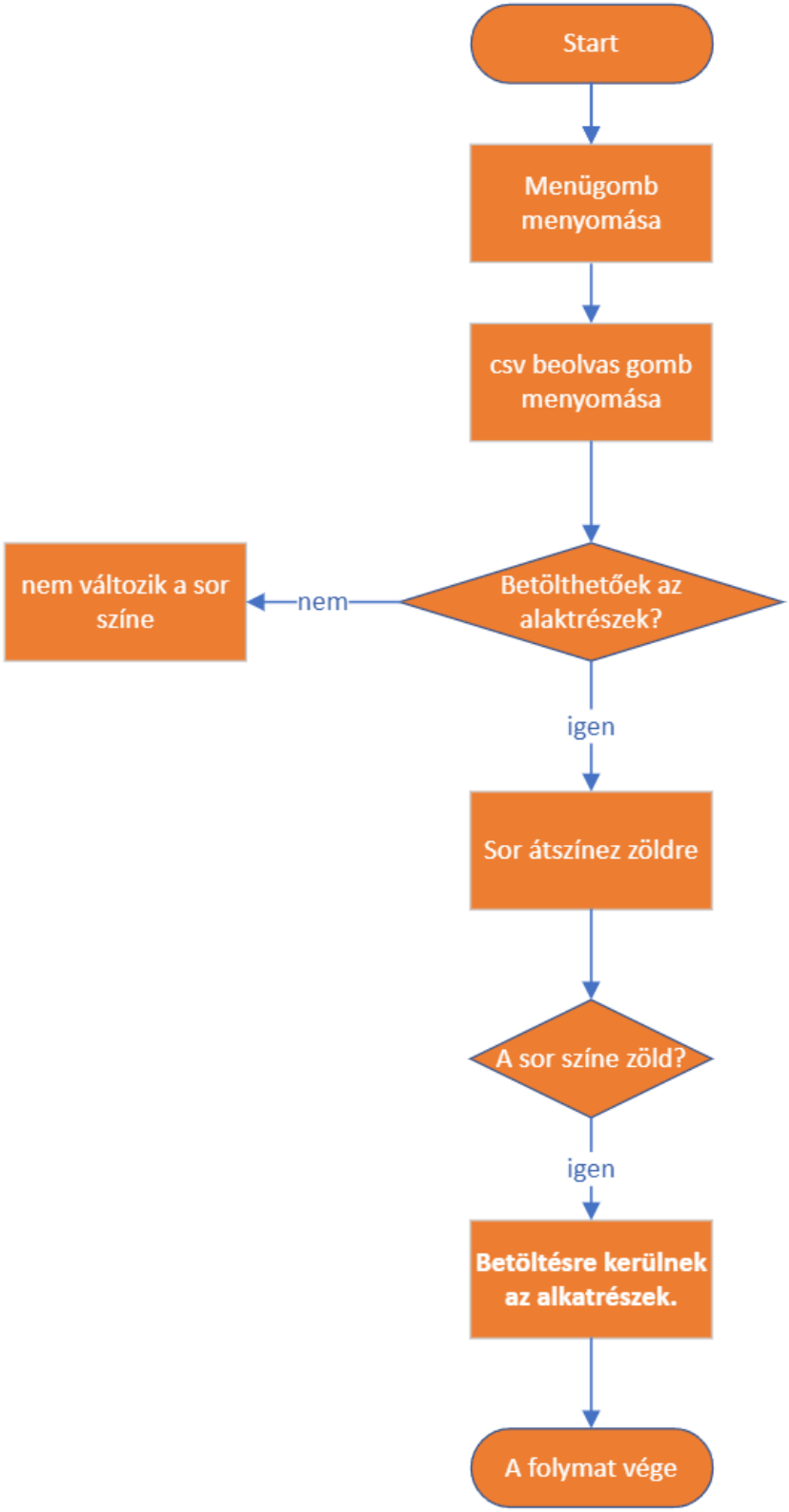
Az nc kód futtatása után az elkészült tábla tartalmazza az alkatrészeket és a selejteket. Feltételezve, hogy kevesebb a selejt, mint a jó alkatrész, így elegendő csak a kiválasztott alkatrészhez lementeni a selejt darabszámát. Ha készen vagyok a selejtek darabszámának összegzésével lementem az adatokat. A maradék táblát, hozzá adom a raktározási adatbázishoz. Ha már létezik, akkor a darabszámot megnövelem egyel. A futtatás után szintúgy a raktározási adatbázisból a felhasznált tábla felhasználható és lefoglalt darabszámát csökkentem egyel, egyel.

- Ötödik lépés:

Selejt alkatrészek újbóli kiosztása.

Miután a szerszámgépen lefutott az összes aktuális alkatrészt tartalmazó tábla, újbóli kiosztás történik a szoftverben automatikusan.

3.2 Betöltési folyamat

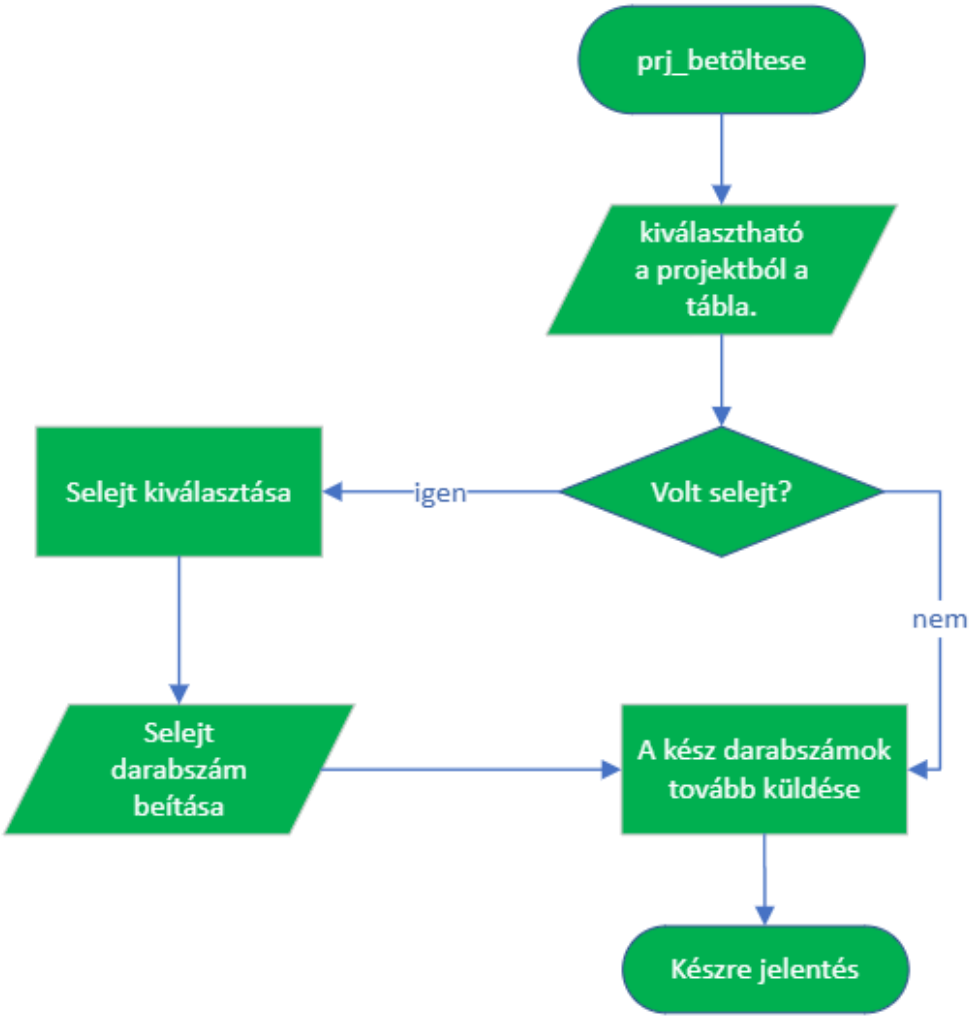


5. ábra:
Betöltési folyamatábra

A betöltési folyamatábra a betöltő ablak működését ábrázolja.

- Az ablak megnyitásához rá kell kattintani a felső menüsorra.
- Az ablak megnyitása utána az erre a célra kialakított CSV-t be kell tallózni. Az alkatrészeket egy erre a célra kialakított CSV file tartalmazza, ami a szükséges darabszámot, anyag megnevezést és a DXF elérési útvonalát.
- A beolvasás a gomb megnyomásával folytatódik. Ezek alapján automatikusan betöltésre kerülhetnek az alkatrészek a projektbe, ha a raktáron létezik, hozzá megfelelő anyagú megmunkálható tábla.
- Betöltés ellenőrzése. Ezt a sor zöld színre váltásával jeleníti meg az ablak. Ha nem vált színt akkor nem kerül betöltésre sem mert nincs hozzá anyag. Ha a sor színe zöldre változik akkor betöltődik a radan projektbe az alkatrész.
- Betöltődnek az alkatrészek. Automatikusan ki lehet őket osztani.

3.3 Lejelentő folyamat



6. ábra
Lejelentő folyamatábra

A lejelentő ablak célja, hogy amikor a CNC lézervágógépen lefutott a program akkor vissza lehessen adni a rendszernek a selejtek számát. Feltételezve, hogy a selejtekből kevesebb van ezért ezek darabszámát kell lejelenteni és nem a jó alkatrészeket.

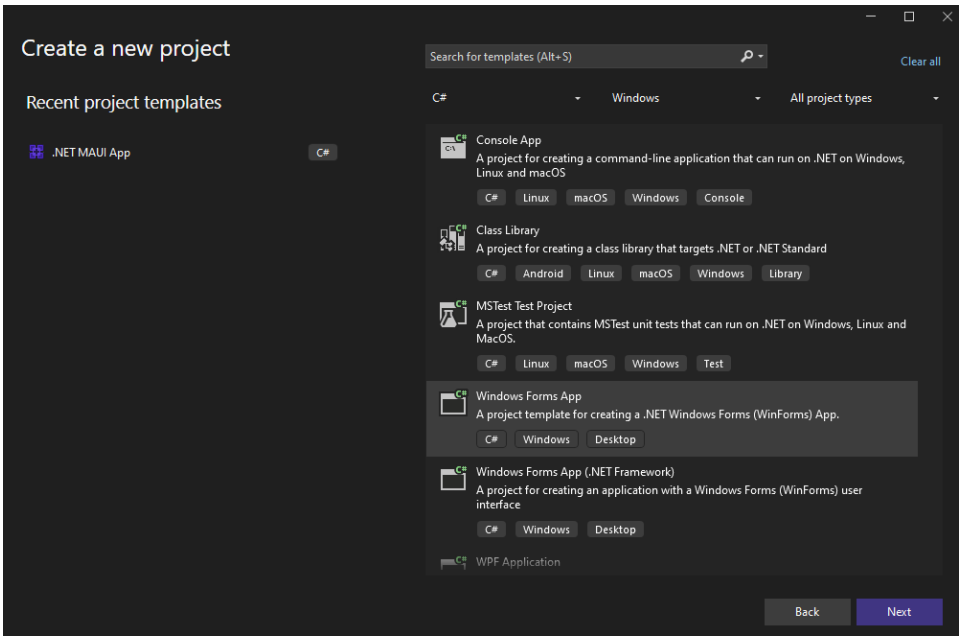
A folyamat a lézer gépnél lévő számítógépen vagy tableten lévő felület kezelésével kezdődik.

