Szegedi Tudományegyetem

Informatikai Intézet

**SZAKDOLGOZAT**

**Ujszászi János**

**2022**

**Szegedi Tudományegyetem**

**Informatikai intézet**

**DXF fájlba mentett épületterv dokumentum megjelenítése és felületszámítása**

Szakdolgozat

Készítette:

**Ujszászi János**

BSC programtervező informatikus

szakos hallgató

Témavezető:

**Dr. Ferenc Rudolf**

egyetemi docens

**Szeged**

**2022**

# Feladatkiírás

Egy épület tervezése során mindig nagy munka mind a tervezőnek, mind az épület leendő tulajdonosának a költségek megállapítása. Ezek a számok az emberi tényező miatt gyakran hibásak, és már csak egy szakágon belül, egy felület mennyiségére is ritka, hogy a tervező, a kivitelező és a tulajdonos ugyanazt az értéket számolja ki. Mindhárom szereplő számára előre lépést jelenthet, ha a számított mennyiségek pontosak.

A hallgató célja egy online platform kialakítása, ahol a felhasználó feltöltheti a tervezőtől kapott DXF fájlt. A feltöltött fájlt parsolja, és tárolja szerver oldalon, a parsolt adatokat relációs adatbázisban menti, canvas objektumban megjeleníti. A számítási algoritmus elméleti szinten tárgyalja magas szintű tervezési metodikával.

A hallagató a feladat megvalósítása közben az előforduló problémaköröket kibontja detektálástól, hibakeresésen keresztül a megoldásig. A fellelt dokumentációk felhasználásának ismertetésével.

# Tartalmi összefoglaló

* ***Téma megnevezése:***

DXF fájlba mentett épületterv dokumentum megjelenítése és felületszámítása

* ***A megadott feladat megfogalmazása***

A hallgató célja egy online platform kialakítása, ahol a felhasználó feltöltheti a tervezőtől kapott DXF fájlt. A feltöltött fájlt parsolja, és tárolja szerver oldalon, a parsolt adatokat relációs adatbázisban menti, canvas objektumban megjeleníti. A számítási algoritmus elméleti szinten tárgyalja magas szintű tervezési metodikával.

* ***A megoldási mód***

Egy HTML/CSS alapú online felületen drag&drop módon feltölthető platform kialakítása. A feltöltött fájl tárolása szerver oldalon és annak parsolás back end oldalon java osztály hierarchiába. Az osztályba mentett adatokat relációs adatbázisban is tárolom. Az osztályokban mentett egyedeket JSON adatformátumként kerül visszaadásra a frontend oldalra, ahol JavaScriptben kialakított bővebb osztálystruktúrába mentem a parsolt adatokat. Minden specifikus Entitás osztálynak felüldefiniált rajzolási eljárása készül, ami egy canvas hívást valósít meg.

A falazatok és egyéb objektumok meghatározására és mérésére kettő lehetséges eljárást ismertetek és a megvalósítás során gyűjtött tapasztalatok alapján.

* ***Alkalmazott eszközök, módszerek***

Az online felületet HTML5, CSS felhasználásával készítem el.

A parsolt fájl osztály leképezéseit H2 relációs adatbázisba mentem. Az adatbázisból a fájl érdemi része visszaállítható.

Backend:

* Java
* SpringBoot

Eredmény visszaírás: JavaScript

* ***Elért eredmény***

A kiírt feladatot sikerült megvalósítani. A fájlt tároltam és feltöltöttem szerver oldalra. Sikerült egy hatékony parsolo eljárást készíteni, amivel a teljes fájlstruktúrát egyetlen bejárással, osztályhierarchiában és adatbázisba tároltam.A megjelenítés során a fontos és elengedhetetlen objektumokat visszarajzoltam és arányosítottam az browser méretéhez.

A kódolás során modul és E2E teszteléseket hajtottam végre.

Kalkulációs megoldások kidolgozása és azok elemzése.

* ***Kulcsszavak***

DXF feldolgozás, parsolás, canvas objektum, osztály struktúra, visszarajzolás, kalkuláció, tervező, megrendelő, kivitelező

**Tartalomjegyzék**

[Feladatkiírás 4](#_Toc100518209)

[Tartalmi összefoglaló 5](#_Toc100518210)

[Bevezetés 9](#_Toc100518211)

[1. Lakóház költség meghatározása 11](#_Toc100518212)

[1.1 Tervezői költség meghatározása 11](#_Toc100518213)

[1.2 Kivitelezői költség meghatározás 11](#_Toc100518214)

[1.3 Megrendelői költség meghatározás 12](#_Toc100518215)

[2. DXF felépítése, adatkapcsolatok kapcsolatok 14](#_Toc100518216)

[2.1. Általános ismertető 14](#_Toc100518217)

[2.2. Felépítése 14](#_Toc100518218)

[2.2.1. Header 14](#_Toc100518219)

[2.2.2. Classes 14](#_Toc100518220)

[2.2.3. Tables 16](#_Toc100518221)

[2.2.4. Blocks 17](#_Toc100518222)

[2.2.5. Entities 17](#_Toc100518223)

[2.2.6. Object 18](#_Toc100518224)

[3. User interfész 19](#_Toc100518225)

[3.1. Beviteli képernyő 19](#_Toc100518226)

[3.2. Eredmény visszajelző képernyő 20](#_Toc100518227)

[4. Parsolás kezelése 22](#_Toc100518228)

[4.1. BackEnd oldali adatszerkezetek használata és osztály struktúra 22](#_Toc100518229)

[4.2. Parsolás menete 23](#_Toc100518230)

[4.3. Parsolt adat átadása FrontEnd felé 24](#_Toc100518231)

[5. Adatbázis felépítése, és tárolás 25](#_Toc100518232)

[5.1. Táblák 25](#_Toc100518233)

[6. Elemek visszarajzolása Javascripttel 27](#_Toc100518234)

[6.1. Osztály struktúra 29](#_Toc100518235)

[6.2. Segédeljárások az ENTITY osztályban aszámtásokhoz 30](#_Toc100518236)

[6.3. Entitás típusokhoz felüldefiniált draw eljárások bemutatása 31](#_Toc100518237)

[6.3.1. 3dface 31](#_Toc100518238)

[6.3.2. Arc 33](#_Toc100518239)

[6.3.3. LINE 34](#_Toc100518240)

[6.3.4. Circle 35](#_Toc100518241)

[6.3.5. AttDef, MTEXT, TEXT, AttRib 36](#_Toc100518242)

[6.3.6. POINT 36](#_Toc100518243)

[6.3.7. LWPOLYLINE 37](#_Toc100518244)

[6.3.8. SOLID 39](#_Toc100518245)

[6.3.9. HATCH 40](#_Toc100518246)

[6.4. Arányok kezelése 44](#_Toc100518247)

[7. Számítási algoritmus megoldási lehetőségei elméleti szinten 45](#_Toc100518248)

[7.1. Objektum felismerés és mérése 45](#_Toc100518249)

[7.1.1. Képfeldolgozás pásztázással 45](#_Toc100518250)

[7.1.2. Folytonos HATCH keresés 45](#_Toc100518251)

[8. Alkalmazás tesztelése 47](#_Toc100518252)

[8.1. Modul szintű tesztelés 47](#_Toc100518253)

[8.2. EndtoEnd tesztelés 47](#_Toc100518254)

[Irodalom jegyzék 47](#_Toc100518255)

[Nyilatkozat 48](#_Toc100518256)

[Köszönetnyilvánítás 48](#_Toc100518257)

[Mellékletek és elektronikus melléklet 48](#_Toc100518258)

# Bevezetés

Egy épület tervezése során mindig nagy munka mind a tervezőnek, mind az épület leendő tulajdonosának a költségek megállapítása. Ezek a számok az emberi tényező miatt gyakran hibásak, és már csak egy szakágon belül, egy felület mennyiségére is ritka, hogy a tervező, a kivitelező és a tulajdonos ugyanazt az értéket számolja ki. Mindhárom szereplő számára előre lépést jelenthet, ha a számított mennyiségek pontosak.

