ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

# SZAKDOLGOZAT

OE-NIK
2023

Hallgató neve: Mészáros Máté
Hallgató törzskönyvi száma: T/006867/FI12904/N

Óbudai Egyetem
Neumann János Informatikai Kar
Szoftvertervezés és -fejlesztés Intézet

# SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Mészáros Máté**
Törzskönyvi száma: T/006867/FI12904/N
Neptun kódja: [redacted]

A dolgozat címe:

**Téroptimális bútorelrendező webalkalmazás 3D megjelenítéssel**
**Space-optimal furniture layout web application with 3D visualization**

Intézményi konzulens: Sipos Miklós László
Külső konzulens:

Beadási határidő: 2022. december 15.

A záróvizsga tárgyai: Számítógép architektúrák
Szoftvertervezés és –fejlesztés specializáció

**A feladat**

A feladat egy webalkalmazás készítése egy adott szoba bútorzatának kialakítására. A feladat megvalósításakor vizsgálja meg a hasonló fejlesztéseket, az adatok eltárolására alkalmas adatbázis rendszereket (mind a lokális, mind felhő alapú eszközöket) és az ilyen jellegű fejlesztésekhez rendelkezésre álló 3D renderelőket. Kerüljön kialakításra a backend és frontend szeparációja. Elemezze és értékelje az erre alkalmas programnyelveket. Az alkalmazás legyen képes bútorok lehelyezhetőségének és ezek pozícianolhatóságának biztosítására. Legyen lehetőség a pakolási kompozíciók elmentésére és visszatöltésére. Képes legyen a bútorok elhelyezkedését térkihasználás szempontjából optimálisan meghatározni. A kész kompozíciót a felhasználó élmények növelése érdekében lehessen háromdimenziós formában is megtekinteni. Vizsgálja meg a feladathoz legjobban illeszkedő technológiákat, amelyek alapján hozza meg a döntését, hogy melyekkel fog dolgozni az alternatívák közül. A fejlesztést verziókövetett formában valósítsa meg Git segítségével.

**A dolgozatnak tartalmaznia kell**:

- a megoldandó feladat pontos leírását és részletes elemzését,
- magas szintű szerkezeti mintát az alkalmazásról,
- az egyes funkciók megvalósításához felhasználható lehetséges technológiákat,
- az elkészített rendszer tervét, blokkdiagramját,
- a megvalósítás menetét, felmerülő problémákat és az azokra adott megoldást,
- a tesztelési tervet, a tesztelés módszertanát és a tesztelés eredményeit,
- az elkészült szoftver felhasználási lehetőségeit.

ÓBUDAI EGYETEM * Neumann János Informatikai Kar * Ph. 1. Budapest

.....
Dr. Vámossy Zoltán
intézetigazgató

A szakdolgozat elévülésének határideje: **2024. december 15.**
(OE TVSz 55.§ szerint)

A dolgozatot beadásra alkalmasnak tartom:

.....
külső konzulens

.....
intézményi konzulens

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

# HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat / diplomamunka saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatomban / diplomamunkámban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja.

Budapest, 20.22. 12. 13.

Mészáros Máté

hallgató aláírása

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

# KONZULTÁCIÓS NAPLÓ

Hallgató neve: MÉSZÁROS MÁTÉ

Neptun Kód: [redacted]

Tagozat: NAPPALI TAGOZAT

Telefon: [redacted]

Levelezési cím (pl.: lakcím): [redacted]

Szakdolgozat címe magyarul:

TÉROPTIMÁLIS BÚTORELRENDEZŐ WEBALKALMAZÁS 3D MEGJELENÍTÉSSEL

Szakdolgozat címe angolul:

SPACE-OPTIMAL FURNITURE LAYOUT WEB APPLICATION WITH 3D VISUALIZATION

Intézményi konzulens: SIPOS MIKLÓS LÁSZLÓ

Külső konzulens: -

| Alk. | Dátum | Tartalom | Aláírás |
|---|---|---|---|
| 1. | 2022.03.04. | **Félév teendőinek, menetrendjének és mérföldköveinek ismertetése.** | |
| 2. | 2022.04.01. | **Hasonló rendszerek elemzése.** | |
| 3. | 2022.04.15. | **Irodalomkutatás, annak összegzése valamint konzekvenciák meghatározása.** | |
| 4. | 2022.05.06. | **Tervezés, követelmények specifikációja. Véglegesítés. Leadással kapcsolatos teendők egyeztetése.** | |

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. tantárgy követelményét teljesítette, beszámolóra bocsátható.

.....

Intézményi konzulens

Budapest, 2022.05.10.