- A Projekt kiválasztásam, hogy hol szerepel a futtatott tábla.
- Futtatott tábla kiválasztása a projektből.
- Selejt alkatrészek kiválasztása a táblából.
- Ha volt selejt akkor a hibás darabszámot be kell írni.
- A kész darabok tovább küldése a rendszernek.

Munkámmal azt szeretném igazolni, hogy hatékony alapanyag felhasználással és tudatos nyomon követhető raktározással az alapanyag hulladék csökkenthető.

4 Szoftver testreszabás bemutatása

A szoftver testreszabást a fejlesztői környezetben egy üres projekt indításával kezdem. Ebben a projektben létrehozok a létező sablon közül kiválasztom a Windows Forms App-ot. Ennek segítségével már előre definiált ablakot kapok, amit az igényeimnek megfelelően képes vagyok módosítani.

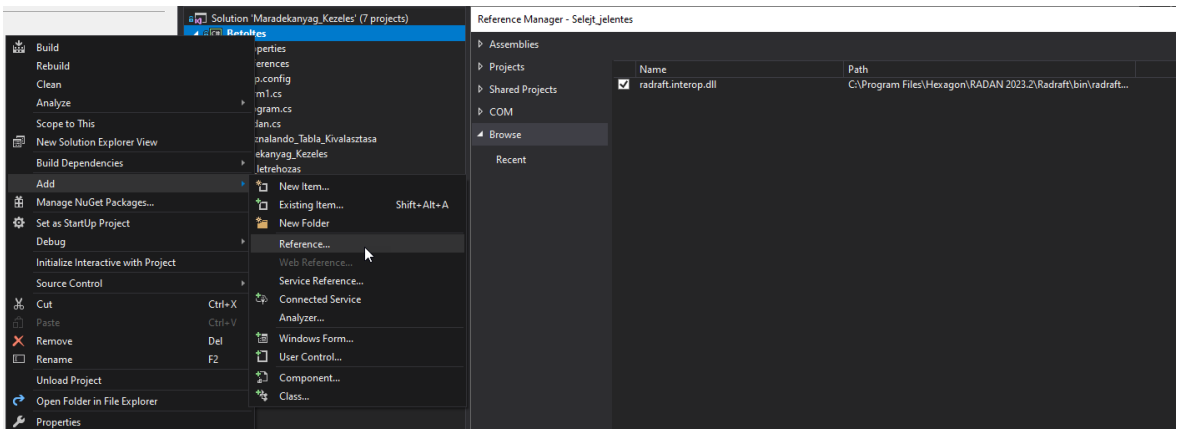


7. ábra Üres Windos Form App létrehozása

4.1 Referencia hozzáadása és Projekt alaphelyzetbe állítása

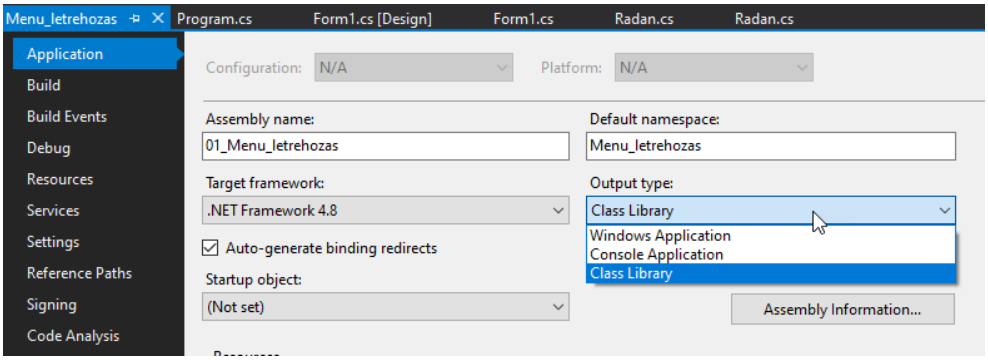
A RADAN API használatát a súgó segítségével lehet alkalmazni. Vannak példa kódok és már felhasználható megoldások is.

Ahhoz, hogy elérhetőek legyenek számomra a RADAN API eszközei, hozzá kell adnom a Visual Studio Projektemhez, az úgy nevezett Solution-hoz a Radraft.Interop.dll-t amit Referencia hozzá adásával lehet megtenni.



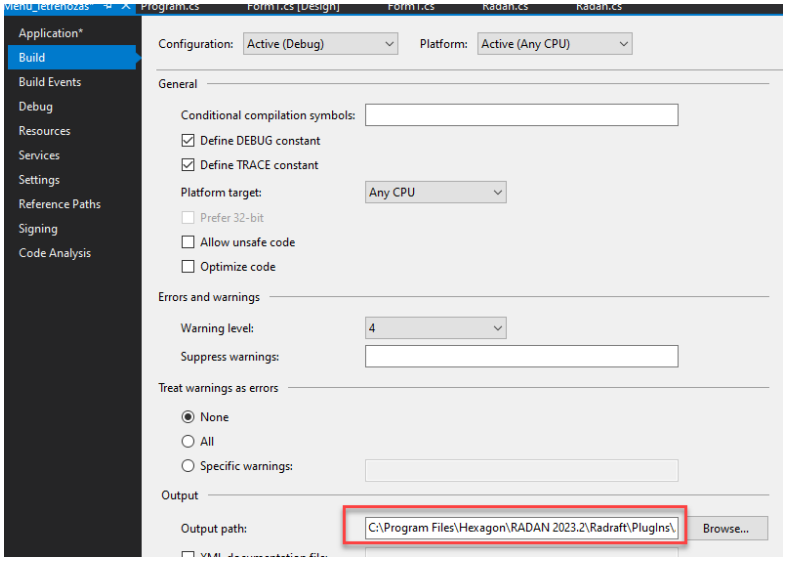
8. ábra
Radan API hozzáadása

A Visual Studio segítségével létre tudok hozni indítható .exe alkalmazásokat. Számomra viszont .dll formában kell a program hisz majd később ezt fogja olvasni a szoftver a megfelelő helyre való beillesztéssel. Egy ilyen dll-t létre hozásához be kell állítanom az adott Projekt kimeneti típusát „Class Libary”-ra. Így az eredmény .dll kiterjesztésű lesz.



9. ábra
dll készítés

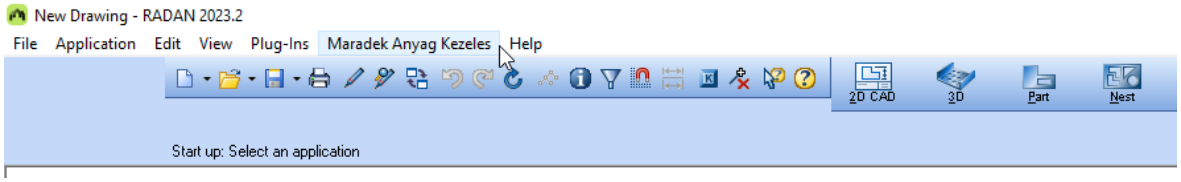
Ahhoz, hogy a megfelelőhelyre küldjem ki a már említett .dll-t be kell állítanom a telepített RADAN mappáján belül a PlugIns\AutoLoad\ mappa szerkezetbe kimeneti útvonalat.



10. ábra
kihelyezési útvonal

4.2 Menüszalag testre szabás

A szoftvertestre szabás egy indító menü gomb létrehozásával kezdődik a fenti menüsoron.



11. ábra
Módosított menüszalag

Erre a lépésre azért van szükség mert a többi programot is ebbe a menübe teszem bele a rendezettség miatt. Ha ezt a felső menü gomb létrehozást minden programban megcsinálnám akkor képes lenne hibásan működni amikor több testre szabási file is hivatkozik erre a főmenüre. Így viszont a fix főmenübe teszem bele az almenüket.

4.2.1 Menü létrehozás

Látható, hogy a Radraft.Interop névtér hozzá lett adva a kódhoz így elérhetőek a szoftverfejlesztők által írt API eszközök.

Ebben a kódban azt lehet észre venni, hogy az OnConnectToApplication() metódussal a „m_app” segítségével érem majd el az aktuálisan futó RADAN ablak vezérlését.

Az OnUpdateGUI() metódus akkor kerül meghívásra amikor a kezelő felület frissül. Az egy soros kódban pedig a már említett aktuális RADAN ablakomban a PluginManager segítségével hozzá adok egy menüt a felső menüsorban. Ezt a
üres string változó a „” jelzés jelöli, hogy nem kötöm semmilyen más menühöz. A következőkben már az idéző jelek közé a „Maradek Anyag Kezeles” kerül így ehhez a menühöz kerülnek az almenük.

```
using Radraft.Interop;
namespace Menu_letrehozas{
  public class RadraftInteropPlugIn{
    private Radraft.Interop.Application m_app = null;
    public void OnConnectToApplication(Radraft.Interop.Application app){
      m_app = app;
    }
    public void OnUpdateGUI(){
      m_app.PluginManager.AddMenu("", "Maradek Anyag Kezeles");
    }
  }
}}
```

4.3 Betöltés vizsgálata

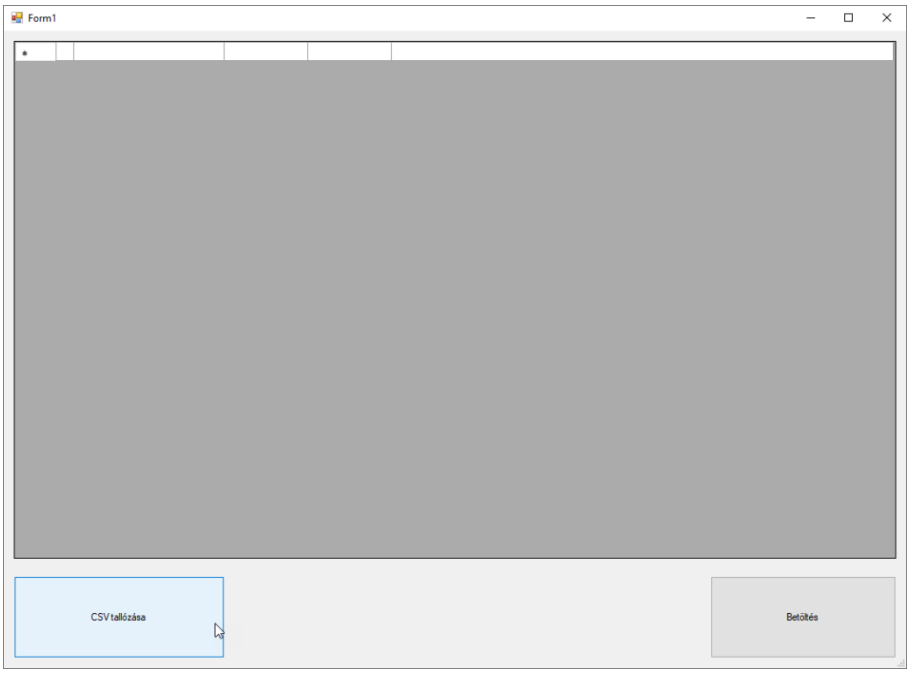
Az alkatrészek feltöltése egy Form segítségével hozom létre. Ez azért könnyíti meg a programozást mert így a .NET keretrendszernek köszönhetően egy már kész aktív ablakot tudok létrehozni, amit egyszerű eszközökkel képes vagyok módosítani. A betöltéshez számomra az alkatrészek elérési útvonalára, darabszámára, lemez vastagságára lesz szükség. Viszont egy ellenőrzést kell végezni a betöltés előtt, hogy a raktárban létezik-e ilyen anyag megnevezésű és lemezvastagságú alapanyag a legyártáshoz. Miután megtörténik a vizsgálat kezdődhet a betöltés.

```
public void OnUpdateGUI(){
  m_app.PluginManager.AddMenuItem("Maradek Anyag Kezeles", "Projekt Alkatrészek
Betöltése", "Load");
  }
private void Load(){
  Form1 Form_1 = new Form1();
  Form_1.ShowDialog();
  storage_DB_Loader();
  Sheet_Load();
  Parts_load_from_csv(Form1.rowsAndColumns);
  m_app.Mac.lay_run_nest(0);
  m_app.Mac.prj_save();
  MessageBox.Show("Az alkatrészek kiosztásra kerültek.");
  }
```

Látható, hogy már az említett módon megtörtént a menü hozzá adás. Ha a jelenleg lenyíló menüben kiválasszuk a „Projekt Alkatrészek Betöltése” menügombot akkor a „Load” nevű metódus hajtóik végre.