A szereplők között egy tévesen meghatározott érték, már a bizalmi kapcsolat kialakulása előtt, rányomhatja a bélyeget a későbbi kapcsolatra.

* A tervező:
  + Hiányosan jelöli ki tervező szoftverbe a számításban érintett objektumokat,
  + Vagy csak rosszul állítja be az objektumra vonatkozó számítási paramétereket.
* A tulajdonos:
  + Excel táblás struktúrában szinte 100%-os a hibázás lehetőség
  + Hozzá nem értés, egy objektum valós méretét tekintve.
* A kivitelező:
  + Beleszámolja az ablak helyét is a falazásba.
  + Felhasznált anyag alapján számol.

Ezek a példák is jól mutatják, hogy nagyon nehezen tud jól indulni egy ilyen kapcsolat. Ahhoz, hogy egy mindenki számára használható eszközt készítsek, fel kell mérni a szükséges igényeket.

Alapvető igények, elvárások az alkalmazással kapcsolatban:

* Tervezői oldalról egyértelmű elvárás, hogy a szellemi tőke megmaradhasson a tervezőnél. Ezért a nyers tervezésben érintett fájl formátum felhasználása nem lehet opció.
  + Mindhárom oldalról elvárás a kényelem és a gyors számítás. Automatizálással több embernapnyi munka váltható ki egyszerűen
* Mindhárom oldalról elvárás az egyszerűség
  + Egy újabb bonyolult, több opciós paraméterezéssel működő eszköz nem használható hatékonyan
* Eredmények átláthatósága.
  + Opciót kell biztosítani a felhasználó felé, hogy meghatározza, az eredményben visszaadott objektum típusokat, egyedeket.
  + Amennyiben nem definiálja elvárásait a felhasználó, akkor mindent vissza kell adni, ezzel elkerülve, hogy a rengeteg adatban elvesszen a mezei felhasználó.

# Lakóház költség meghatározása

## Tervezői költség meghatározása

Minden épület tervezés ügyfél egyeztetésekkel indul. Az ügyfél szeretne egy épületet, ami az igényeinek megfelelő és ehhez van egy reális vagy éppenséggel irreális költségkerete. A tervező feladatai közé tartozik, hogy a költség keretnek megfelelő épületet tervezzen. Ez a feladat egy a sokból, és ez csak egy paraméter az épület tervezése során, ami sok feladat mögé kerül, mert fontosabb, hogy az épület:

* biztonságos legyen
* megfeleljen a helyi és országos építésihatóság által előírt szabályoknak
* megfelelő és időtállóanyagokkal valósuljon meg a kivitelezés
* illeszkedjen a helyi és/vagy területi építési környezetbe, stílusba

Sajnos a költségvetési kiírás sok esetben messze van a gyakorlati megvalósítástól, illetve az objektumok alapján számol. Ergo, ha egy objektum nincs megnevezve, címkézve azt csak manuálisan tudja hozzátenni a tervező.

Ideális esetben minden lerajzolt vonal, vagy réteg költsége becsülhető lenne, de ehhez minden lerajzolt vonalhoz azonosítás szüksége, hogy az az objektum éppen micsoda. Ez a címkézés időigényes.

A tervezők számára ideális megoldás lehet viszont az, hogy ha a tervezés során részfázisokat kiexportálnak, és azokra végeztetnek költségelemzést.

## Kivitelezői költség meghatározás

A mai építőipari környezetben egyetlen hibásan elkészített árajánlat komoly veszteséget jelenthet egy kivitelező számára, nem csak a félreszámolás miatt, hanem a folyamatos áremelkedés okán is.

Több szempontból sem előnyös, ha a kivitelező számolja ki a mennyiségeket:

* időt és energiát visz el az értékesebb helyszíni szak tevékenységtől
* általában a számolások munka után második műszakban történik, ami növeli a hiba lehetőséget
* hiba esetén bizalmatlanság alakulhat ki.
* visszaélésre adhat lehetőséget mind az ügyfél, mind a kivitelező szemszögéből.

Az ideális támogató rendszer kiszámolja a mennyiségeket és anyag bontásba adja át a kivitelező felé azokat. Egy leolvasott mennyiséghez nagyon egyszerű a rétegrend további elemét kikalkulálni.

Szemléltetés egyszerű példával: Külső falazat.

* Bentről kifelé indulva.
  + A falazatot, csupasz falat belülről borítja valami.
    - Burkolat
      * A burkolat alá vakolat kell
      * Vagy gipszkarton
        + alá vázszerkezet
        + vagy ragasztó
    - Festék
      * A festék alá glett
        + alá vakolat
        + vagy gipszkarton

Ezzel az egyszerű példával szemléltehető, hogy egy meghatározott külső falazat belső felületéhez mennyi különböző anyag mennyiség, és munkabér költség határozható meg.

## Megrendelői költség meghatározás

A megrendelő a tervezői és kivitelezői hibáknak és visszaéléseknek az elszenvedője. Ha nincs a kezében egy eszköz vagy nem kér fel borsos áron ellenőrt, akkor marad az Excel tábla és több hetes számolgatási procedúra.

Az ilyen jellegű ellenőrző tevékenység a következő problémákat hordozza:

* Ha már ellenőrzés szükséges a részünkről akkor már probléma van.
  + Vagy a tervező rontott el valamit vagy a kivitelező számolt rosszul, de a két szám eltér, ezért szükséges az ellenőrzés.
  + Az ügyfél szintű ellenőrzés általában kezdetleges eszközökkel valósul meg, rutin és tapasztalat nélkül.
  + Nagy mértékű hibalehetőség
  + Gyakori hiba kellemetlené teszik a kapcsolatot, hitelesség csökken.
  + Kezdeti költségelemzés meghatározásakor nagy mértékű pontatlanság.

Nincs hatékony segítség a megrendelő kezében.

# DXF felépítése, adatkapcsolatok kapcsolatok

## Általános ismertető

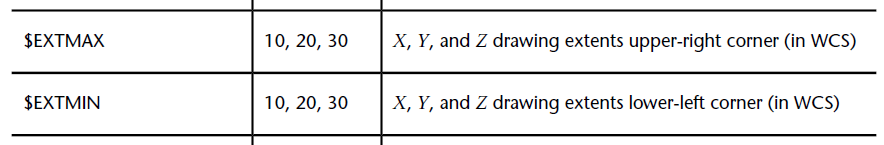
A DXF egy fix szabályrendszer alapján felépített, ASCII szöveges állomány. Két egymást követő sor mindig összetartozik. Az első sor egy típuskódot ad meg, ami három karakter hosszú balról padolva SPACE karakterrel. a második pedig a típushoz tartozó értéket. A fájl több szekcióból állhat, de a szekciók közül csak az ENTITIES kötelező. A fájl végén EOF kulcsszó található.

Minden szekció előtt meg kell adni, hogy egy új szekció következik a „ 0”, SECTION kóddal, ezt követően „ 2”, Szekció neve következik. A tartalmat követően a szekció „ 0” kóddal és „ENDSEC” kulcsszóval zárul.