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

# KONZULTÁCIÓS NAPLÓ

Hallgató neve: MÉSZÁROS MÁTÉ

Neptun Kód: [redacted]

Tagozat: NAPPALI TAGOZAT

Telefon: [redacted]

Levelezési cím (pl.: lakcím): [redacted]

Szakdolgozat címe magyarul:

TÉROPTIMÁLIS BÚTORELRENDEZŐ WEBALKALMAZÁS 3D MEGJELENÍTÉSSEL

Szakdolgozat címe angolul:

SPACE-OPTIMAL FURNITURE LAYOUT WEB APPLICATION WITH 3D VISUALIZATION

Intézményi konzulens: SIPOS MIKLÓS LÁSZLÓ

Külső konzulens: -

| Alk. | Dátum | Tartalom | Aláírás |
|---|---|---|---|
| 1. | 2022.10.02. | **Fejlesztés tervezett menetének revíziója, szakmai egyeztetés, fejlesztési teendők megkezdése.** | |
| 2. | 2022.10.30. | **Fejlesztés állapotának és haladásának áttekintése. Technikai problémák és módosítások egyeztetése.** | |
| 3. | 2022.11.20. | **Felmerülő problémák egyeztetése és megoldása. Tesztelési fázis megindítása.** | |
| 4. | 2022.11.30. | **Dokumentáció véglegesítése. Leadással kapcsolatos teendők egyeztetése.** | |

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat II. tantárgy követelményét teljesítette, védésre bocsátható.

.....

Intézményi konzulens

Budapest, 2022.12.01.

# Tartalomjegyzék

# Absztarkt

Az alábbi dokumentum a bútorelrendező és háromdimenziós grafikát megvalósító alkalmazásokat vizsgálja webes környezetben. Hiszen tudjuk, hogy a webfejlesztés fénykorát éljük, egyre több és több hasonló fejlesztés lát napvilágot. Egyre magasabb szintű programozási nyelveknek és eszközöknek köszönhetően a megvalósítani kívánt cél elérése egyre zökkenőmentesebbé válik. Egyre jobb, egyre gyorsabb technikák jelennek meg, amikre érdemes figyelmet fordítani. Többek között ez fogalmazódik meg ebben a dokumentumban, mely az elemzés mellett egy lehetséges megvalósítást is szemléltet. A dokumentum mindenekelőtt elemzi és bemutatja a hasonló fejlesztések előnyeit és hátrányait. Hoz példát mérnöki és átlagfelhasználó számára is alkalmas háromdimenziós tervező alkalmazásokra. Vizsgálja az ehhez szükséges adatok tárolására alkalmas adatbázisrendszereket. Tárgyalja alapvető tulajdonságaikat, elemzi azokat, párhuzamot, vagy ellentétet állít közöttük, kitér erősségeikre és gyengeségeikre elemzi az aktuális trendeket. Kitér a háromdimenziós grafikára, bemutatja háttérfolyamatokat. Szemléltet webes és lokális grafikai megjelenítőket. Objektívan kiemeli előnyeiket és bemutatja hátrányaikat. Leírást ad a webfejlesztésben legtöbbet használt programozási nyelvekről. Az elemzés alapján a megvalósításhoz alkalmasnak vélt eszközöket felsorakoztatja. Tervezési mintát és rendszertervet ajánl a megvalósításhoz, melyekről leírást ad és az áttekinthetőség érdekében diagrammokkal szemléltet.

# Abstract

The following document examines applications that implement furniture layout and three-dimensional graphics in a web environment. After all, we know that we are living in the heyday of web development, more and more similar developments are coming to light. Thanks to ever-increasing levels of programming languages and tools, achieving the goal you want to achieve is becoming more and more seamless. There are better and faster techniques that are worth paying attention to. Among other things, this is stated in this document, which, in addition to the analysis, also illustrates a possible implementation. Above all, the document analyzes and presents the advantages and disadvantages of similar developments. It provides an example for three-dimensional design applications suitable for both engineering and average users. Examine the database systems that can store the data you need to do this. It discusses their basic qualities, analyzes them, draws parallels or contrasts between them, covers their strengths and weaknesses, and analyzes current trends. It covers three-dimensional graphics and shows background processes. Illustrates web and local graphics viewers. It objectively highlights their advantages and presents their disadvantages. Provides a description of the programming languages most commonly used in web development. Based on the analysis, it lists the tools deemed suitable for implementation. It offers a design template and system plan for implementation, which are described and illustrated with diagrams for clarity.

# 1. Bevezetés

Napjainkban bárhová is megyünk, bútorok vesznek minket körül. A komfortos lét elengedhetetlen tényezője a környezet, melybe a bútorok elrendezése nagy szerepet játszik. Sokkal jobban tudunk dolgozni, figyelni, ha azt szép, rendezett környezetben tehetjük meg. Mind ezek mellet érzéseinkre is behatással van. Sokkal jobb kedvünk lesz ha egy nehéz nap után szép környezetbe tudunk megérkezni. Gondoljunk bele, mennyivel jobban éreznénk magunkat a környezetünkbe, ha egyéni preferenciánk alapján tudnánk környezetünket berendezni, sok opciót kipróbálva. Sok munkahelyen épp ezért fordít fokozott figyelmet a munkahelyi környezet komfortos, tágas, rendezett kialakítására. Számos munkáltatónál elvárás, az irodai asztalok rendezetten tartása. Az alkalmazottaknak kötelessége rendet tartani maga után, ezzel is tudatosan növelve a környezet produktivitásra gyakorolt hatását.