A „Load” metódusban elsőnek példányosítom a Form ablakom. Majd a .ShowDialog() metódussal megjelenítem. Amikor bezárul a betöltő ablak akkor folytatódik tovább a már leválogatott információkkal az alkatrészek és a lemezalkatrészek betöltése.

4.4 Betöltő ablak



12. ábra
Betöltő ablak

A folyamatot az alkatrészek automatikus betöltésével kezdem. Visual Studio-ban készítettem, egy C# Form projektet. Ezt a Projektet elneveztem „betöltésnek”. Mivel ez egy fejlesztői környezet így rendelkezésünkre állnak már elölre megírt kódok. Ilyen például maga a „Form” is. A Form egy üres ablak jelenleg. Ide az eszköztárból betudom húzni egy gombot,

aminek a kód sora akkor lép életbe, ha lenyomjuk. Mondhatni kapcsolóként viselkedik. A fejlesztői környezetben dupla kattintással a gombon megnyílik a mögötte lévő kód.

Látható, hogy ez egy „Event” ami igazolja, hogy csak akkor lép működésbe, hogy ha a gomb lenyomásra került.

```
private void button3_Click(object sender, EventArgs e){
  Csv_Selection();
  Csv_Reading();
  Part_Dir_Listing(dataGridView3);
  Storage_DB_Loading();
  ColorChange();
  }
```

Mivel a C# nyelveben lehet függvényeket használni, így a könnyebb olvashatóságért a folyamatot rész folyamatokra szettem szét.

A `Csv_Selection()` kódrészlet felelős a CSV kiválasztásának útvonaláért. A fájl tallózását beépített API segítségével oldottam meg. Ez is System.IO Névtér része.

```
public void Csv_Selection(){
  OpenFileDialog openFileDialog = new OpenFileDialog();
  DialogResult result = openFileDialog.ShowDialog();
  if (result == DialogResult.OK){
    string filePath = openFileDialog.FileName;
    csv_file = filePath;
    }
  }
```

A `Csv_Reading()` a már kiválasztott útvonalbeolvasása Eddigre már a fájl útvonala meg van ezért StreamReader segítségével be kell olvasni.

A beolvasás soronként történik. Minden sort egy globális string listába olvastatok be így. Látható hogy egy egyszerű while ciklust használok ami a következő feltételig olvasas: (!reader.EndOfStream).

A felkiáltó jel a tagadást jelenti, tehát ha nem az utolsó sorban vagyunk akkor folytatódik tovább a ciklus. A következő sort a „`string line = reader.ReadLine();`” hívja meg.

Látható ezen felül, hogy a rowsAndColumns.Clear() segítségével a már esetleg feltöltött lisát kiürítem. Ez azért lehet fontos, mert ha több egy másik CSV-t szeretnék betölteni, akkor mindkét CSV adata benne szerepelne. Így képes lenne több CSV -t is olvasni viszont számunkra, hogy ne keveredjenek a projektek így csak egy adathalmazt szabad használnunk.

```
public void Csv_Reading(){
  string csvFilePath = csv_file;
  using (StreamReader reader = new StreamReader(csvFilePath)){
    while (!reader.EndOfStream){
      string line = reader.ReadLine();
      string[] columns = line.Split(';');
      List<string> row = new List<string>(columns);
      if (columns[0] != "alk"){
        rowsAndColumns.Add(row);
        }
      }
    }
  }
```

Part_Dir_Listing. A Form-hoz hozzá adtam egy DataGridView eszközt, amivel különböző adatokat lehet megjeleníteni. Azért nem csak egy szöveges adat kell használni mert a DataGridView használhatóval több megjelenítési eszközt is elérünk. Mint például a cellánkként színezést vagy a sor megjelenítését/elrejtését. Az utóbbi esetében az adat sort nem kell kitörölni csak a láthatóságát állítjuk át, így nincs adatvesztés sem. Ez a függvény gyakorlatilag a már feltöltött listánk adatait, hozzáadja a DataGridView oszlopaihoz és celláihoz.

```
public void Part_Dir_Listing(DataGridView dataGridView){
  for (int i = dataGridView3.Rows.Count-1; i < rowsAndColumns.Count; i++){
    dataGridView.Rows.Add();
    for (int j = 0; j < rowsAndColumns[i].Count; j++){
      dataGridView.Rows[i].Cells[j].Value = (rowsAndColumns[i][j]);
      }
    }
  }
```

Storage_DB_Loading. Azon felül, hogy sikerült feltölteni a DataGridView-t ellenőrzést kell tenni, hogy a lemezraktárunkba szerepelnek-e ezen anyagok. A beolvasás itt is StreamReaderrel történik. A feltöltés pedig szintúgy egy globális lista, aminek a neve Storage_DB. Abban az esetben, ha a raktári adatbázisban az adott tábla paramétereinél a darabszám oszlopban „0” érték szerepel akkor nem történik betöltésre a Storage_DB Listába. Itt is a folyamat a Storage_DB lista törlésével kezdődik a már említett okok miatt.

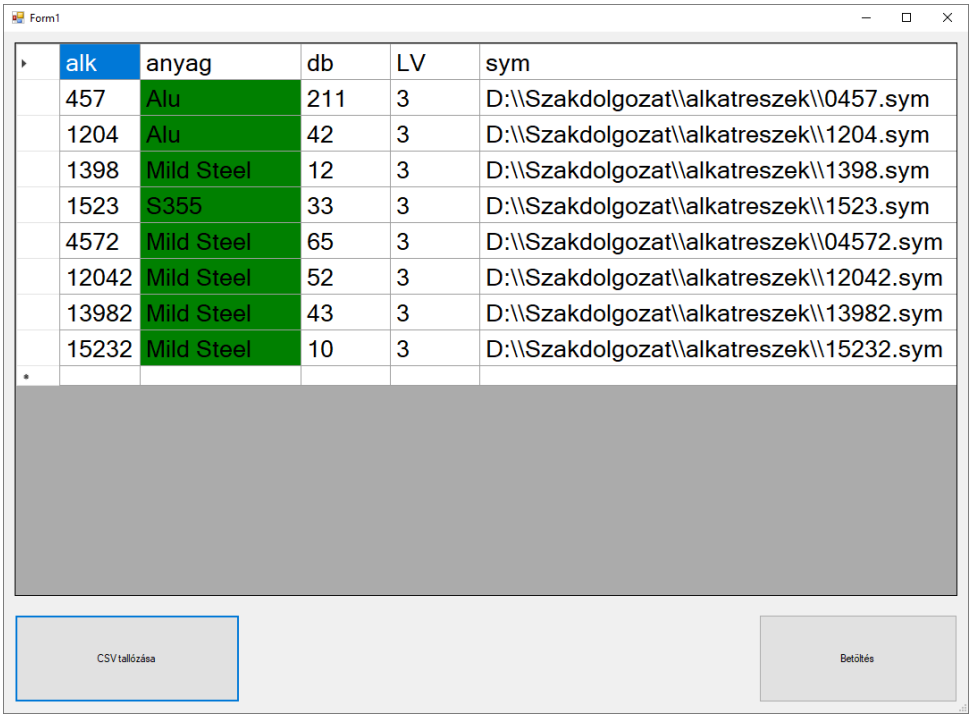
```
private void Storage_DB_Loading(){
  string filePath = MaterialListPath;
  storage_DB.Clear();
  using (StreamReader sr = new StreamReader(filePath)){
    while (!sr.EndOfStream){
      string line = sr.ReadLine();
      string[] values = line.Split(';');
      if (!(values[values.Length - 2] == "0")){
        storage_DB.Add(new List<string>(values));
      }
    }
  }
```

ColorChange. Ez a metódus az alkatrészek átszínezésért felelős. Minden DataGridView során át fut a Storage_DB lista összes tagján. Ellenőrzi, hogy az Storage_DB sor második tagja meg egyezik-e a DataGridView adott sorának második tagjával. Az az létezik-e olyan anyag megnevezés a raktári táblák közül, ami egyezik a betöltendő alkatrész anyag megnevezésével. Az „&” jel a logikai „és”-re utal. Ami ez után következik az a lemez vastagságot is beleveszi az ellenőrzésbe. Hisz számomra csak akkor lesz betölthető az alkatrész, ha van olyan táblánk a raktárba, aminek az anyag megnevezése és a lemezvastagsága is egyezi. Ha van egyezés akkor át színezi az egyik cellát zöldre. Ez lesz később a betöltés feltétele.

```
private void ColorChange(){
  for (int i = 1; i < dataGridView3.RowCount - 2; i++){
    for (int j = 0; j < storage_DB.Count; j++){
      if ((dataGridView3.Rows[i].Cells[1].Value.ToString() == storage_DB[j][1])
& (dataGridView3.Rows[i].Cells[3].Value.ToString() == storage_DB[j][2])){
        if           (dataGridView3.Rows[i].Cells[1].Style.BackColor          !=
System.Drawing.Color.Green){
          dataGridView3.Rows[i].Cells[1].Style.BackColor                      =
System.Drawing.Color.Green;
          }
        }
      }
    }
  }
```

4.4.1 Ablak használata

Miután megtörtént a betöltés az átszínezés jelzi, hogy mit tudunk betölteni majd a RADAN-ba. Ha esetleg van olyan alkatrész, amihez nincs anyag, azt itt még ki tudjuk törölni a DataGridView adatai között. A „Bezárás” gombra kattintva egyszerűen csak bezárul az ablak és folytatódik a betöltő program most már a megfelelő alkatrészekkel és táblákkal.



alk	anyag	db	LV	sym
457	Alu	211	3	D:\\Szakdolgozat\\alkatreszek\\0457.sym
1204	Alu	42	3	D:\\Szakdolgozat\\alkatreszek\\1204.sym
1398	Mild Steel	12	3	D:\\Szakdolgozat\\alkatreszek\\1398.sym
1523	S355	33	3	D:\\Szakdolgozat\\alkatreszek\\1523.sym
4572	Mild Steel	65	3	D:\\Szakdolgozat\\alkatreszek\\04572.sym
12042	Mild Steel	52	3	D:\\Szakdolgozat\\alkatreszek\\12042.sym
13982	Mild Steel	43	3	D:\\Szakdolgozat\\alkatreszek\\13982.sym
15232	Mild Steel	10	3	D:\\Szakdolgozat\\alkatreszek\\15232.sym