## Felépítése

## Header

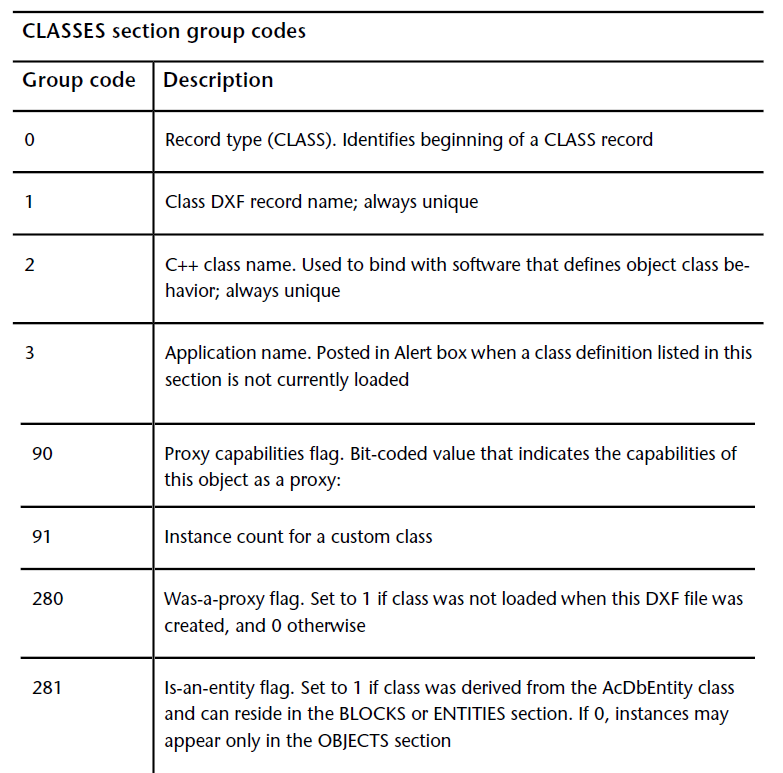
A HEADER szekcióban az általános fájladatok találhatóak a beállított változók formájában. A változók elnevezése általában $ karakterrel kezdődnek. A teljesség igénye nélkül, itt található például a fájl formátumának verzió száma, a kép méretéhez köthető különböző változók, az entitásokhoz társítható általános preferenciák, mint például hogy egy körív előre definiált forgásiránya ($ANGDIR) óramutató járásával ellentétes, vagy nem.



1. ábra Header formátum leírás hivatalos dokumentációból

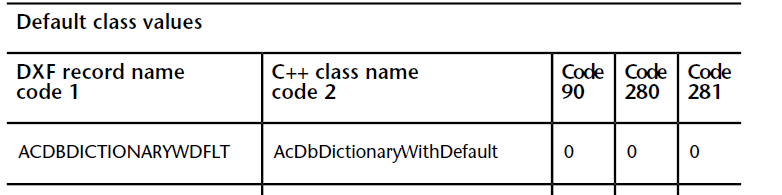
## Classes

A CLASSES szekcióban azok az osztályok találhatóak, amit az exportáló alkalmazás példányosít és később megjelennek a BLOCKS, ENTITIES és OBJE CT szekcióban. Az osztályokat előre definiált mezőkkel lehet leírni és minden mezőnek a kitöltése kötelező. A mezők a 2. ábrában találhatóak.



2. ábra Classes szekciót leíró adatok

Vannak előre definiált osztályok is, amik fix elnevezési párosítással érthetőek el. Ennek a leírásnak a formátuma a 3. ábrában látható.



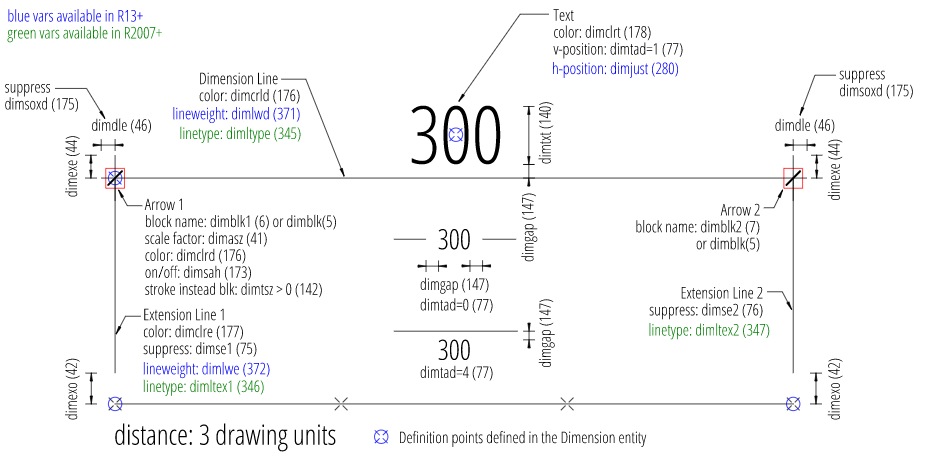
3. ábra Default classes

Az dolgozatban a szekciót nem dolgozom fel. Figyelmen kívül hagyom.

## Tables

A TABLE szekcióban a következő táblázatban megtalálható típus definíciók deklarálhatók.

|  |  |  |
| --- | --- | --- |
| Név | Kód | Leírás |
| Linetype table | LTYPE | A DXF fájlban megtalálható LINE entitásnál megjelenő vonaltípus definíciókat tartalmazza. |
| Layer table | LAYER | Rétegeket lehet definiálni |
| Text Style table | STYLE | A szöveg stílusokat tartalmazza |
| View table | VIEW | A rajzterület elrendezéseinek nézeteit tárolja. Nem befolyásolja a rajzot, de az AUTOCAD alkalmazásnak segít a feldolgozásban. |
| User Coordinate System table | UCS | Nevesített vagy nem megnevezett felhasználói koordináta rendszer, ami a CAD alkalmazások használnak. |
| Viewport configuration table | VPORT | A DXF fájlban megtalálható VIEWPORT entitásnál megjelenő elem definíciókat tartalmazza. |
| Dimension style table | DIMSTYLE | A DIMENSION Entitások stílusmeghatározása |
| Application Identification table | APPID | Alkalmazásoknak fenntartott definíciós lehetőségek |



4. ábra DIMSTYLE ábrázolása (<https://ezdxf.readthedocs.io/>)

Az dolgozatban a szekciót nem dolgozom fel. Figyelmen kívül hagyom.

## Blocks

A blokkok entitások gyűjtemény, amik több példányban elhelyezhetők a rajztérben, eltérő elrendezésben, eltérő helyen.

Egy blokk bejegyzés a BLOCKS szekcióban BLOCK bejegyzéssel kezdődik és ENDBLK-val zárul közben INSERT, ATTRIB, ATTDEF block referenciákat kezel.

Az dolgozatban a szekciót nem dolgozom fel. Figyelmen kívül hagyom.

## Entities

Az EINTITIES az egyetlen kötelező szekció. A rajzi elemeket tartalmazza.

Az ENTITY-k „ 0” kóddal kezdődik. ez mutatja meg a ENTITY típusát. Az ENTITIES végét szintén ENDSEC zárja. Közben akármennyi ENTITY előfordulhat. A szekciónak előforduló entitásoknak vannak általános típus független és típusfüggő jellemzői.

Az általános jellemzők a „ 0” típus definíció, elnevezések, hivatkozások más szekciókra.

A specifikus jellemzők között is átjárás van, de vannak kirajzolhatósághoz elengedhetetlen jellemzők, amiket specifikusan a 6.2 fejezetben fogok kifejteni.

## Object

Az ENTITIES-hez hasonló általános tulajdonságokkal rendelkező szekció, de itt nem grafikus, megjelenítendő elemek találhatóak.

Az OBJECT szekcióban megtalálható elemek például a DICTIONARY, GEODATA, MATERIAL.

Az dolgozatban a szekciót nem dolgozom fel. Figyelmen kívül hagyom.

# User interfész

A User interfész egyképernyős webalkalmazásban jelenik meg. Az alkalmazás UI kialakítása során HTML, CSS, és Javascript technológiák falhasználása történt.

Az animációk megvalósítása @keyframes használatával készült.