Egyes bútordarabok áthelyezése nehéz és időigényes lehet súlyukból és méreteikből adódóan vagy egyszerűen csak az idő hiányába. Bútorok átrendezésénél szintén időigényes feladat az egyes darabok méretinek detektálása és az ezzel járó kalkulációk, hogy az adott tárgy be tehető-e a szoba egy adott pontjára. Emelet ott van a bútorok elhelyezhetőségének számtalan lehetősége, amit nem túl kecsegtető egyesével kipróbálni. A zsúfolt, kis helységek növelhetik a frusztrációt és csökkenthetik a munkára való fókuszálás képességét.

Ezen és hasonló problémák megoldását hivatott elvégezni az alábbiakban kifejtett szoftver.
Ugyanis az optimális elrendezés megtalálásával egy aprónak tűnő szoba esetében is kelthetünk tág érzetet.

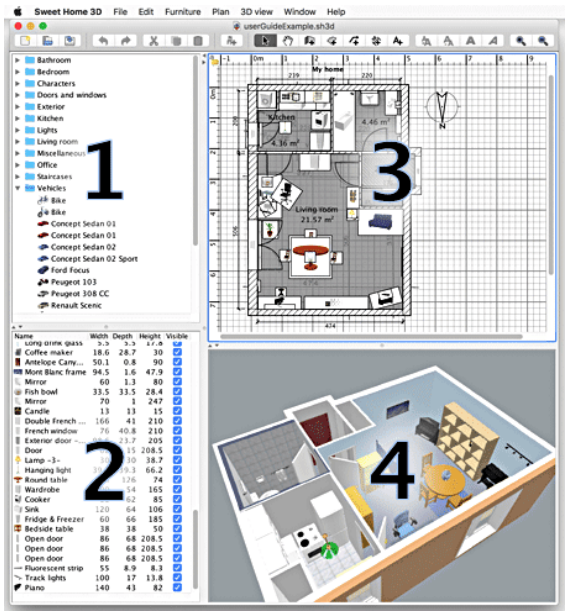
# 2. Irodalomkutatás

## 2.1 Hasonló fejlesztések áttekintése

Ehhez és hasonló problémakörre választ kereső szoftverekből számos létezik. Sőt olyannyira sok, hogy mindent bemutatni sem lehet, ezért az alábbiakban ezek közül a személyes preferenciáimnak leginkább és a projektmunkám szempontjából leghasznosabb alkalmazásokat vetem vizsgálat alá.

### 2.1.1. Sweet Home 3D

A Sweet Home 3D nevű számítógépes szoftvert a francia eTesk cég készítette 2005-ben. Viszonylag régi koncepció, viszont a mai napig frissítik, és nagyon sokan használják. Alapjába véve nem csak szobaspecifikusan, hanem egész házas berendezési kompozíció megtervezésére ad lehetőséget. Egy kétdimenziós tervezőfelület áll rendelkezésünkre, ahol a ház specifikációi után (Falak méreteinek megadása. Ajtók, ablakok pozícionálása.) kedvünkre rendezhetjük be



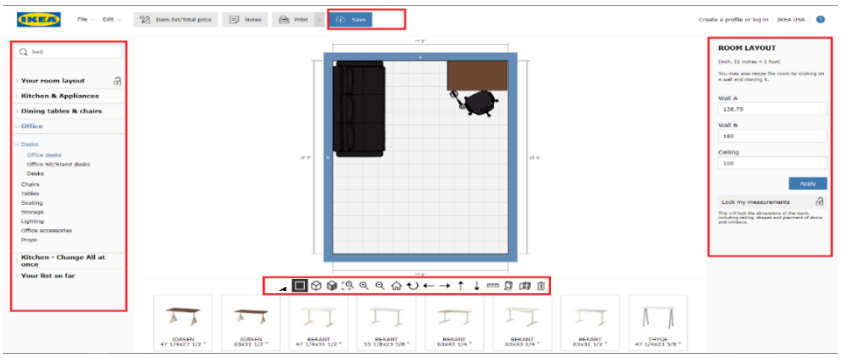

1.1. ábra. Sweet Home 3D kezelőfelület [24]

akár az egész házat, vagy csak szobákat. Ezt az 1.1-es ábra 1-es és 3-as számú területén látható. Lehetőség van előre definiált bútorokat kiválasztani, méretre szabni, elforgatni és azokat elhelyezni a kétdimenziós modellben. Ezt az 1.1-es ábra 2-es számmal jelölt területéről tehetjük meg. Az eredmény párhuzamosan lekövethető a képernyő másik felén háromdimenziós formában. Ez látható az 1.1-es ábra 4-essel jelölt területén. A falak és a bútorok átlátszósága állítható, így nem okoz problémát a makettet oldalról is szemlélni. A tárgyakat ki is lehet színezni, valamint, ha nincs betáplálva megfelelő lehelyezni kívánt objektum, azesetben azt az archive3d nevezetű weboldalról le lehet tölteni, de akár saját tárgy létrehozására is van lehetőség. A szoftver számos előnnyel rendelkezik. Kezelése könnyen elsajátítható a témában