13. ábra
Betöltő ablak működés közben

4.5 Táblák betöltése a RADAN-ba

Storage_DB_Loader. Az ablak bezárása után Storage_DB_Loader () metódus segítségével átadjuk az adatokat a Storage_DataBas nevű lét dimenziós tömbnek, így elérve azt, hogy a raktári táblákat betudjam tölteni a RADAN-ba.

```
private void storage_DB_Loader(){
  storage_DataBase.Clear();
  string filePath = MaterialListPath;
  using (StreamReader sr = new StreamReader(filePath)){
    while (!sr.EndOfStream){
      string line = sr.ReadLine();
      string[] values = line.Split(';');
      storage_DataBase.Add(new List<string>(values));
      }
    }
  }
```

Sheet_Load() A Sheet_Load. A metódusban látható, hogy végig iterálok az egész listán és minden elemet egy vizsgálattal kezdek. Ez a vizsgálat felelős azért, hogy ha a raktárban „0” darab van jelenleg soron következő táblából akkor azt ne töltse be a felhasználandó táblák közé. A felhasználandó táblák adatait m_app.Mac parancsokkal kell feltölteni. Látható, befoglaló méret, anyag vastagság, anyag megnevezés és egy Stock_ID. Az utóbbi a tábla egyedi azonosítója. A „-2” arra utal, hogy a betöltendő sor, oszlopainak összegéből kiátkozunk az utolsó előtt kettővel lévőre. Ez az oszlop tartalmazza a darabszámot az adott raktári táblán.

```
public void Sheet_Load(){
  for (int i = 0; i < storage_DataBase.Count; i++){
    if (storage_DataBase[i][storage_DataBase[i].Count - 2] != "0"){
      m_app.Mac.PRJ_SHEET_MATERIAL = storage_DataBase[i][1];
      m_app.Mac.PRJ_SHEET_THICKNESS = double.Parse(storage_DataBase[i][2]);
      m_app.Mac.PRJ_SHEET_NUM_AVAILABLE                                    =
int.Parse(storage_DataBase[i][storage_DataBase[i].Count - 2]);
      double x = double.Parse(storage_DataBase[i][3].Replace(".", ","));
      double y = double.Parse(storage_DataBase[i][4].Replace(".", ","));
      m_app.Mac.PRJ_SHEET_X_SIZE = x;
      m_app.Mac.PRJ_SHEET_Y_SIZE = y;
      m_app.Mac.PRJ_SHEET_STOCK_ID = storage_DataBase[i][0];
      m_app.Mac.prj_add_sheet();
    }
  }
```

4.6 Alkatrészek betöltése és kiosztása a RADAN-ba

Az előzőhöz hasonlóan mivel már rendelkezésemre áll az adathalmaz így már csak végig kell iterálnom a megfelelő adatokat tartalmazó Listán.

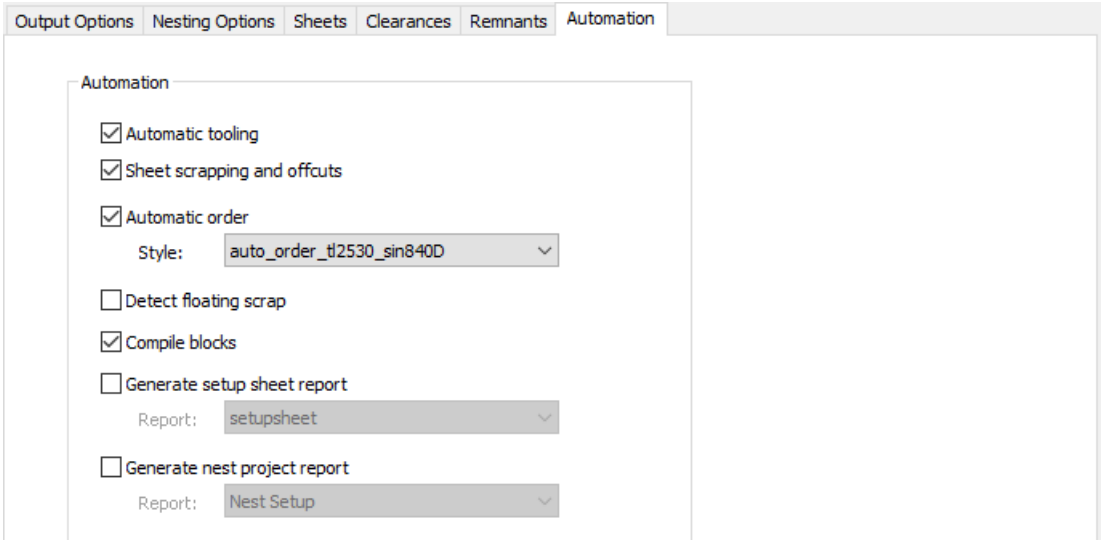
A `Parts_load_from_csv` hasonlóan a Táblák betöltésénél itt is m_app.Mac parancsok állnak rendelkezésünkre a megfelelő változók feltöltésére. Ha meg van az elégséges változók adatainak kitöltése a `m_app.Mac.prj_add_part()` metódus segítségével töltöm be egyesével az alkatrészeket. Ezt követően egy újabb ellenőrzés történik a táblákra vonatkozóan. Ez az ellenőrzés egyszer már felfutott viszont abban esetben, ha több CSV-t kell betölteni könnyen össze kimaradhatnak táblák.

Az ellenőrzés minden alkatrész után lefut és ellenőrzi, hogy töltöttünk-e már fel hozzá táblát. Hanem akkor itt most újra történik.

```
public void Parts_load_from_csv(List<List<string>> rowsAndColumns){
  List<string> loaded_Sheet_Material = new List<string>();
  for (int i = 1; i < rowsAndColumns.Count; i++){
    m_app.Mac.PRJ_PART_MATERIAL = rowsAndColumns[i][1];
    m_app.Mac.PRJ_PART_NUMBER_REQUIRED = int.Parse(rowsAndColumns[i][2]);
    m_app.Mac.PRJ_PART_THICKNESS = int.Parse(rowsAndColumns[i][3]);
    m_app.Mac.PRJ_PART_FILENAME = rowsAndColumns[i][4];
    m_app.Mac.prj_add_part();
    for (int j = 0; j < storage_DataBase.Count; j++){
      if (!(storage_DataBase[j][storage_DataBase[0].Count - 1] == "0") &&
        (m_app.Mac.PRJ_PART_MATERIAL == storage_DataBase[j][1] &&
         !(loaded_Sheet_Material.Contains(m_app.Mac.PRJ_PART_MATERIAL)))){
         m_app.Mac.PRJ_SHEET_MATERIAL = storage_DataBase[j][1];
         m_app.Mac.PRJ_SHEET_THICKNESS = double.Parse(storage_DataBase[j][2]);
        m_app.Mac.PRJ_SHEET_NUM_AVAILABLE                                    =
int.Parse(storage_DataBase[j][storage_DataBase[0].Count - 1]);
        double x = double.Parse(storage_DataBase[j][3].Replace(".", ","));
        double y = double.Parse(storage_DataBase[j][4].Replace(".", ","));
        m_app.Mac.PRJ_SHEET_X_SIZE = x;
        m_app.Mac.PRJ_SHEET_Y_SIZE = y;
        m_app.Mac.PRJ_SHEET_STOCK_ID = storage_DataBase[j][0];
        m_app.Mac.prj_add_sheet();
      }
    }
    if (!loaded_Sheet_Material.Contains(rowsAndColumns[i][1])){
      loaded_Sheet_Material.Add(rowsAndColumns[i][1]);
    }
  }
```

4.6.1 Kiosztás, Szerszámozás, Sorrendtervezés és Post Processzálás

A főbb folyamatokat már elő készítettük így kiosztásra csak egyetlen sor hivatkozik. A „m_app.Mac.lay_run_nest(0)” a már feltöltött alkatrészeket feltölti a feltöltött táblákra, automatikusan szerszámozza, végrehajtja a táblalevágást, sorrendtervezi a megmunkálást és elkészíti a NC kódot minden kiosztott táblákra.

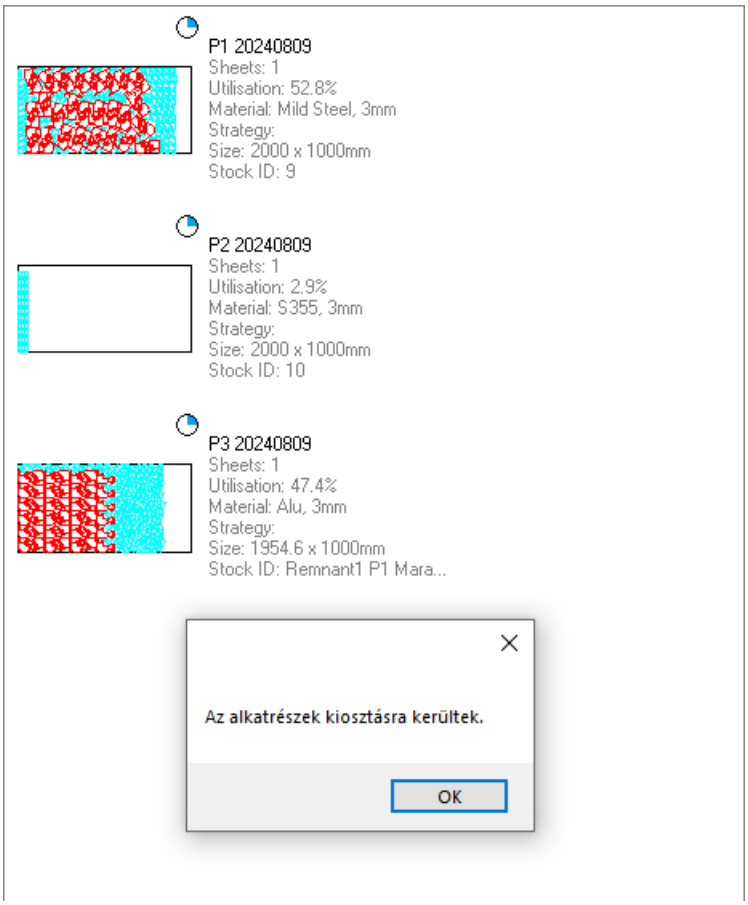


14. ábra
Automatizálás RADAN-ban

Ezt követően már csak egy automatikus mentésre van szükségünk, hogy minden képen frissüljön a projekt file. A későbbi felhasználás során a Projekt file adatai és táblák adatait fogjuk olvasni így azoknak minden módosítás után lementéssel frissíteni kell.

```
private void Load(){
  Form1 Form_1 = new Form1();
  Form_1.ShowDialog();
  storage_DB_Loader();
  Sheet_Load();
  Parts_load_from_csv(Form1.rowsAndColumns);
  m_app.Mac.lay_run_nest(0);
  m_app.Mac.prj_save();
  MessageBox.Show("Az alkatrészek kiosztásra kerültek.");
  }
```