A visszarajzolási felület canvas

## Beviteli képernyő

Az alkalmazás induláskor megjelenő háttár egy adásmentes TV képernyőre hasonlítható, amiből kettő másodperc elteltével egy terminál lesz látható. A terminálban üdvözlő üzentek, illetve bemutatkozás jelenik meg sorról sorra.

Az animációk CSS stíluslappal egymás után mennek végbe a következő sorrendben:

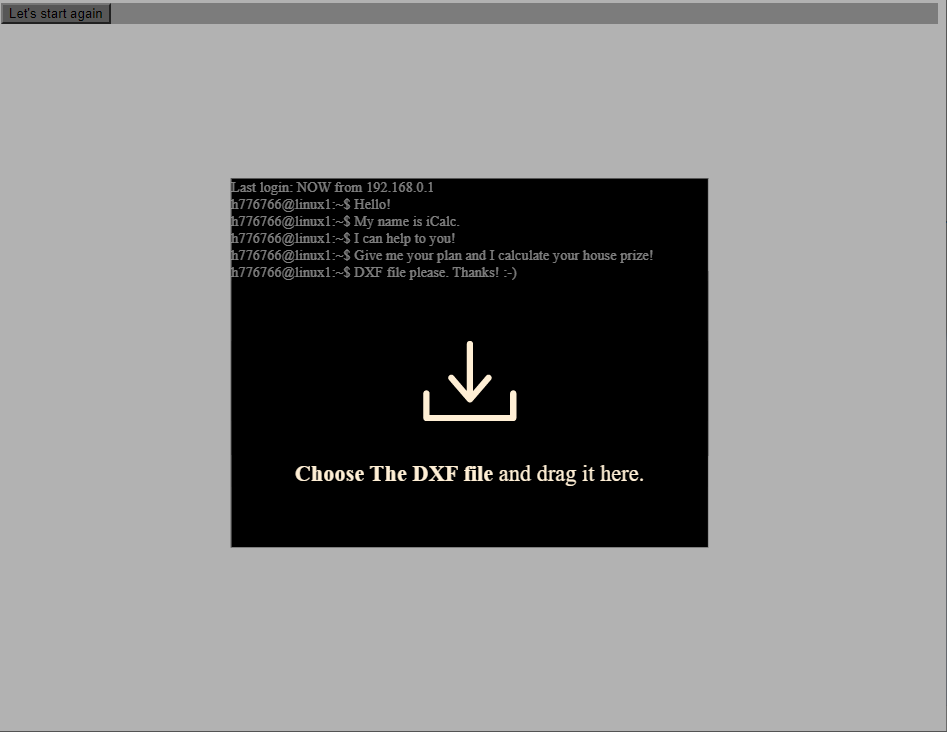
* A @keyframes backgroundchg segítségével a kezdeti noise.gif képet előszőr fehérre módosul
* Majd ugyanebben a @keyframe-ben szürkére.
* Közben a megjelenítem fokozatosan a terminál felületét
* A terminálban miután a háttér szürkére váltott a @keyframes cursor-visible animációval egymás után jelenítem meg a kiírt szöveget.
* Végül megjelenik a drag&drop area-t

A bemutatkozást követően előtűnik a drag&drop felület ahová a felhasználó feltöltheti a kezelendő DXF fájlt.

A fájl felületre mozgatása módosítja a feltöltés alatt megtalálható szöveget

A fájl elengedés és a feldolgozó eljárás megkezdése szintén visszajelzésre kerül a felhasználó számára.

Ha fájl mérete túl nagy vagy a kiterjesztése nem DXF, azt alert formájában visszajelzem a felhasználó felé, és ismételt fájlfeltöltés lehetséges.



5. ábra kezdő képernyő

## Eredmény visszajelző képernyő

A beolvasást követően a képernyőről eltűnik a terminál szöveges felület és browser méretével arányosan 90%.-osra nagyítom a terminál DIV objektumát. Az animációt követően a terminál DIV objektuma eltűnik és ugyanebben a pillanatban megjelenítem az azonos méretű és pozíciójú canvas objektumot.

A canvas jobb oldalának 220 pixel nagyságú részében egy calculation rész helyezkedik el ahova majd a számítási eredményeket írnám vissza.

A maradék rajzfelület 1%-os kertet kap.

A visszarajzolás során a rajzokat arányosítani kell, ezért minden esetben az eredeti méretarányokat megtartva adom vissza az eredményt a felületre. Az arányosítással kapcsolatos teljes leírás az 5.3. fejezetben lesz elérhető.



6. ábra Eredmény visszaírása képernyő

# Parsolás kezelése

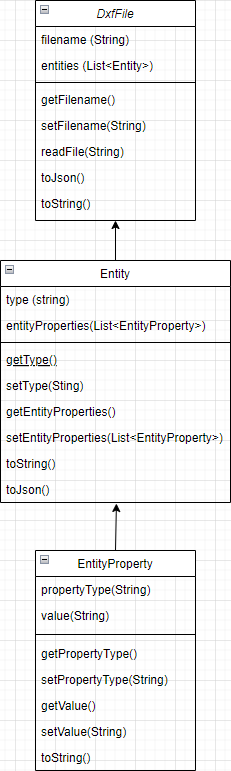
Az adatok parsolása során kihasználom a formátum legfontosabb szabályát miszerint a teljes formátum adatpárokból épül fel. Tehát egy típus után a következő sorban egy érték fog jönni.

Mivel nagy az adathalmaz a parsolásnak gyorsnak kellett lennie ezért a kezdeti feldolgozó eljárás a fájl első bejárása során a teljes fájlt kezelte és strukturálta osztályokba.

Az átadott fájlokat backend oldalon mentem.

## BackEnd oldali adatszerkezetek használata és osztály struktúra

A DXF file backend oldali kezelését három osztállyal lehet megvalósítani, amik között erős aggregációs kapcsolat van.



7. ábra backend class diagram

## Parsolás menete

A parsolás eljárás során a HEADER szekcióból a $EXTMIN és $EXTMAX értékeket gyűjtötte össze, mert ezek az adatok határozzák meg a rajztábla jobbfelső és bal alsó pontját, ami ahhoz szükséges, hogy a visszarajzolt kép elhelyezkedése és mérte megfelelő legyen Ezeket az adatokat entitásként kezeli a parser.

A header szekcióból érkező kettő speciális entitást a tömb elejére helyeztem el. Ezeknek az entitásoknak négy entityproperty-vel kellett dolgozni. A 10,20,30 egy pontot írt le a 9-es pedig figyelmen kívül hagyható volt.

A parsolás további része akkor kezdődik, ha a fájlban elérem a ENTITIES szekciót.

A while futása közben segéd változókkal határozom meg hogy éppen milyen osztályból kell példányosítani. Az entitás kezdetekor („ 0” típus ID esetén) kigyűjtöm az entitás típusát és létrehozom a EntityProperties listát, amiben gyűjtöm a következő „ 0” értékig az entityproperty értékpárokat. A lista zárásával az entity objektumot példányosítom és ürítem az entityproperty listát. A boolean segédváltozókat a kezdeti állapotba állítom.

A parsolás metódus egészen a ENDSEC részig fut. Ezt követően a file feldolgozása már nem hoz létre entitásokat.

A feldolgozás végén egy kapcsolható logolás található, ami kilistázza a kigyűjtött entitásokat. Feldolgozási idő gyorsítása miatt volt szükség a logolás kapcsolhatóságára, mert a logolás gyakorlatilag újra végig olvassa az összegyűjtött entitások tömbjét.

## Parsolt adat átadása FrontEnd felé

A feldolgozott adatok visszaküldése frontend oldalra a JSON formátumban valósult meg. Az osztály struktúra DxfFile és Entity osztályai a toJson eljárás segítségével állították elő a JSON formátumot, a JSON eljárás végén az osztály felüldefiniált toString eljárását használtam, így a return már egy valid JSON formátumot eredményezett.