laikusok számára is, viszont meglehetősen összetett tervezések megvalósítására is alkalmas, így hasznos tud lenni ezen a területen jártas szakemberek számára. Folyamatosan látjuk, a háromdimenziós modellt miközben tervezünk. Ez nagy előnyt jelenthet a vizuálisabb típusú embereknek. Sőt az elkészült ház megtekinthető belül nézetből és tetszés szerint bejárható. A testreszabhatósági lehetőségek száma határtalan. Tényleg bármit el lehet helyezni az épületben. Szélsőségesebb példákkal élve lehelyezhetünk akár egy pár cipőt, keresek széket, nyomtatókat, festményeket, farakást, játékautót, pókhálót, de még akár egy szendvicset is. A kreativitás határtalan, így a program alkalmas lehet egy ház minden kis apró részlet lemodellezésére. A ház külseje szintén teljesen testre szabható, a tetőszerkezet, a kocsibejáró, a fák a bokrok és még számos opció. A program egyértelműen több előnnyel rendelkezik, mint hátránnyal, viszont hátrányokból is akad néhány. A szoftver eredendően ingyenes viszont bizonyos funkciók elérése érdekében létezik fizetős verzió is, ez felhasználói szempontból nem előnyös. Egyelőre csak számítógépre elérhető, így aki nem rendelkezik ilyen eszközzel, az nem tudja használni a programot. Lehetnek problémák a háromdimenziós renderelő rendszerrel, bár a probléma számítógépekhez értők számára könnyen megoldható, alap felhasználó számára nem biztos, hogy triviális. Nagyobb volumenű projektek esetén nagyon magas a gépigény, amihez az átlagfelhasználói PC-k teljesítménybeli feltételei nem biztos, hogy adottak. Bár a szoftvert kezelni viszonylag egyszerű és könnyen tanulható, összetetteb tervezések esetén meglehetősen bonyolulttá és nehezen áttekinthetővé tud válni. Véleményem szerint hiányzik belőle egy „könnyített mód”, azoknak a felhasználóknak, akik nem akarnak bonyolultabb projektekbe belebonyolódni, csak kis volumenű szoba megtervezését szeretnék megvalósítani. Úgy gondolom, hogy a Sweet Home 3D, szoftverem elkészítésé során nagy segítséget tud nyújtani számomra és alapul tudok venni néhány megvalósítási technikák. Rendkívül jó ötletnek tartom az előre detektált bútorokat és azok átméretezhetőségét. Szintén tetszik, hogy a falakat a kurzorral tudjuk megrajzolni, szerintem ez sokkal felhasználóbarátabb kivitelezés, mint paraméterek megadásával. Jónak gondolom a kétdimenziós lehelyező felületet is. Ezeket a módszereket mindenképpen hasonlóképpen szeretném megvalósítani. [1] [2]

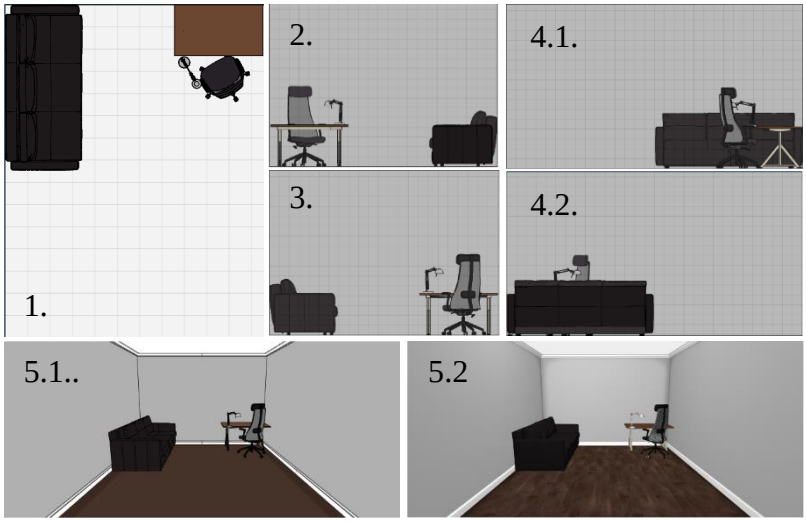
## 2.1.2. IKEA Home Planner

Bizonyára sokan ismerik a világ legnagyobb bútor áruházát, az IKEA-t. Az általuk fejlesztett, ingyenes, online szoftver az Ikea Home Planner. A program az előzővel ellentétben




1.3. Ikea Home Planner kezelő felület [26]

szobaspecifikus berendezésre ad lehetőséget. A kezelőfelületen öt különböző nézet között váltogathatunk, viszont egyszerre csak az egyiket jeleníthetjük meg. Van, egy kétdimenziós fölülnézet (1.2. ábra 1. kép), egy kétdimenziós elölnézet (1.2. ábra 2. kép), egy kétdimenziós hátulnézet (1.2. ábra 3. kép) ,egy kétdimenziós oldalnézet jobbról és balról (1.2. ábra 4.1. és