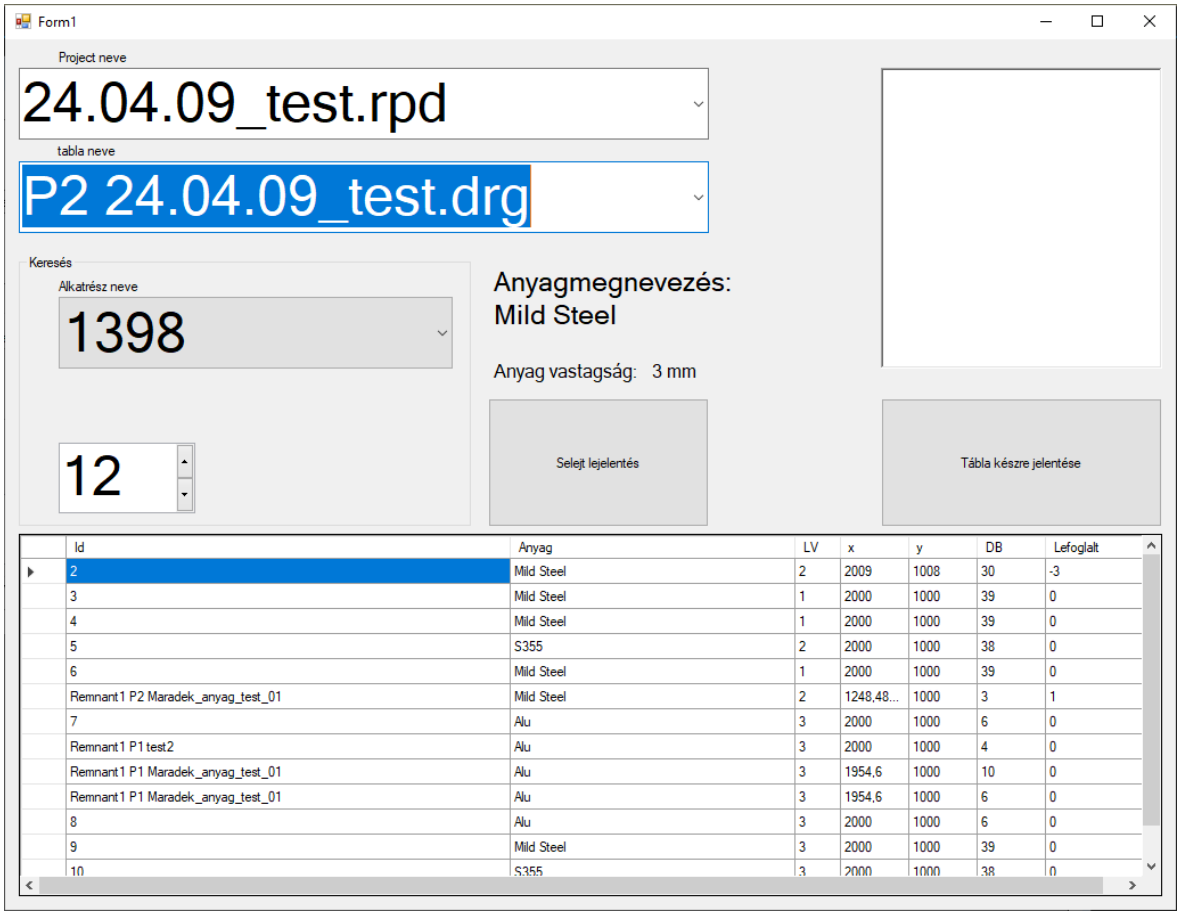
4.6.2 Betöltés vége



15. ábra
a folyamat vége

4.7 Lejelentő ablak

4.7.1 Áttekintés

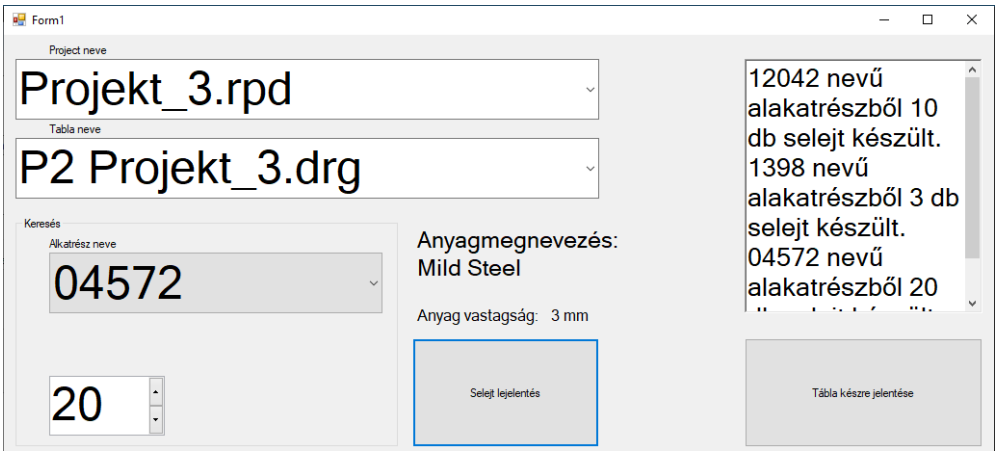


16. ábra
Lejelentő ablak működése

Ez az ablak selejtek lejelentésére szolgál. A CNC program futtatása után az alkatrészeket le kell jelentenünk, hogy mik készültek el és melyik alkatrészek lettek selejtesek. Az ablak a projekt mappa szerkezetét és fájljait olvassa, hogy pontosan meg tudjuk határozni a táblán szereplő alkatrészeket. Miután kiválasztottuk a kezelő felületen az alkatrészeket és beírtuk a megfelelő darabszámot az ablak az alapanyag raktár adatbázisával kommunikálva csökkenti az alapanyag darabszámot és elkészíti a selejt CSV-t amit majd a következő projekthez hozzá lehet főzni és újra kiosztani.

Az ablakban látható, hogy különböző lenyíló felületeket tartalmaz, amiből ki lehet választani pontosam melyik tábláról van is szó. Az adatokat a projekt kiválasztásával kezdjük és felülről lefelé haladva következőnek a táblát válasszuk ki, majd a táblán szereplő alkatrészt. Minden tábla kiválasztásnál az anyag megnevezés és anyag vastagság is frissül, ha esetleg nem változik az azért van mert az előző kiválasztott tábla ezen paraméterei megegyeznek vele.

Feltételezve, hogy a táblán sokkal kevesebb a selejt alkatrész ezért elég csak kiválasztani a selejt alkatrész nevét és beírni, hogy hány darab selejt készül. Alap esetben nulla helyett az összeset olvastatom ki az ablakkal, hogy egyszerűbb legyen a lejelentés, ha esetleg valóban mind selejt lenne, ha véletlen rossz technológiai adattal helytelen kompenzációs értékkel és ezért rossz geometriai méretekkel készült volna az egész tábla.



17. ábra
Az ablak felső része

A selejteket a szöveg dobozban listázzuk ki így a lejelentés előtt tisztán látszik, hogy miből hány darabot jelentettünk le eddig. Ameddig nem nyomok rá a „Tábla készre jelentése” gombra addig gyűjtöm az információkat a selejtek nevéről és darabszámáról. Fontos, hogy a lejelentés előtt ki válasszam a lenti felsorolásból, hogy melyik táblát használom fel.

Miután meg vannak a megfelelő adataim csak el kell indítanom a lejelentési folyamatot a gomb segítségével. A folyamatban már eddigre rendelkezésre álló adatokból elkészítek egy CSV-t ami tartalmazza ezeket az adatokat és csökkentem a raktári adatbázisban a felhasznált tábla darabszámát. Ha rendelkezésre áll a tábla megmunkálása után maradék táblánk akkor azt hozzáadom az anyag adatbázishoz. Abban az esetben, ha ez már létezik akkor csak hozzá adok a darabszámához egyet. Ellenkező esetben csak létre is hozom új elemként a raktári anyaglistába.

4.8 Az ablak mögötti forráskód

4.8.1 Változók

```
  public List<List<string>> actualSheetDB = new List<List<string>>();
  public List<List<string>> warehouseTables = new List<List<string>>();
  public static string tabelfile;
  public static string projectfile;
  public static string remnantfile;
  public List<string> parts = new List<string>();
  public List<List<List<string>>> projectsList = new List<List<List<string>>>();
  public List<List<string >> selejtek = new List<List<string>>();
  public static string MaterialListPath = @"D:\Szakdolgozat\DB_2.csv";
```

A háttérben különböző listákban vannak tárolva az adatok. Mivel legördülő adatsorok tárolják a kiválasztott projektet és tábla nevet ezért külön változókban tárolom ezeket, hogy amikor elindítom a lementési folyamatot akkor ne kelljen lekérdezni ezeket az adatokat csak hivatkozni rájuk. `List<List<string>>` egy kétdimenziós megoldás az adatok tárolására. Azért praktikus számomra ezekben tárolnom az adatokat mert nem lehet előre tudni az ablak megnyitásának pillanatában pontosan hány darab projektel vagy táblával kell majd dolgozni ezért olyan adat szerkezetre volt szükségem, amit korlátlanul tudok bővíteni.

4.8.2 Ablak betöltése

```
private void Form1_Load(object sender, EventArgs e){
  LoadingTheWindow();
}
public void LoadingTheWindow(){
  warehouseTables.Clear();
  Reading_warehouseTables();
  Uploading_Datagrid();
  LoadingTheDitailsToProjectsList();
  ReadingTheProjectsList();
}
```

Az ablak betöltését praktikusan külön metódusba szedtem mert amikor vége a tábla lejelentési folyamatnak akkor majd szintúgy megtudom hívni, mint most az ablak első betöltésénél.

A folyamat olvashatóságáért próbáltam leíró metódus megnevezéseket alkalmazni, amivel a könnyebb megértés volt a célom. A raktár készletet bár első indításnál üres Lista szerkezettel indítjuk de ha ilyenkor alkalmazóm rá a `.Clear()`metódust akkor voltaképpen nem vesztek adatot de ha a folyamat végén hívom meg újra ezt az ablak betöltést akkor a már feltöltött adatbázis duplikálódnál. Ezért célszerű alaphelyzetbe állítani a feltöltés előtt.

Miután feltöltöttem a raktári adatbázist már van elég adatom, hogy feltöltsem a Datagrid-et ami majd a felhasználható táblákat fogja jelölni.

A `LoadingTheDitailsToProjectsList()`metódus feladata nagyon hasonlít a `warehouseTables_Reading()` metódusra. Mind a kettő Listákat tölt fel, viszont az előbbi a nevéből is érhetően a `ProjectsList`-et ölti fel adatokka. Erre azért van szükség, hogy a `ReadingTheProjectsList()` metódus a legördülő mezőket feltudja tölteni adattal.

4.8.3 Legördülő mezők

A legördülő mezők, mint a gombok és más eszközök is az eszköz tárból kiválasztható, így már előre definiáltak. A legördülő mezőket „comboBox”-nak nevezik az eszköztárban. Terjedelmi okok miatt csak a Tábla legördülő mezőjét mutatom be, de minden legördülő mező saját kóddal rendelkezik, amik a saját feladatukat látják el. Például a Projekt legördülő mező a lehetséges táblákat állítja be, de ekkor még képtelen beállítani az anyagot és anyagvastagságot mert még nincs kiválasztva tábla. Alapértelmezetten minden esetben a nulladik projekt vagy nulladik tábla vagy nulladik alkatrész lesz kiválasztva.

4.8.4 Tábla legördülő mező

```
private void comboBox4_SelectedIndexChanged(object sender, EventArgs e){
  actualSheetDB.Clear();
  tabelfile = projectsList[comboBox3.SelectedIndex][1][comboBox4.SelectedIndex];
  projectfile = projectsList[comboBox3.SelectedIndex][0][0];
  SearchRemnant();
  Loading_actualSheetDB();
  CB_Upload(actualSheetDB, comboBox1);
  Material_Load();
  actualSheetDB.Clear();
  defectives.Clear();
  richTextBox1.Text = "";
}
```

Látható, hogy a folyamat akkor fut le amikor egy másik elemet választok ki a listából. Itt is duplikáció elkerülése érdeképen az adatok tisztításival kezdődik a folyamat. Beállítom a kiválasztás után a `tabelfile` változót. Majd ezt követően a `projectfile` is módosításra került. `SearchRemnant()`metódus a `remnantfile` ért felelős. Amikor majd lefut a maréktábla kimentés egyszerűbb lesz csak azt ellenőrizni, hogy üres-e. Hanem üres akkor a táblának létezik maradék táblája.