Azért ezt a formátumot használtam, mert a JSON formátum a JS frontendnek az egyik legmegfelelőbb struktúra, mert a visszatérő stringet a json() hívással könnyen kezelhető objektummá lehet alakítani.

# Adatbázis felépítése, és tárolás

A parsolt adatokat backend oldalon táblákba rendezem. A rendezett adatokból, egy fájl visszaalakíthatóvá, de az aktuális parsolo logika szándékosan szűri a tárolandó adatok mennyiségét a szekciók segítségével.

Az adatbázis egy H2 adatbázis lett.

## Táblák

A „DXF\_FILE” nevű tábla fogja tartalmazni a fájl feltöltésének eseménykor elérhető adatokat.

Mezők:

|  |  |  |  |
| --- | --- | --- | --- |
| Mezőnév | Megnevezés | Megkötés | típus |
| ID | szekvenciából osztott unique index | not null constraint | Long |
| FILENAME | fájl név | not null constraint | Sting |

A „DXF\_ENTITY” nevű tábla tartalmazza az entitás fő adatait az ENTITIES szekcióból.

Mezők:

|  |  |  |  |
| --- | --- | --- | --- |
| Mezőnév | Megnevezés | Megkötés | típus |
| ID | szekvenciából osztott unique index | not null constraint | Long |
| DXF\_FILE\_ID | Foreign key a file\_basic\_data táblához | not null constraint | Long |
| TYPE | Entitás típusa | not null constraint | String |

A „ENTITY\_PROPERTY” nevű tábla tartalmazza az entitáshoz köthető tulajdonságok azonosító típuskódját és az ahhoz tartozó értéket. adatait az ENTITIES szekcióból.

Mezők:

|  |  |  |  |
| --- | --- | --- | --- |
| Mezőnév | Megnevezés | Megkötés | típus |
| ID | szekvenciából osztott unique index | not null constraint | Long |
| DXF\_ENTITY\_ID | Foreign key a entities\_basic\_data táblához | not null constraint | Long |
| PROPERTY\_TYPE | EntitásPropery típusa |  | String |
| VALUE | EntitásProperty típushoz tartozó érték |  | String |

# Elemek visszarajzolása Javascripttel

A visszarajzolás során a JSON struktúrában érkező adatokat javascript osztályszerkezetű objektumokba szervezem.

A JSON adatok első elemei a $EXTMAX és $EXTMIN entitások voltak. Ezek feldolgozását a cikluson kívül kezelem, hogy példányosítani lehessen egy DxfFile objektumot, ahova a további entitásokat és a hozzájuk tartozó entitástulajdonságok típus, érték párjait tároltam el.

Az aktuális hivatalos DXF formátum leíró dokumentáció alapján 45 grafikus entitás objektum kezelése lehetséges a formátummal. Mind a 45 grafikus entitás lekódolása nagyon sok időt venne igényben, ezért meg kellett határoznom a projekthez szükséges feltétlenül lekódolandó megjelenítendő entitás típusokat.

Az interneten nem találtam olyan jellegű kimutatást, amiből ki lehetett volna indulni, és olyan segéd eljárást sem, ami szűrte volna a fájlok tartalmát, ezért a parsolo eljáráshoz készült egy modul, ami a kívánt módon működik.

A modul segítségével tároltam egy fájlba az összes leparsolt fájlban fellelhető entitás típust, és ebből egy xls fájlba csináltam egy rendezett kimutatást. A kimutatás alapján összeraktam egy listát a leggyakrabban használt entitásokról. A hivatalos DXF formátum leíró dokumentáció alapján megvizsgáltam, hogy ezek valóban rajzoló vagy textúrális entitások-e.

Ezeknek sorrendje a következő volt:

1, Line

2, Arc

3, Hatch

4, Circle

5, LWPolyline

6, MText

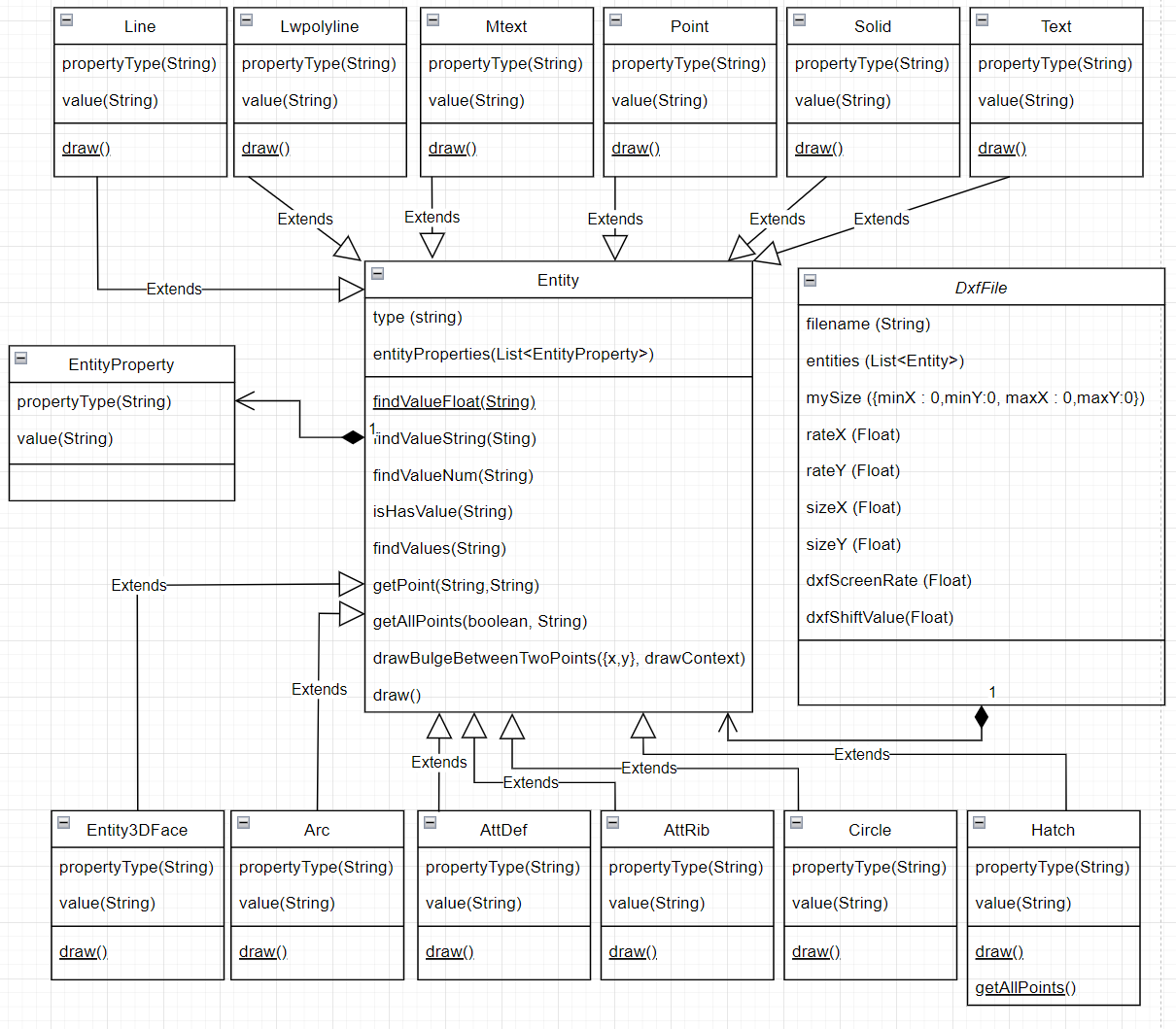
7, 3dFace

A lista alapján ezek az entitások lefejlesztése elengedhetetlen volt az objektum visszarajzolásához,

A DXF fájlleíró dokumentáció alapján kiválasztottam azokat az entitásokat, amik a fenti típusok alapján könnyen lefejleszthető voltak. A később beolvasott tesztfájlok alapján a kiválasztott entitások hozták az elvárt eredményt.