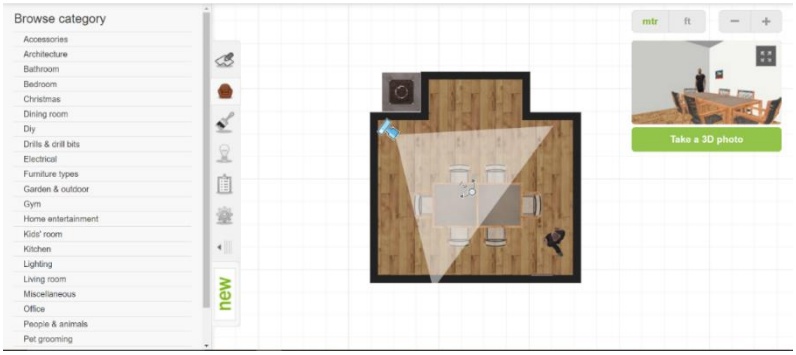

1.2. Ikea Home Planner nézetei [26]

4.2-es képek) és egy háromdimenziós elölnézet (1.2. ábra 5.2. kép) valamint egy háromdimenziós vonalas elölnézet (1.2. ábra 5.1. kép).
Mindegyik nézetet dedikált gombokkal tudjuk csak mozgatni. Föl, le jobbra, balra irányokba. Van egy alaprajz és egy átméretező ablak, ahol specifikálhatjuk a helység méreteit. A méretbeállítások végeztével egy legördülő menüsorból a tárgyakat a kurzor segítségével a megfelelő pozícióba helyezhetjük. A legördülő listában előre definiált tárgyak vannak, melyeknek méretén nem tudunk változtatni. A menüsorbeli tárgyak szobatípusok szerint vannak csoportosítva. Ha egy tárgyra kattintunk, akkor a kijelölt elemet tudjuk mozgatni, forgatni, törölni vagy egyes bútorok esetén plusz elemekkel ellátni. A kurzorral egy nagyobb területet kijelölve egyszerre akár több tárgyat is arrébb mozgathatunk. Az elkészült munkát pedig le tudjuk menteni a számítógépünkre. A program számos jó tulajdonságot tudhat magáénak. Felhasználó szempontból közelítve első pozitívum, hogy ingyenes. Szintén nagy előny, hogy online felület révén nem foglal memóriát, valamint rendkívül alacsony a gépigénye. Nagyon triviális kezelőfelület, már az első használatkor is tökéletesen kezelhető. Nagyszerű koncepció egyszerűbb szobák tervezésére. A falra kattintva és az egeret mozgatva is átméretezhetjük a szobát. A tárgyakat bármelyik nézőpontból mozgathatók. Ez nagy pozitívum, mivel több programnál is csak a felülnézetes bútormozgatásra van lehetőség. Ha oldal vagy fölülnézetben vagyunk a program folyamatosan megjeleníti mindegyik tárgyak méretét miniméter pontosan, így látjuk, hogy mennyi hely marad az adott tárgy és egy másik objektum között, ez nem tűnik nagy dolognak viszont a tervezésben sokat tud segíteni. Ez a funkció a legtöbb szoftverben csak a tárgyakra való kattintás után érhető el, viszont itt nagy pozitívum,

hogy egyből kijelzésre kerül. A program sok pozitívum mellet sajnos legalább annyi negatívummal is rendelkezik. Szerintem a legnagyobb probléma a többi szoftverhez képest, a nagyon kevés bútor elérhetőség és nagyon kötött átméretezhetőségi lehetőség. A korlátozott lehetőségek gyanánt csak új szobák berendezését lehet megvalósítani, nincs lehetőség a már meglévő bútoraink detektálására és virtuális újra rendezésére (Persze ez a bútorbolt specifikus szoftver mivoltijából adódik, viszont hátrány lehet a többi versenytársra nézve). A kétdimenziós oldalnézet elérése csak a kétdimenziós elölnézetből érehető el. Ez elég kényelmetlen módja az oldalnézet elérésének és sajnos nincs dedikált gomb. Nincs lehetőség kurzoros forgatásra. Szerintem ez felhasználói szempontból elég hátrányos és megnehezíti a szoftver kezelhetőségét. Nincs lehetőség a szoba területének megtekintésére. Ugyan a falak hosszát látjuk, viszont szerintem elvárható lenne, hogy ne a felhasználónak kelljen kiszámolnia. A szoba kurzoros szélesítésére és mélyítésére csak fölülnézetből van lehetőség, tehát oldal vagy háromdimenziós nézetből ezt nem tehetjük meg. Szintén kifogásolható kezelhetőségi szempontból, mivel vissza kell váltanunk először fölülnézetbe ahhoz, hogy a szoba paraméterein változtassunk. A belmagasság kurzorosan szintén csak oldalnézetből állítható. A program trivialitása hátrány is lehet szakemberek szempontjából, hiszen nem ad túl tág testreszabhatósági szabadságot, persze ennek a szoftvernek úgy gondolom, hogy nem is ők a célközönségei. Összeségében az egyik legjobb koncepciónak gondolom átlagfelhasználók számára. Azért esett a választásom ennek a szoftvernek az elemzésére mert a projektmunkámat én is szobaspecifikus szeretném megvalósítani és szintén főleg átlagfelhasználók számára. Tetszik a kezelőfelület egyszerűsége és a tervezési felület átláthatósága. Szintén tetszik a méretek folyamatos megjelenítése. Ezeket a megvalósításokat hasonlóan képzelem el a saját projektemben.