`Loading_actualSheetDB()` metódusban a nevéből érthetően a tábla adatai feldolgozásával foglalkozom. Megnyitom az tábla xml formátumát és ki olvasom belőle az anyagmegnevezést, anyagvastagságot, alkatrész nevet és darabszámát. A `CB_Upload(actualSheetDB, comboBox1)` feltölti a legördülő mezőt a már kiolvasott adatokkal így kiválasztható lesz az alkatrész a listából.

`Material_Load()` felelős az ablakban lévő anyagmegnevezés és anyagvastagság frissítésért. Amilyen tábla van kiválasztva olyan adatok fognak itt szerepleni.

Ezt követően már csak adatok tisztítása következik. Mivel a folyamat akkor fut le amikor táblát váltunk, így a selejteket tartalmazó adatokat és a szövegdoboz adatait alaphelyzetbe kell állítani.

4.8.5 Selejt lejelentő gomb

```
private void button2_Click(object sender, EventArgs e){
  richTextBox1.AppendText($"{comboBox1.Text}         nevű         alakatrészből
{numericUpDown1.Value} db selejt készült.");
  richTextBox1.AppendText("\n");
  List<string> row = new List<string>();
  row.Add(comboBox1.Text);
  row.Add(numericUpDown1.Value.ToString());
  defectives.Add(row);
}
```

A gomb megnyomása után a szövegmezőt töltöm fel az adatokkal, amik éppen az ablak kiválasztott adatai. Majd ezt követően létrehozok egy listát, amit feltöltöm a kiválasztott selejt alkatrész adataival. Miután kész az ideiglenes lista hozzá adom a `defectives` Listához így amikor a tábla lejelentésre kerül a sor, minden adattal rendelkezni fogok, hogy elkészítsem a táblához tartozó CSV-t, amit tartalmazza ezeket az adatokat.

4.8.6 Tábla lejelentő gomb

```
private void button1_Click_1(object sender, EventArgs e){
  Loading_actualSheetDB();
  Tabel_Data_Reading();
  Sym_Reading_From_Tabel();
  Waste_Quantity();
  RemnantsWrite();
  string sheetDB_csv = tabelfile.Remove(tabelfile.Length - 4) + ".csv";
  Tabel_Csv_Writeing(sheetDB_csv);
  TabelUsing();
  AddingRemnantTOUse();
  actualSheetDB.Clear();
  LoadingTheWindow();
}
```

A `Loading_actualSheetDB()` metódus a nevéből olvashatóan feltöltöm az alap adatok az aktuális táblához. Ezek az alapadatok az alkatrészneve anyagmegnevezése vastagsága és darabszáma. Ezeket az adatokat úgy értem, hogy xml formátumban megnyitom és a gyermek elemeken keresztül ki keresem az adatokat. Bár az anyagmegnevezés és a vastagság nem változik már a táblán mivel nem jelent nagy teljesítmény vesztést így azok az adatok is szerepelnek itt.

A `Tabel_Data_Reading()` metódusban az előzőekhez hasonlóan itt is xml dokumentum ként történik a tábla fálj megnyitása. Viszont itt a `actualSheetDB` Listához tábla fájl elérési útvonalát, azonosítóját futtatási számát és az aktuális futtatási számát adom hozzá. Miután megtörtént a kiolvasás az adott tábla faljában csökkentem a darabszámot, hogy majd a RADAN-ban vizsgálható legyen melyik táblák vannak már készen.

A `Sym_Reading_From_Tabel()` metódusban a projekt adataiból adom hozzá a `actualSheetDB` Lista megfelelő sorához az alkatrész file elérési útvonalát. A kiválasztott projket összes alkatrészén végig iterál és ha tartalmazza már a List akkor hozzáadja az elérési útvonalát.

A `Waste_Quantity()` nagyon hasonlít az előző metódushoz viszont itt a `defectives` Listát veszi alapul. Ha van selejt akkor azt itt adja hozzá. Ha nincs akkor „0” számot írbe, hogy ne maradjon üres oszlop a kiírandó CSV-ben.

A `RemnantsWrite()` hozzá adja a maradéktábla útvonalát az összes sorhoz akkor ha létezik. Ha nem létezik akkor a `remnantfile` egy üres string így nem üres oszlopot kapunk eredményül amikor elkészül a CSV.

A név megadás után a `Tabel_Data_Reading()`metódus a CSV elkészítésért felelős az `actualSheetDB` Listát soronként kiírja egy CSV fileba minek a neve a tábla neve. Az oszlopok sorrendje az alábbi kódban olvasható:

```
private void Tabel_Csv_Writeing(string filePath){
  using (StreamWriter writer = new StreamWriter(filePath)){
    string[] header = { "Alk_neve", "Alk_Dbszam", "Anyagmegnev" ,
    "LV", "Drg_Utvonala", "Tabla_Id", "Futtatasok_Szama",
    "Aktualis_Futtatasok_Szama", "Sym_Utvonal",
    "Selejt_Dbszam", "Maradek_Tabla" };
    writer.WriteLine(string.Join(";", header));
    for (int i = 0; i < actualSheetDB.Count; i++){
      writer.WriteLine(string.Join(";", actualSheetDB[i]));
    }
  }
}
```

	A	B	C	D	E	F	G	H	I	J	
1	Alk_neve	Alk_Dbsza	Anyagmeg	LV	Drg_Utvonala	Tabla_Id	Futtata	Aktual	Sym_Utvonal	Selejt	Maradek_Tabla
2	457	211	Alu	3	D:\Szakdolgozat\Projekt\24.04.09_test\nests\P1 24.04.09_test.drg	P1 24.04.09_test	1	2	D:\\Szakdolgozat\\alkatreszek\0457.sym	209	D:\Szakdolgozat\Projekt
3	1204	42	Alu	3	D:\Szakdolgozat\Projekt\24.04.09_test\nests\P1 24.04.09_test.drg	P1 24.04.09_test	1	2	D:\\Szakdolgozat\\alkatreszek\1204.sym	0	D:\Szakdolgozat\Projekt
4											

18. ábra

Kész CSV

Ezt követően `TabelUsing()` metódus csökkenti a datagridben kijelölt tábla darabszámát a warehouseTables Lista teljese újra írása előtt.

Nagyon hasonló a metódus is de itt természetesen a maradék tábláról van szó. Ha a maradéktábla már létezik akkor csak a darabszámát növel, hisz ez esetben keletkezik. Ha viszont még nem léteik akkor létrehozza és hozzáadja új sorként.

Mivel a folyamat végére értem így már csak az aktuális tábla alaphelyzet beállítása szükséges az adatok törlése segítségével és az ablak frissítése a már bemutatott `LoadingTheWindow()` metódus segítségével.

5 Gazdasági számítások

A szoftver testre szabás egyik legnagyobb haszna, hogy kevesebb hulladék anyaggal képes legyen dolgozni a felhasználó. A rendezhetőségnek köszönhetően pedig, az újra felhasználható, egy másik projektből származó maradéktáblákat automatikusan képes a rendszer nyilvántartani.

A nyilván tarthatóságnak köszönhetően lehetőség nyílik összhulladékot is nyilvántartani, ami azért lehet fontos, mert ha egy cégnél 12 lézergép vág heti 5 napot egy műszakban 8 órában, akkor az éves szinten keletkezett hulladék elérheti a több száz tonnát is. Ennek a hulladéknak a minimalizálásával pedig nagy mennyiségű pénzt képes megspórolni az alkalmazás.

Sajnos egy kis méretű cégnél, ahol nem folyamatos a megrendelés így a termelés sem állandó. Ezért csak olyan adatokkal számolok, ahol már annak a feltétele is adott, hogy folyamatos legyen a munka.

A kiszolgált cégek méretei alapján átlagos számítást végeztem. Üzleti titok miatt a pontos adatokat nem adhatom ki a szakdolgozatomban, de ha átlagolom ezeket az adatokat, akkor azon cégek, akik megengedhetnek maguknak egy ilyen mértékű fejlesztést, azok minimum 3 lézer géppel rendelkeznek. Ezen 3 lézergép számításomban napi 8 órát dolgozik és heti 5 napot.

A lemez ár is átlagolt számokból jön ki, hiszen folyamatosan nem csak egy fajta lemezt munkálnak meg a cégek és ezen anyagoknak is, különböző vastagságaik vannak. Az egyszerűség kedvért ezt a lemezárat 45 000 forinttal számolom. Egy tábla megmunkálási ideje is széles skálán mozoghat, de átlagosan 15percel fogok számolni. Ezekből az adatokból kiszámolható az óránkénti alapanyag felhasználás forintban kiszámolva.

$$\frac{45\,000\,Ft * 3gép * 60perc}{15perc} = 540\,000\,\frac{Ft}{óra}$$

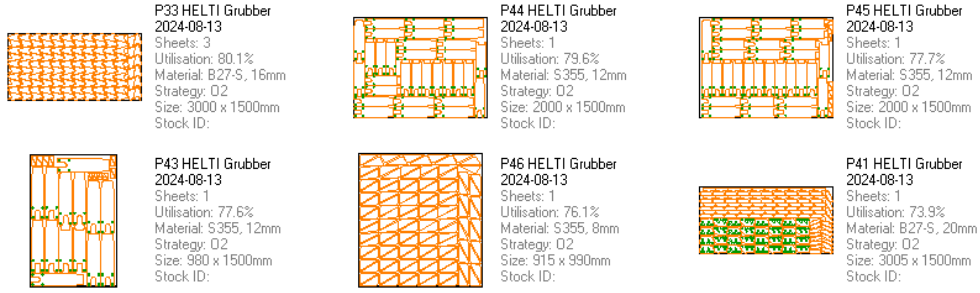
A számolás alapján óránként 540 ezer forintnyi alapanyag kerül megmunkálásra. Ami egy teljes évben elérheti a több mint 1,1 milliárd forintot. Ez az éves elméleti alapanyag felhasználás.

$$540\,000\,\frac{Ft}{óra} * 8óra * 5nap * \ 52hét = 1\,123\,200\,000\,\frac{Ft}{év}$$

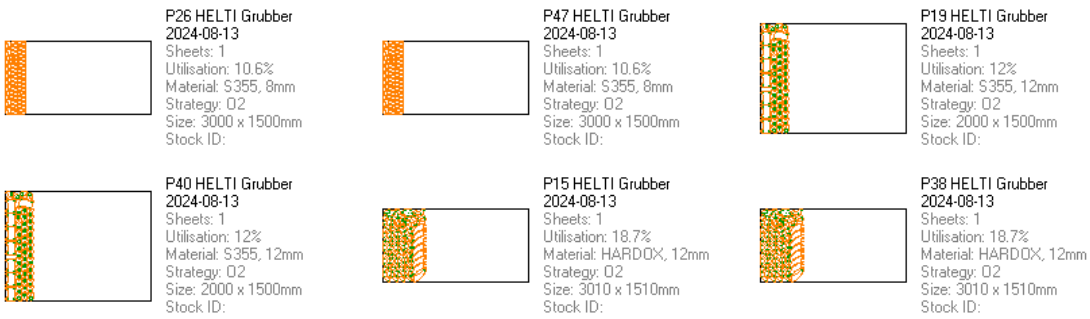
A valóságban a gépek kihasználtsága nem 100% amit a karbantartási idő, szerszámgép kihasználtság, szabadságolások és egyéb okok miatt is szükséges csökkenteni. Így végül 50%-át veszem az éves elméleti alapanyag felhasználásnak. Ezzel megkapom a sokkal valósabb gyakorlati alapanyagfelhasználást forintban kifejezve.

$$1\,123\,200\,000\,\frac{Ft}{év} * 0.5 = \ 561\,600\,000\,\frac{Ft}{év}$$

A számításból kiderült, hogy nagy mennyiségű pénzről van szó. A táblákra az alkatrészek általánosan a RADAN segítségével 85-90%-os kihasználtsággal kerülnek kiosztásra. Ami azt jelenti, hogy 10-15% hulladék marad, ez általában az alkatrész közöket jelenti. A projekt utolsó táblájára viszont általában rosszabb lesz a táblakihasználás, mert már addigra elfogynak az alkatrészek darabszámai. Ilyenkor egy tábla levágással a maradék táblát még újra lehet majd használni vagy ebben a projektben, vagy egy másikban.