## Osztály struktúra

Az JavaScript osztály struktúrát a következő CLASS diagramm írja le.



8. ábra Javascript class diagram

## Segédeljárások az ENTITY osztályban a rajzpont számtásokhoz

* + - 1. Pontadatok kezelése:

getPoint: Kettő beérkező típus alapján visszaad egy point objektumot

Mivel nem csak egy fix pártípus tud leírni egy pontot ezért bejövő paraméterként szükséges volt megadni a keresendő típus értékeket.

A visszaadott objektum könnyen kezelhető a hívó oldalon.

A következő eljárásban megjelenik a bulge mint adat. A bulge egy ívet jelent a két pont között az egyenes helyett. A 1.0 értékű bulge jelenti a tökéletes félkörívet a két pont között a -1.0 pedig a negatív irányba határozza meg ugyanezt. A bulge értéke a két pont között megtalálható szakaszfelező hosszát arányosítja. Tehát az iv legfelső/legalsó pontja pont olyan magasan lesz mint a szakaszfelező hosszának és a bulge értékének a szorzata.

getAllPoints: bejövő paramétereként megjelenik az isbulge boolean adat és a bulge adat típusa. Az eljárás egy point objektum listát ad vissza, ahol meg tud jelenni a bulge adata is.

A bulgeType értéke eltérő az entitások között ezért szükséges megadni azt az eljárás híváskor.

* + - 1. Érték meghatározás

A következő eljárások bejövő típushoz adnak vissza értékeket.

findValueFloat: Egy bejövő property típushoz ad vissza float értéket.

findValueString: Egy bejövő property típushoz ad vissza string értéket

findValueNum: Egy bejövő property típushoz ad vissza string értéket

isHasValue: Egy bejövő property típushoz visszaadja hogy van-e az entitásban ilyen típusú property.

findValues: Egy bejövő property típushoz adja vissza listában az összes előforduló értéket

* + - 1. Bulge kezelés

drawBulgeBetweenTwoPoints: bejövő paramétereként kettő point objektum illetve a rajztér a rajzoláshoz. Visszatérési érték nincs a függvény a lefutás végén rajzol.

A bejövő paraméterben meghatározható első és második pont közé rajzolni kell egy vonalat, ha null vagy 0.0 értékű az első pont bulge értéke vagy egy körívet, ha ettől eltérő

Több matematikai művelet szükséges a teljes folyamat megvalósításához.

* Megkeresem a domborítandó, vagy homorítand szakasz közepét
* A szakaszfelező pontot elviszem az origóba
* Elforgatom az origó körül 90 fokkal.
* Kiszámítom a domborítás/homorítás tetejét
* Megállapítom a szakaszfelező merőleges hosszát az eltolt középpont és a domborítás/homorítás távolságából.
* Ezt a pontot visszateszem a helyére
* Kiszámolom a szakasz kezdő, végpontja és domborítás teteje pontokkal meghatározott háromszög köré rajzolható kört
* Meghatározom a szögeket, amik meghatározzák a levágandó körív valós hosszát.

## Entitás típusokhoz felüldefiniált draw eljárások bemutatása

Minden rajzolható entitás őse az ENTITY osztály, amiben megtalálható draw() függvény felüldefiniálom. A következő fejezetben ezeket a felüldefiniálásokat fogom bemutatni.

A folyamat megkezdése előtt a rajzvászonra meghívom a beginpath() eljárást, ami nem engedi hogy olyan elem legyen a rajzolási metodikában, amire még nem hívódott fill() mint kitöltés vagy stroke() mint kirajzolás parancs.

### 3dface

Az entitás négy darab zárt vonalhalmazt tartalmaz. Specializációja, hogy bináris változóval jelölni lehet, hogy egy vonal látszódik vagy nem.

* + - 1. Rajzolásban érintett entityproperties

|  |  |
| --- | --- |
| Típuskód | Leírás |
| 39 | A rajzolás során használt vonalhossz vastagsága. Ha nem érkezik változó szinten definiálható alapértelmezett vonalvastagság. |
| 70 | Bináris blocktype flag. Azt jelöli, hogy melyik vonal látható és melyik nem.0-16ig vehet fel bináris értékeket. Az egymás utáni helyiértékek jelölik, hogy a négy vonal közül melyik látszódik. |
| 10 | első pont X koordinátája |
| 20 | első pont Y koordinátája |
| 11 | második pont X koordinátája |
| 21 | második pont Y koordinátája |
| 12 | harmadik pont X koordinátája |
| 22 | harmadik pont Y koordinátája |
| 13 | negyedik pont X koordinátája |
| 23 | negyedik pont Y koordinátája |

* + - 1. Visszarajzolás kezelése

A „ 70”-es kódhoz tartozó értéket stringként karakteresen értelmezem hátulról. Abban az esetben ha a karakter 0 értékű, akkor kirajzolom a megfelelő két pont közötti vonalalt, ellenkező esetben a feldolgozással és nem történik rajzolás.

A canvas szinten a moveto(), line to() függvényeket használom a valós rajzolás elvégzéséhez.

* + - 1. Felmerült problémák kezelése

A fejlesztés során nem volt egyértelmű, hogy honnan indítom a bináris fájl feldolgozását. Valójába ha megnézem a csatolt kódot látható, hogy a bináris stringet hátulról kell olvasni, de a pontokat elölről kell feldolgozni fel.



9. ábra 3dFace forrás

### Arc

Az entitás egy körív objektumot ír le. Nagyon hasonlóan, mint a canvas objektum. A lényegi különbség a rajzolás kezdő pontja.

* + - 1. Rajzolásban érintett entityproperties

|  |  |
| --- | --- |
| Típuskód | Leírás |
| 39 | A rajzolás során használt vonalhossz vastagsága. Ha nem érkezik változó szinten definiálható alapértelmezett vonalvastagság. |
| 10 | kör középpontjának X koordinátája |
| 20 | kör középpontjának Y koordinátája |
| 40 | Sugár |
| 50 | Körív kezdetének szöge |
| 51 | Körív zárásának szöge |
| 210 | bejárási irány (default = 0 (óramutató járása szerint), 0, 1) |

* + - 1. Visszarajzolás kezelése

A canvas objektum és az ARC leíró entitás ugyanazokat az értékeket veszi fel. A szögeket radiánban kellett megadni. A óramutató járás kezelés is megegyezett ezért azt is csak át kellett adni a rajzfelületnek az arc() függvényhívással.

* + - 1. felmerült problémák kezelése

A arc entitás és a canvas objektum körív rajzolónak eltérő kezdőponttal indítja a rajzolást. Ezzel sajnos nem voltam tisztában, ezért nagyon sokat kellett debugolni. Végül a következő szögátalakítással lehetett kezelni az eltérő kezelést.

Math.PI/180\*-1\*(this.findValueFloat(' 50')-360)

### LINE

Az entitás egy darab zárt vonalhalmazt tartalmaz.

* + - 1. Rajzolásban érintett entityproperties

|  |  |
| --- | --- |
| Típuskód | Leírás |
| 39 | A rajzolás során használt vonalhossz vastagsága. Ha nem érkezik változó szinten definiálható alapértelmezett vonalvastagság. |
| 10 | a kezdőpont X koordinátája |
| 20 | a kezdőpont Y koordinátája |
| 11 | a végpont X koordinátája |
| 21 | a végpont Y koordinátája |

* + - 1. Visszarajzolás kezelése

A visszarajzolás során először rajzolás nélkül el kell mozgatni a rajzolás kezdőpontját a 10,20-es value értékekre a moveTo() függvénnyel. Ezt követően rajzolható ki a vonal a lineTo() függvénnyel.