## 2.1.3. Roomstyler 3D Home Planner

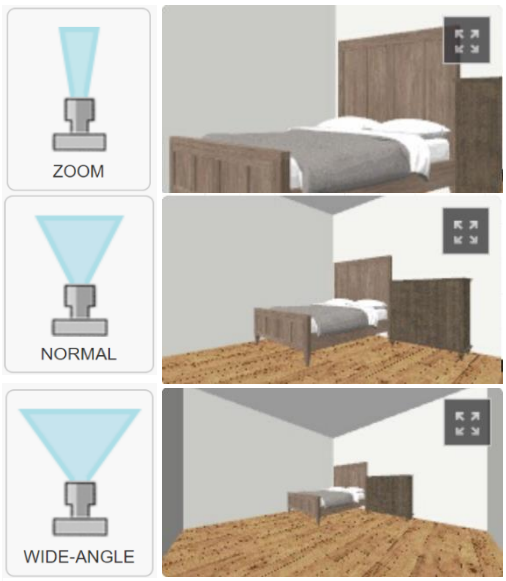
A Roomstyler 3D, egy online, ingyenes, szoftver, amit a Floorplanner készítői alkottak meg 2013-ban. Egy végtelenített kétdimenziós tervezőfelületre van lehetőségünk elemek lerakásár, egy bal oldali legördülő listából. A falak lerakására csak a kurzorral van lehetőség. A falakat




1.4. Roomstyler 3D Home Planner kezelőfelület [28]

lehelyezhetjük előre elkészített sablonok szerint is. A lehelyezhető elemek különböző kategóriák szerint vannak csoportosítva. A kiválasztott bútort bal egérgomb lenyomásával a

tervezőterületre rakható. A bútorra kattintva tudjuk mozgatni, forgatni vagy egy ugyan olyan elemet lerakni. A felület bal sarkában egy kijelző jelzi a lerakott falak, ajtók, ablakok, tárgyak és anyagok számát.
Ha elkészítettük a kétdimenziós modellt lehetőségünk van megtekinteni három dimenzióban is, viszont az előbbiektől rendkívül eltérő módon. A programban kamerákat helyezhetünk le a kétdimenziós felületen akár csak a többi tárgyat és a képernyő jobb felső sarkában jelenik meg a kamera által látott háromdimenziós kép. A kamerán három betekintési szög közül választhatunk, szűk betekintési szög ámde közeli kép (zoom), normál betekintési szög, normál képközelség (normal) és széles látószög távoli képközelség (wide-angle) ez látható az 1.5. ábrán. A kamerának állíthatunk még a döntési szögén is, hogy milyen szögből szeretnénk a



1.5. Kameratípusok szemléltetése [28]

szobát szemlélni. A kamerával fotót is készíthetünk, amit a rendeszer emailben küld el a felhasználónak. A lehelyezett objektumok kiszinezhetők, az anyagok cserélhetők. Be lehet állítani, hogy nappal vagy este legyen és ha nappal, akkor milyen erősen fényviszonyok legyenek és, hogy melyik égtáj felöl világítson a fényforrás. A szoftver számos jó tulajdonsággal rendelkezik. Felhasználói szempontból nézve előny, hogy teljesen ingyenes. A kezelőfelület rendkívül egyszerű, átlátható. Rengeteg tárgy közül választhatunk és a program különlegessége, hogy élőlényket is lehelyezhetünk. A jó tulajdonságokhoz képest sajnos több rosszat lehet a program mellett fölsorakoztatni. Rossz kameraminőség, a formák nehezen kivehetők. Szerintem sokkal kényelmetlenebb és időigényesebb az elkészült munka megtekintése, a kameralehelyezésen alapuló háromdimenziós modell legenerálásával. Rossz grafika (Betudható annak, hogy ingyenes szoftver). A program nem foglalkozik a tárgyak ütközésével, simán egymásra pakolhatók, vagy falon átlógathatók. Ez nagy probléma mivel szemmérték alapján tudjuk csak megállapítani, hogy elfog-e férni egy bútor az adott helyen vagy sem. Nem lehet benne a tárgyakat méretezni. Nem látunk semmilyen méretbeli adatot,

sem bútorok nagysága, sem falak hossz, sem terület így nincs viszonyítási alapunk. Szakemberek számára egyáltalán nem alkalmas komolyabb architektúrák megtervezésére, a célközönség inkább az átlag felhasználók. Összeségében, tehát a Roomstyler 3D Home Planner egy gyors berendezés összeállítására alkalmas, viszont komolyabb tervezések esetén kényelmetlen és közel sem lehet benne olyan pontossággal tervezni, mint a fentebb említésre került szoftverekben. Ami megtetszett és alkalmazni tudok a projektemben az a program letisztultsága és egyszerűsége, valamint a praktikus megoldások.