19. ábra Magas kihasználtságú alkatrész kiosztások



20. ábra Alacsony kihasználtságú alkatrész kiosztások

Abban az esetben, ha az éves alapanyagfelhasználást elosztom az átlagos tábla árral akkor eredménykét megkapom, hogy hány darab táblát vág a cég évente.

$$\frac{561\,600\,000\,\frac{Ft}{év}}{45000FT} = 12\,480\,\frac{tábla}{év}$$

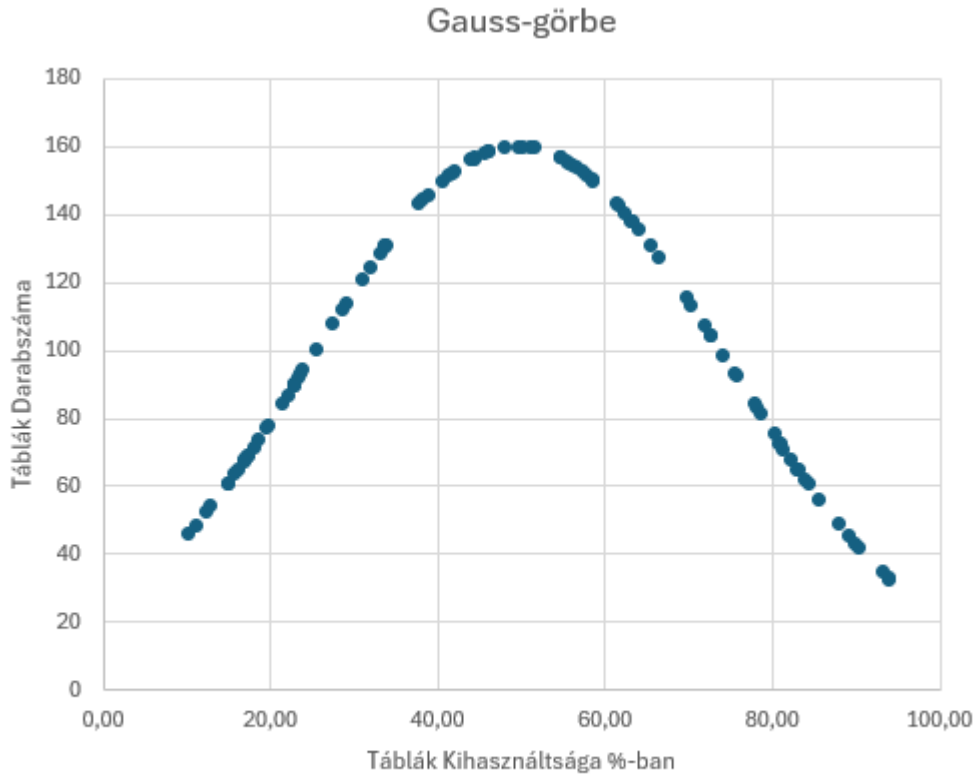
Ha ismét átlagolok akkor kiderül, hogy egy projekt 20 táblát tartalmaz. Ha 20 darabonként történne egy tábla levágás és azt újra fel tudom használni, akkor az újra felhasználható maradék. Bár ezen táblák nem teljes területe lesz újra felhasználható, de a táblalevágások darabszámával egyenlő a maradék táblák száma.

$$\frac{12\,480\ db\ tábla}{20\ db\ tábla} = 624\,\frac{tábla}{év}$$

A terület, amit megtudunk menteni egy tábla levágással nagy mértékben függ, hogy mennyi alkatrész kerül az utolsó táblára. De átlagolásból kiderült, hogy az utolsó tábla területének 50%-t tudom lementeni mint szabadterületű táblarajz. Ezt pedig a raktári rendszerbe

tovább küldeni, mint felhasználható maradéktábla. Az 50% elnagyolt, de segítségével a biztonságosabb megtérülést felé tévedhetünk.

Automatizációs teszt. Készítettem egy kiosztási tesztet, ahol 100 projekt utolsó tábláiból készítettem egy eloszlást. A teszt első adata, hogy hány darab táblát tartalmazott a projekt, ez 10-től 300darab tábláig terjedt. A második adat a tábla levágással rendelkező tábla kihasználtsága. Az adatokból az derülti ki, hogy az esetek többségében az eredeti tábla terület 50%-os területével rendelkeznek a maradék táblák.



21. ábra Tábladarabszám és a maradék tábla összefüggése

Ha ezt a 624 táblát megszorozzuk 50%-kal és felszorozzuk a táblák árával akkor megkapjuk, hogy mekkora mennyiségű pénzt lehet körülbelül megspórolni a maradék anyagkezeléssel.

$$624\ \frac{tábla}{év} * 50\% * 45\ 000Ft = 14\ 040\ 000Ft$$

Természetesen az így kiszámolt összeg csak becsült, hisz nagy skálán mozoghat a táblalevágások száma és kimenthető, újra felhasználható terület is. Viszont látható, hogy jelentős mennyiségű pénzt lehet megtakarítani a maradékanyagkezelés rendszerével.

6 További fejlesztési lehetőség

A szakdolgozatomban szerettem volna még több folyamatot is kidolgozni, ami vagy program futási biztonságot, vagy hatékonyabb működést eredményezhet. Úgy gondolom, ha a funkcionalitáson túli témákat is fel dolgozom azzal túl lépem a terjedelmi korlátokat. Ezeket a folyamatokat a Mester szakon szeretném kidolgozni.

6.1 SQL rendszer bevezetése

A feladat során CSV-t használta, ahol az adatokat tárolom. Ez akkor jelent problémát amikor egynél több felhasználó használja egyszerre ugyan azt a fájlt. Mivel egy akkora szoftvert testre szabást már több felhasználós cégek vásárolnak ezért hatékonyabb megoldás az adatok tárolására egy SQL szerver, ahol egyszerre több felhasználó is hozzá fér az adatokhoz egy időben.

6.2 Automatikus visszatöltés

A folyamatban jelenleg manuálisan történik az esetleges selejtek újra betöltése, egy új projekt esetén. Ezt viszont ki lehet dolgozni úgy is, hogy Projekt sablont használok. Egy ilyen sablonban eddig csak kiosztó paramétereit, automatikus beállítások paramétereit állítottam be. Egy ilyen sablon úgy készül, hogy egy üres projektet lementek. Abban az esetben, ha alkatrészeket is mentek le hozzá akkor az ezzel a sablonnal elkészített üres projektekben szerepelni fog úgy mond alap alkatrészként az esetleges selejt. Ennek köszönhetően már milyen adatbázisból is osztanánk ki a következő projekt alkatrészeit, a lehetséges selejtek mindig szerepelnének a projektünkben és ha van ilyen anyagú alkatrészünk, vagy manuálisan hozzá rendelünk ilyen anyag és anyagvastagságú lemez táblát. Akkor el is készíti a kiosztásokat számunkra a RADAN.

6.3 Folyamatos tábla követés

A RADAN tábla fájlában van lehetőség a még futtatandó darabszám követésére. Ha ez a szám eléri a nullát akkor megmunkálási státusza „vágott”-ra kerül. Az általános felhasználó ezt nem használja hiszen vagy vágottra teszi manuálisan, vagy nem foglalkozik már tovább ezzel a projekttel. Ha viszont ilyen mértékű testre szabást hajtunk végre akkor a szakdolgozatomban említett fájl figyelés miatt könnyedén hozzá férek a tábla fájlhoz és miután egy tábla lejelentésre kerül akkor ugyan azon tábla fájl futtatási számát is csökkentem. Ahogy a **Hiba! A hivatkozási forrás nem található.** mutatja a vágott jelölést a RADAN-ban szürke pöttyel jelöljük, a már kész, de még vágandó táblák pedig zöld pöttyel vannak ellátva.

22. ábra
Jelölések

Ez a funkció akkor lehet hasznos a felhasználó számára, ha egy sok táblás projektben már pár táblát megmunkált, de szeretné új alkatrészekkel föltölteni. Ebben az esetben a jobb kiosztás érdekében a még nem vágott táblákat ki kell törölni és újra kiosztani az újalkatrészekkel. Ezen táblafigyelés segítségével viszont pontosan tudni lehet melyik táblák voltam már vágva és így melyik lakatrészekből mennyi darab készülhetett már el.

6.4 Felhő technológia

A felhasználás az szakdolgozat terjedelmében is részben automatikus. A modern .NET keretrendszernek köszönhetően viszont megoldható úgy is a felhasználás, hogy egy folyamatosan futó számítógépen futtatva a RADAN -t, a vállalat irányítási rendszer erre a gépre küldi ki a megmunkálandó alkatrész adatokat. Ez a gép a feladatokat sorba állítva kapna adatokat és ha az egyikkel végzett akkor a sorban következőre lépne tovább. Ez azért lehetne hasznos a felhasználó szempontjából mert így csak a kisebb módosítások igényelnének emberi felügyeletet, a projektek elkészítése nem.

A másik felhasználási nagy lehetőség ebben a technológiában, hogy akár valamilyen multi platformos applikáció segítségével könnyedén lehet olyan telefonos alkalmazást készíteni, amiben lehet adatokat fogadni az alkatrészek geometriájáról, anyagáról, vastagságáról és darabszámáról. Ennek köszönhetően akár egy tárgyaláson, ahol az alkatrészek árajánlat adásáról esik szó, így képes lehet a felhasználó pontos megmunkálási adatokat mondani pár perc úgy, hogy nem szükséges RADAN lisencel foglalkoznia vagy épp egy komolyabb számítógépet magánál hordania.

6.5 Egységtesztek írása

Az egység tesztek a program legalapvetőbb funkcióinak futását ellenőrzi. Egy egyszerű esetben amikor egy függvénytől azt várom el, hogy kérjen be két számot és adja össze őket akkor azt úgy az összegüket adja vissza, akkor ezt egy úgy nevezett Unittest-el egyszerűen

tudom ellenőrizni, hisz 2+2 eredményeként 4-et várok. Ha az eredmény nem ennyi akkor a metódusom nem működik megfelelően. Az ilyen tesztekből természetesen sokat kell írni és egy jól működő program minden sora van legalább egy módon tesztelve.