### Circle

Az entitás egy speciális körív objektumot ír le. Eltérése a körívtől hogy nincsenek szög értékek és bejárási irány

* + - 1. Rajzolásban érintett entityproperties

|  |  |
| --- | --- |
| Típuskód | Leírás |
| 39 | A rajzolás során használt vonalhossz vastagsága. Ha nem érkezik változó szinten definiálható alapértelmezett vonalvastagság. |
| 10 | kör középpontjának X koordinátája |
| 20 | kör középpontjának Y koordinátája |
| 40 | Sugár |

* + - 1. Visszarajzolás kezelése

A canvas objektum arc-ként valósul meg ezért a kezdő szöget 0 értékkel állítom be a vég szöget pedig Math.PI \* 2 értékkel. A rajzolás iránya nem releváns.

### AttDef, MTEXT, TEXT, AttRib

Szöveg megjelenítésére alkalmas entitás. A projekt szempontjából csak a szöveg megjelenítése a lényeg. Az érdemi változást az entitások között nem kell külön részletezni.

* + - 1. Rajzolásban érintett entityproperties

|  |  |
| --- | --- |
| Típuskód | Leírás |
| 1 | megjelenítendő szöveg |
| 10 | a kezdőpont X koordinátája |
| 20 | a kezdőpont Y koordinátája |
| 40 | a szöveg mérete |

* + - 1. Visszarajzolás kezelése

A 10, 20 értékkel megtalálható kezdő pontra mozgatom a kurzort. Beállítom a 40-es értékben érkező betűméretet. Ha nem érkezik akkor a kód szinte meghatározott default értékkel dolgozik.

### POINT

* + - 1. Rajzolásban érintett entityproperties

|  |  |
| --- | --- |
| Típuskód | Leírás |
| 10 | a kezdőpont X koordinátája |
| 20 | a kezdőpont Y koordinátája |
| 39 | Kitöltött pont sugara |

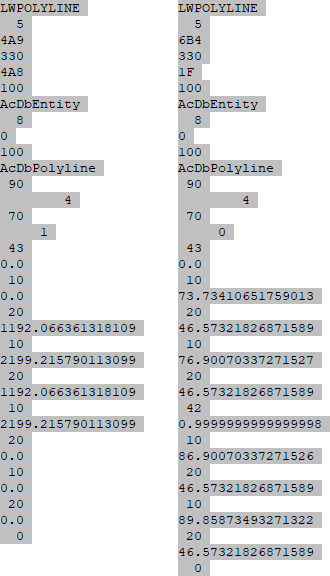
* + - 1. Visszarajzolás kezelése

Egy kitöltött kört kell megjeleníteni. A 10,20 értékkel megtalálható kezdő pontra mozgatom a kurzort. Beállítom a 39-es értékben érkező pont méretet. A fill() paranccsal a rajztérre kitöltött kör rajzobjektumot tudom létre hozni.

### LWPOLYLINE

Egy n vonalból álló objektum, ahol n természetes szám. Minden vonal az utána következő ponttal van összekötve. A formátum jelöli, hogy zárt vagy nyitott a vonalhalmaz.

Az LWPOLYLINE objektum kezeli a pontok közötti görbe vonalat is így lehetséges bulge érték is a pont leírásban. A következő ábra fogja megmutatni a valódi formátumokat.



10. ábra LWPolyline 42-es bulge nélkül és 42-es bulge értékkel

* + - 1. Rajzolásban érintett entityproperties

|  |  |
| --- | --- |
| Típuskód | Leírás |
| 43 | A rajzolás során használt vonalhossz vastagsága. Ha nem érkezik változó szinten definiálható alapértelmezett vonalvastagság. |
| 10 | a 0…n-edik pont X koordinátája (a) |
| 20 | a 0…n-edik pont Y koordinátája |
| 42 | a 0…n-edik pont bulge értéke |
| 70 | Zárt polyline 0 érték. nyitott polyline 1-es érték |

* + - 1. Visszarajzolás kezelése

Egy pont kettő fix és 1 opcionális mezővel rendelkezik. A 10-es és 20-as koordináta érték minden esetben érkezik. Ha az vonalak között van görbe vonal akkor az egyenes vonalakhoz is érkezhet bulge type 0.0 értékkel. Ha kizárólag egyenesek vannak az entitásba akkor nem érkezik bulge érték egyik ponthoz sem.

A „ 70” -es típus kód mutatja meg hogy a vonallánca az nyitott vagy zárt. Nyitott esetben a vonallánc végpontja a kigyűjtött pontok utolsó pontja. Zárt esetben az utolsó pont a kezdő pont lesz.

* + - 1. Felmerült problémák kezelése

Nehezen sikerült detektálni, hogy mikor érkezik egy pont listában bulge érték. Több fájl teljes vizsgálata alapján tudtam leszűrni a bulge-re vonatkozó fenti ténymegállapításokat.

### SOLID

A solid grafikai objektum egy 3 vagy 4 pontból álló kitöltött alakzat.

* + - 1. Rajzolásban érintett entityproperties

|  |  |
| --- | --- |
| Típuskód | Leírás |
| 39 | A rajzolás során használt vonalhossz vastagsága. Ha nem érkezik változó szinten definiálható alapértelmezett vonalvastagság. |
| 10 | első pont X koordinátája |
| 20 | első pont Y koordinátája |
| 11 | második pont X koordinátája |
| 21 | második pont Y koordinátája |
| 12 | harmadik pont X koordinátája |
| 22 | harmadik pont Y koordinátája |
| 13 | negyedik pont X koordinátája |
| 23 | negyedik pont Y koordinátája |

* + - 1. Visszarajzolás kezelése

Első lépésként moveTo függvényhívás segítségével beállításra kerül a grafikai objektum kezdő pozíciója. Ezt követően, ha volt 13 és 23-as típusú pontleíró tulajdonsága akkor négy vonallal körbezárt alakzatra futott le a fill eljárás, ellenkező esetben a zárt alakzat amire fill parancs ki lett adva csak három vonalból állt.

### HATCH

A HATCH (kitöltés) objektum a legösszetettebb feldolgozott objektum, ami szerepel a projektmunkában. Az objektum leírás felhasználja az ARC, LINE, LWPolyline, ELIPSE, SPLINE objektumokat. Az adatgyűjtés során látható volt, hogy kiemelkedően sok entityproperty-vel rendelkezik az objektum.

A legkülönfélébb objektumok előállíthatók vele, ezért nagyon érdekes volt a munka az objektummal.

Ennél az objektumnál a pontok összegyűjtés módját is felül kellett definiálni, mert az öt különböző esetben, máshogy kell megállapítani azokat.

* + - 1. Rajzolásban érintett entityproperties

Általános leíró adatok:

|  |  |
| --- | --- |
| Típuskód | Leírás |
| 91 | A feldolgozandó összeköttetések darabszáma |
| 72 | A HATCH-ben szereplő pontok közötti összekötések típusa 1 = Line; 2 = Circular arc; 3 = Elliptic arc; 4 = Spline 0 = polyline |

A „ 72” kód alapján változó definiálás.