## 2.1.4. Összefoglalás

Az alábbi táblázat összefoglalja az imént tárgyalt, általam valamilyen szempontból kiemelkedőnek vélt szoftvereket. Ezen felül a táblázatban említésre, összehasonlításra kerül egyéb, ismert szoftver. Ezen megemlített szoftverek kipróbálására sajnos nem volt lehetőségem ezért, a hivatalos oldalukon megtalálható videók, leírások, demóverzió, ingyenes verzió alapján kerülnek megállapításra. Fontos megjegyezni, hogy a táblázat átlagos felhasználói oldalról megközelítve került kialakításra.

| | Sweet Home 3D | Ikea Home Planner | Roomstyler 3D | Floorplanner | SketchUp Pro | AutoCAD |
|---|---|---|---|---|---|---|
| **ingyenes** | ✓ | ✓ | ✓ | X | X | X |
| **online** | X | ✓ | ✓ | ✓ | ✓ | X |
| **van asztali alkalmazás** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **van androidos alkalmazás** | X | X | ✓ | X | X | X |
| **gépigény** | közepes | alacsony | alacsony | közepes | közepes | magas |
| **grafika** | jó | jó | alacsony | jó | jó | magas |
| **testreszabhatóság** | magas | alacsony | magas | magas | magas | magas |
| **kezelhetőség** | átlagos | átlagos | könnyű | könnyű | nehéz | nehéz |
| **specifikusság** | teljes ház | szoba | ház | ház | „minden”[1] | „minden” [1] |
| **mérnöki munkához alkalmas** | ✓ | X | X | X | ✓+[2] | ✓+[2] |
| **átlagfelhasználók számára alkalmas** | ✓ | ✓ | ✓ | ✓ | X | X-[3] |

1.1. Összefoglaló táblázat

## 2.1.5. Előrelépés

Az általam fejlesztett program a következőben lépne előre a fentebb említett alkalmazásokhoz képest.

Magába foglalja egyes fizetős szoftverek legtöbb funkcióját, mindezt alacsony számítás kapacitással, díjmentesen és webes környezetbe.

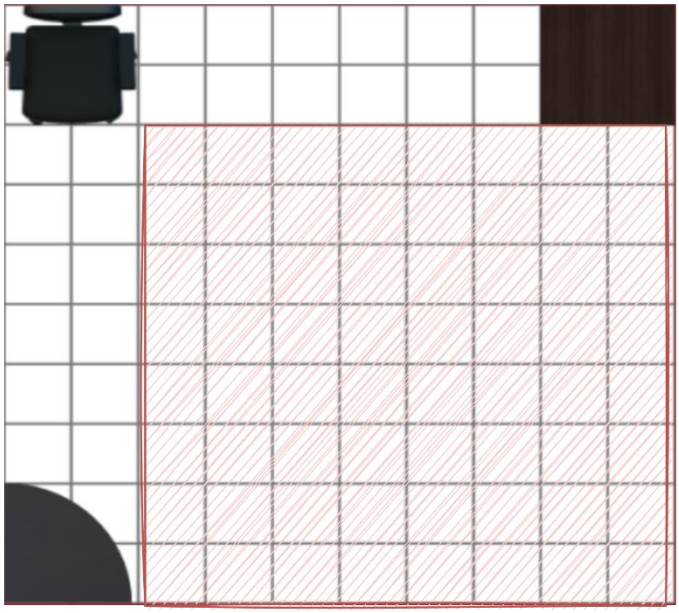
Tartalmaz egy optimális bútorelrendező funkciót. Optimális elrendezés alatt azt értem, hogy a program képes összeállítani a szoba egy olyan bútorelrendezését, amelyben megtalálható a

[1] Nem csak lakóházak tervezésére alkalmas, hanem bármilyen tárgy megtervezhető a segítségével.

[2] A ✓+ jelölés arra vonatkozik, hogy ezek a szoftverek szinte teljesen mérnöki munkára lettek kifejlesztve.

[3] Az X- jelölés arra vonatkozik, hogy egyáltalán nem ajánlott átlagos felhasználók számára.

lehető legnagyobb egybefüggő tér. Egybefüggő tér alatt az elérhető tér legnagyobb téglalap alakú rézhalmazát értem. Tehát a szoftver visszaadja egy olyan pakolását a bútoroknak melyben az elérhető téglalap alakú rézhalmaz területe a lehető legnagyobb. Az alábbi ábrán egy ilyen összeállításra látható egy példa, melyen az elérhető legnagyobb téglalap alakú rézhalmazt a piros kiemelés jelöl.



2.1 Maximális téglalap alakú terület szemléltetése

# 2.2 Használt módszerek, technikák

## 2.2.1. Adatbázis rendszerek

Az adatok tárolásának igénye már a számítástechnika korai szakaszában fellépet. Az adatbázis adatok és információk gyűjteménye, amelyet szervezett stílusban tárolnak és állítanak be a könnyű hozzáférés, kezelés és frissítés érdekében. Az adatbázisban lévő adatok hozzáadhatók, törölhetők vagy módosíthatók.