Ezen egység tesztek legnagyobb haszna a fentarthatóságban van. Ha egy ilyen egység tesztelt forráskódot elő kell venni újabb fejlesztések érdekében akkor sokkal könnyebb újra átlátni a folyamatokat, hisz mindenről készült egy elvárt működés.

Abban az esetben, ha egy teszt során kiderül, hogy módosítanunk kell egy metóduson, ami több tesztben is szerepel akkor azok a tesztek hibára fognak futni, ha valamit rosszul módosítok. Ez lehetővé teszi, hogy ne kelljen egyesével minden változtatásnál az össze lehetőséget kézzel újra tesztelni, hanem ezred másodpercben mért idő alatt lefutó rengeteg automatizált teszt informál, hogy minden az elvártnak megfelelően működik vagy épp, hogy mi és miért nem működik.

7 Összefoglalás

Szakdolgozatomból kiderült, hogy lehetséges maradékanyag kezeléssel bővíteni a RADAN lemezmegmunkáló szoftvert. Az alkatrészek betöltését és megmunkálását automatikussá tudtam programozni, így a bővítmény használata nem igényleg nagyobb szaktudást vagy speciális oktatást. A betöltendő alkatrészek adatait egy erre a célra kialakított formában egy CSV file tartalmazza, amit feltételezhetően egy vállalat irányítási rendszer ad át a felhasználó részére. Természetesen a szakdolgozatban bemutatott példa fájl oszlopai és adatai módosíthatóak, az adott ügyfél vállalatirányítási rendszeréből érkező CSV-re kell átalakítanom a beolvasott file-t, amikor felhasználásra kerül a bővítmény egy cégnél.

Az alkatrészek betöltését követően, automatikusan betöltődnek a felhasználható raktári táblák. Ezt egy raktári adatbázisból olvasom ki, amely adatokat szintén egy CSV tárol. Ha maradéktábla vágásra kerül a sor, akkor a keletkező maradék táblát lementi ebbe az adatbázisba és azt innen képes vissza olvasni a rendszer, ezért a maradéktábla felhasználás is megvalósul.

A folyamat következő része, hogy automatikusan kiosztásra kerülnek a betöltött táblákon a betöltött alkatrészek. A RADAN belső kiosztó algoritmusa gondoskodik róla, hogy a lehető legkevesebb anyagfelhasználásba kerüljön a tábla kiosztása. Hulladékanyag ilyenkor az alkatrészek közötti úgynevezett csontváz marad. Az alkatrész közökre azért van szükség, mert a bevitt hő hatására a tábla deformálódik és ezért, ha minden úgynevezett közösvágáson történne, akkor akár egyszerre az egész táblakiosztás is selejt lehet. Az alkatrész közök általában 10mm-esek szoktak lenni, viszont a téglalap alakú alkatrészeknél ezt egyszerű elképzelni, ugyanakkor a „banán” alakú alkatrészeket sajnos nagyon nehéz „kézi” módszerekkel kiosztani. A RADAN kiosztó rendszere a piacon a legjobb, eltekintve bizonyos felhős megoldásoktól, ahol természetesen drasztikusan nagyobb gépi teljesítménynek köszönhetően, egy sokkal rosszabb algoritmus is képes jobb kiosztást generálni.

Miután megtörtént a kiosztás az nc kód a szerszámgépre kerül. A futtatás követően, viszont nem garantált, hogy nem keletkezett volna selejt. A lejelentő alkalmazás, amit készítettem telepíthető bármilyen Windows-os felületre, így a feltételezett gyártósori számítógépen a selejt alkatrészek lejelentése lehetséges.

A selejtek lejelentése után, már elengedő információval rendelkezek, hogy készítsek egy újabb CSV-t, ami segítségével nyomon lehet követni a hulladék mennyiséget és az újra vágandó alkatrészek darabszámát. A lejelentő alkalmazás nem csak a selejtek nyilvántartásáért felelős. Ha készül maradéktábla vágás, amit a jövőben fel lehet használni, mint programozható tábla, akkor azt is itt adom hozzá a raktári táblához. Ennek oka, hogy fizikálisan csak itt készül el a

maradéktábla, hiába, hogy tervezetten már az alkatrész kiosztáskor lehet tudni, hogy milyen maradéktábla fog keletkezni.

8 Summary

In my thesis, I demonstrated that it is possible to extend the RADAN sheet metal processing software with scrap material management. I was able to program the loading and processing of parts to be automatic, so the use of the extension does not require extensive expertise or specialized training. The data of the parts to be loaded is contained in a CSV file created for this purpose, which is presumably provided to the user by an enterprise resource planning (ERP) system. Naturally, the columns and data of the example file presented in the thesis can be modified; the file needs to be adapted to the CSV coming from the client's ERP system when the extension is implemented at a company.

After loading the parts, the usable stock sheets are automatically loaded. I retrieve this data from a warehouse database, which is also stored in a CSV. If a remnant sheet is cut, the system saves the remaining sheet into this database and can read it back from there, thus enabling the reuse of remnant sheets.

The next step in the process is that the loaded parts are automatically assigned to the loaded sheets. RADAN's internal nesting algorithm ensures that the sheet is laid out with as little material waste as possible. The waste material, in this case, is the so-called skeleton left between the parts. The part gaps are necessary because the sheet deforms due to the heat applied, and if all cuts were common cuts, the entire sheet layout could potentially be rejected at once. The gaps between parts are usually 10mm wide, and while this is easy to imagine with rectangular parts, it is quite difficult to manually nest "banana"-shaped parts. RADAN's nesting system is the best on the market, except for certain cloud solutions, where, thanks to drastically higher computing power, even a much poorer algorithm can generate a better layout.

Once the nesting is complete, the NC code is transferred to the machine tool. However, after running the code, it is not guaranteed that no defective parts were produced. The reporting application I created can be installed on any Windows platform, allowing for the reporting of defective parts on the assumed production line computer.

After reporting the defective parts, I have enough information to generate a new CSV that can be used to track the amount of waste and the number of parts to be re-cut. The reporting application is not only responsible for tracking defects. If a remnant sheet is created that can be used as a programmable sheet in the future, I add it to the warehouse stock here as well. The reason for this is that the remnant sheet is physically created only here, even though it can already be known during the part nesting process what kind of remnant sheet will result.

9 Köszönetnyilvánítás

Mindenekelőtt külön hálával és köszönettel tartozom a témavezetőmnek Madarász Istvánnak és Tóth Jánosnak, hogy szakmai tanácsaikkal, észrevételeikkel és javaslataikkal segítették munkám és végig érezhettem a támogatásuk. Nélkülük nem sikerült volna.

Továbbá szeretném köszönetem kifejezni Veres Tibornak a külsős konzulensemnek, munkahelyi vezetőmnek az egészéves támogató munkáját, hogy számíthattam a segítségére és időt szakított rám.

Szeretném megköszönni szüleimnek és barátaimnak, akik támogatásukkal, biztatásukkal és oda figyelésükkel hozzásegítettek a szakdolgozatom létrejöttéhez.

Végül, de nem utolsó sorban pedig szeretném megköszönni a feleségemnek, Dórinak, hogy elviselt és támogatott a nehéz időszakokban.

10 Ábrajegyzék

11 Irodalomjegyzék

TRUMPF Werkzeugmaschinen GmbH + Co., (2004) Operator's manual – TRUMPF LASERCELL 1005 3. fejezet (Letöltve: 2024.03.30)

Fledrich G, Kári-Horváth A, Kakuk Gy, Zsidai L, (2017) Mechanikai Technológiák, Szent István Egyetem ISBN: ISBN 978-963-269-596-9

Senthilkumar, V. (2014) Laser cutting process – A Review, International Journal Of Darshan Institute On Engineering Research & Emerging Technologies, Vol. 3, No. 1, 2014. ISSN 2320-7590

Dubey, A. K., Yadava, V. (2008) Multi-objective optimisation of laser beam cutting process, Optics & Laser Technology, Vol. 40, pp. 562–570.

Hügel H., (2000) New solid-state lasers and their application potentials, Optics and Lasers in Engineering, Vol. 34, pp. 213-229

Albahari, J., (2023) C# 12 in a Nutshell The Definitive Reference, O'Reilly Media, Inc., pp. 1-2, ISBN: 978-1-098-14744-0,

Markovits T. (2018). Korszerű lézersugaras technológiák. Akadémiai Kiadó. https://doi.org/10.1556/9789630599405. (Letöltve: 2024. 09. 14.)

Molnár J., Nánási J., Szakály N., Tamás P., Tóth B. (2018). Programozzunk C# nyelven. Akadémiai Kiadó. https://doi.org/10.1556/9789634541141. (Letöltve: 2024. 09. 14.)

12 Egyéb internetes források:

https://www.cnc.hu/2016/08/miert-alkalmazzunk-gorgos-szerszamokat-elhajlitas-soran/
(Letöltve: 2024.09.24)

NYILATKOZAT

a szakdolgozat nyilvános hozzáféréséről és eredetiségéről

A hallgató neve: DEÁK TAMÁS

A Hallgató Neptun kódja: YWJXE1

A dolgozat címe: Maradék anyaggyűjtés fémforgácsoló szoftverben

A megjelenés éve: 2024

A konzulens intézetének neve: Műszaki intézet

A konzulens tanszékének a neve: Gépszerkezettani Tanszék

Kijelentem, hogy az általam benyújtott szakdolgozat egyéni, eredeti jellegű, saját szellemi alkotásom. Azon részeket, melyeket más szerzők munkájából vettem át, egyértelműen megjelöltem, és az irodalomjegyzékben szerepeltettem.

Ha a fenti nyilatkozattal valótlant állítottam, tudomásul veszem, hogy a záróvizsga-bizottság a záróvizsgából kizár és a záróvizsgát csak új dolgozat készítése után tehetek.

A leadott dolgozat, mely PDF dokumentum, szerkesztését nem, megtekintését és nyomtatását engedélyezem.

Tudomásul veszem, hogy az általam készített dolgozatra, mint szellemi alkotás felhasználására, hasznosítására a Magyar Agrár- és Élettudományi Egyetem mindenkori szellemitulajdon-kezelési szabályzatában megfogalmazottak érvényesek.

Tudomásul veszem, hogy dolgozatom elektronikus változata feltöltésre kerül a Magyar Agrár- és Élettudományi Egyetem MATER Hallgatói Dolgozatok repozitoriumába. Tudomásul veszem, hogy a megvédett és

- nem titkosított dolgozat a védést követően
- titkosításra engedélyezett dolgozat a benyújtásától számított 5 év eltelte után

nyilvánosan elérhető és kereshető lesz az Egyetem MATER Hallgatói Dolgozatok repozitoriumában.

Kelt: 2024. év 10. hó 31. nap

Hallgató aláírása

NYILATKOZAT

Deák Tamás (név) (hallgató Neptun azonosítója: **YWJXE1**) konzulenseként nyilatkozom arról, hogy a szakdolgozatot áttekintettem, a hallgatót az irodalmi források korrekt kezelésének követelményeiről, jogi és etikai szabályairól tájékoztattam.

A záródolgozatot/szakdolgozatot/diplomadolgozatot/portfóliót a záróvizsgán történő védésre **javaslom / nem javaslom**[1].

A dolgozat állam- vagy szolgálati titkot tartalmaz: igen nem*[2]

Kelt: Gödöllő 2024.10.30.

_____
belső konzulens

[1] A megfelelő aláhúzandó.
[2] A megfelelő aláhúzandó.