POILYLINE

|  |  |
| --- | --- |
| Típuskód | Leírás |
| 72 | Bulge flag boolean |
| 73 | Zárt-e boolean |
| 93 | Csúcsok száma |
| 10 | a 0…n-edik pont X koordinátája (a) |
| 20 | a 0…n-edik pont Y koordinátája |
| 42 | a 0…n-edik pont bulge értéke |

LINE

|  |  |
| --- | --- |
| Típuskód | Leírás |
| 10 | első pont X koordinátája |
| 20 | első pont Y koordinátája |
| 11 | második pont X koordinátája |
| 21 | második pont Y koordinátája |

ARC

|  |  |
| --- | --- |
| Típuskód | Leírás |
| 10 | a 0…n-edik pont X koordinátája (a) |
| 20 | a 0…n-edik pont Y koordinátája |
| 40 | a 0…n-edik ponthoz tartozó sugár |
| 50 | a 0…n-edik ponthoz tartozó körív kezdetének szöge |
| 51 | a 0…n-edik ponthoz tartozó körív zárásának szöge |
| 73 | a 0…n-edik ponthoz tartozó bejárási irány (default = 0 (óramutató járása szerint), 0, 1) |

ELLIPSE

|  |  |
| --- | --- |
| Típuskód | Leírás |
| 10 | a 0…n-edik pont X koordinátája (a) |
| 20 | a 0…n-edik pont Y koordinátája |
| 11 | a 0…n-edik ponthoz tartozó hosszabb fő átló X koordinátája |
| 21 | a 0…n-edik ponthoz tartozó hosszabb fő átló Y koordinátája |
| 40 | a rövidebb fő átló hossza |
| 50 | a 0…n-edik pont hoz tartozó körív kezdetének szöge |
| 51 | a 0…n-edik ponthoz tartozó körív zárásának szöge |
| 73 | a 0…n-edik ponthoz tartozó bejárási irány (default = 0 (óramutató járása szerint), 0, 1) |

* + - 1. Visszarajzolás kezelése

Egy HATCH entitás során a pontok és azok között megrajzolandó összeköttetések legyűjtése is kihívást jelent.

A „ 72” -es típus kód alapján három különböző képpen gyűjtöttem össze a pontoknak a halmazát.

A fenti táblázatban megtalálható objektumként szerepelt egy-egy pont rekord.

A visszarajzolást szintén a „ 72”-es kód vezérlete.

Attól függetlenül, hogy nem találtam olyan HATCH mintát, ahol egy HATCH-en belül több különböző összeköttetés megjelent, a kód úgy készült el hogy tudja kezelni.

## Arányok kezelése

Ahhoz hogy a megfelelő pontok a megfelelő helyre kerüljenek arányosítani kellett a reszponzív megjelenítő felületet a DXF kép méretéhez.

A parsolás első két entitása a header szekcióból lett kinyerve. Ezek a mezők mutatták meg az arányosításokat és az esetleges eltoláshoz köthető feladatokat.

A DXFFile osztály példányosítása során a konstruktor már a bejövő adatokból kalkulál felhasználható értékeket. A elem példányosításakor átadásra kerül a dxf fájl rajztábla mérete. Ezek a koordináta adatok nagyon ritkán kezdődnek 0,0 értékkel.

A arányosításhoz szükséges kalkulációt eltolással kellett kezdeni, mert a canvas rajztábla koordinátája 0,0 értékkel indul.

A folyamatot ezen a ponton ketté kellett választani. Ha minden oldalra elvégezné, az arányosítást akkor a kép torzul. Egyik oldalt lehetett csak arányosítani a dxf adatok és canvas értékeknek megfelelően. A másikat a már kikalkulált aránnyal kellett kalkulálni.

# Számítási algoritmus megoldási lehetőségei elméleti szinten

A dolgozat további részében elméleti szinten kidolgozott megoldási lehetőségeke fogok bemutatni a számítás elvégzéséhez.

A téma teljes implementálását a MSC-re szeretném elkészíteni.

## Objektum felismerés és mérése

### Képfeldolgozás pásztázással

A kirajzol és elkészült képen soronként keresem az első kettő és utolsó kettő be és kilépési pontot. Egy ház tervénél ezeknek a ki és belépő ponthalmazoknak egy-egy párhuzamos vonalat kell alkotnia.

A detektált falak vég és kezdőpontjaihoz DIMENSION jellegű objektumot kell keresni. A DIMENSION objektumba tárolom za elemhez tartozó méreteket. A DIMENSION text érétkében megjelölt szám alapján mennyiség már arányosítható.

### Folytonos HATCH keresés

A dolgozat megírása során több olyan lehetőséggel találkoztam, amivel a kalkuláció eredményesen elvégezhető, de a legizgalmasabb megoldást ez a fejezet tárgyalja-

A folyamat kezdetekor beolvasásra kerül a teljes ponthalmaz az összes objektumhoz. A ponthalmazokat a megfelelő osztályhoz társítható módon állítom elő, de összegezve egy-egy tömbbe tárolom őket.

A kapott tervrajzon megállapítom a kép legszélső kettő párhuzamos oldalpárjának kezdő pontjait. Ugyanezt a folyamatot elvégezem fordítva is, így meglesz a négy legszélsőbb pontja az objektumnak. A legszélső pontot viszont definiálni kell mert az X és Y tengely vizsgálata közben eltérhet a két legszélső pont. Abban az esetben ha X és Y tengely vizsgálata esetén is ugyanaz a legszélső pont akkor tárolható. Ellenkező esetben fel kell venni az eltérő pontokat is legszélső pontnak, mert ritka a négyszögletes ház a mai világban.

A detektált pontokat rekurzívan vizsgálom és tárolom hogy mennyi hívást követően lehet elérni a legközelebbi ponthoz. Tehát vizsgálni kell, hogy egy entitáson belül szerepel-e a két pont vég és vagy kezdőpontként e. A rekurzió a detektált pontok számánál nem lehet több. Abban az esteben ha több kiesik a tömbből.

A megmaradt tömbök halmazából HATCH keresési algoritmust kell futtatni, aminek lényege a zártság. Visszatérünk a kezdeti adathalmazhoz és minden objektumot megvizsgálva zárt HATCH-et vagy zárt de összefüggő HATCH-eket kell keresni.

# Alkalmazás tesztelése

## Modul szintű tesztelés

Minden létrehozott entitás külön tesztfájllal lett tesztelve. A teszt folyamatot debugger segítségével soronként is el tudom végezni. Továbbá az változók és az eredmények nyomon követhetőségé érdekében külön loglevel változót deklaráltam ami kezeli a túl részletes log adatok átadását a console felé. A teszteléshez szükséges logolást mind java mind JS oldalon elkélszült.

A tesztelés során főleg javascript szintaktikai és funkcionális hibát sikerült kiszűrni.

Például tanulságos eredményeket tud produkálni a JS-ben ha az if vizsgálat során a két változó közé csak egy egyenlőség jel kerül csak.

## EndtoEnd tesztelés

A testfile könyvtárban több ellenőrzött fájl van aminek a beolvasása rendben lezajlott.

# Irodalom jegyzék

<https://images.autodesk.com/adsk/files/autocad_2014_pdf_dxf_reference_enu.pdf>

<https://ezdxf.readthedocs.io/en/stable/dxfinternals/filestructure.html>

<https://www.loc.gov/preservation/digital/formats/fdd/fdd000446.shtml>

<https://damassets.autodesk.net/content/dam/autodesk/www/developer-network/platform-technologies/autocad-dxf-archive/acad_r12_dxf.pdf>

<https://docs.fileformat.com/cad/dxf/>

<https://www.inf.u-szeged.hu/~katona/gis.pdf>

SZTE

04\_DXF segédlet.pdf

dxf\_felepítes\_99old.pdf

<https://github.com/fuzziness/kabeja>

# Nyilatkozat

Alulírott Ujszászi János programtervező informatikus BSc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Szoftverfejlesztés Tanszékén készítettem, programtervező informatikus BSc diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy szakdolgozatomat a Szegedi Tudományegyetem Informatikai Intézet könyvtárában, a helyben olvasható könyvek között helyezik el.

2022. április. 10.

aláírás

# Köszönetnyilvánítás

Szeretném megköszönni a segítséget témavezetőmnek Tóth Zoltánnak.

Kollégáimnak Verner Gábornak, Gulyás Ferencnek és Dr. Ugron Balázsnak.

# Mellékletek és elektronikus melléklet

<https://github.com/JannY0927/iCalc_V0>