Elsők adatbázis típusként a relációs adatbázisok jelentek meg. Az 1970-es évektől kezdve ez az megoldás volt az adatbázis rendszerek éllovasa és csak a közelmúltban jelentek meg más alternatívák az adatok tárolására.
Ezek az adatbázisok, még egy szerveren tárolódnak, valamint skálázhatósági korlátokba ütköznek, ez azt jelenti, hogy az adatmennyiség növekedése gyorsan romló teljesítményhez vezet. A karbantartásuk és konfigurálásuk meglehetősen drága. A mezőhossznak vannak korlátjai, ami megnehezíti nagy mennyiségű információ egyetlen mezőben való tárolását. Mivel az adatokat táblázatok formájában tárolják, az adatokat nem lehet könnyen becsomagolni. Összetett relációs adatbázis-rendszerek esetén nehézkessé válik az információ megosztása egyik adatbázisból a másikba. Az SQL nem képes hatékonyan kezelni a strukturálatlan

adatokat. Ezek olyan adatok, amelyekben általában nem lehet könnyen keresni, ezek az adatok nyersek és rendezetlenek, és alig vagy egyáltalán nem rendelkeznek előre meghatározott szerkezettel vagy formájukkal. Ilyenek például a multimédia, az e-mailek, dokumentumok és közösségi médiák. A megosztott relációs adatbázissal végzett munka nehézségeket okozhat a teljesítmény és az adatintegritás szempontjából. Nehéz megjósolni az adatbázis teljesítménykészségét, mert minden alkalmazás más módon fér hozzá az adatbázishoz. Mindeközben nehéz biztosítani az adatok integritását, mert egyetlen alkalmazás sem rendelkezik az adatok felett, miközben az alkalmazások eltérő üzleti elvek szerint működhetnek.
A relációs modellt ért kritikák ellenére napjainkig nagyon gyakran használt adatbázis modell, ami nem véletlen hisz negatívumai mellet számos erősséggel rendelkezik. Rugalmasság, egyszerűség, könnyű adat visszakeresés, adatintegritás, absztrakció, többfelhasználós hozzáférés és automatikus optimalizálás a kereséshez. Jól és gyorsan kezeli a strukturált adatokat, melyek olyan adatok, amelyek világosan leírt adattípusokból állnak, és amelyeket előre meghatároztak és formáztak egy meghatározott struktúrára, mielőtt adattárolásra kerülnének. Támogatja az ACID korlátozásokat (atomitás, konzisztencia, elszigeteltség, tartósság), e négy minőség mindegyike garantálja a stabilitást, a biztonságot, és hozzájárul a tranzakciók adatintegritásának biztosításához. [3]

Az olyan nagyvállalatok, mint az Amazon, a Google, a Facebook és a LinkedIn felfedték a relációs adatbázis korlátait. A relációs adatbázis-modell ezen hiányosságai és korlátjai a nem relációs adatbázisok fejlődéséhez vezettek, amelyeket NoSQL-nek (Not Only SQL) is neveznek Így jelentek meg az első NoSql adatbázis modell, melyek elsősorban skálázhatóság problémáját hivatottak megoldani. A közelmúltban a NoSQL a hagyományos relációs adatbázis-megközelítés ígéretes alternatívájának tekinthető.
A nem relációs adatbázisok számos előnyel/újdonsággal kecsegtetnek relációs elődjeikhez képest. Először is nem tárolják az adatokat relációs formátumban, és általában nem használnak SQL-t lekérdezési nyelvként. A NoSQL sokféle adatmodellt képes kezelni, beleértve az objektumokat, grafikonokat, oszlopokat, dokumentumokat és sok más formátumot. A relációs adatbázisokkal ellentétben hatékonyan képes kezelni a strukturálatlan és félig strukturált adatokat. Ezek már valóban dinamikusan skálázható adatbázisok, az adatok több szerveren tárolódnak, tehát horizontálisan méretezhetők. Ez azt jelenti, hogy a rendszer méretét lehetőség van befolyásolni további kiszolgáló egységek integrálásával. Tehát nem a külön álló kiszolgálókat erősítjük hardwaresen, hanem további szerverek hozzáadásával növeljük a rendszer terhelhetőségét. Lássuk be, hogy a horizontális átméretezhetőségnek, ilyen értelemben nincsenek korlátai, míg ezzel ellentétben vertikálisan fizikai korlátokba ütközünk. Fokozottan támogatja az adatreplikációt, így biztosítva a magas rendelkezésre állást. Könnyű és egyszerű API-kat biztosít az adatgyűjtéshez, valamint számos egyéb szolgáltatást, melyek megkönnyítik a programozást. Az ACID elvek helyett a BASE elveket követik, melyek a Basically Available (alapvetően elérhető), Soft State (lágy állapotú) és Eventual Consistency (végső következetesség) rövidítése. Bővebben kifejtve az alapvetően elérhető azt jelenti, hogy az

alkalmazásnak alapvetően mindig működnie kell. A lágy állapotú arra utal, hogy a teljes konzisztencia nem állandó elvárás. Valamint a végső következetesség, miszerint az alkalmazásnak végső soron valamilyen ismert állapotban kell lennie. A legtöbb NoSQL adatbázis nyílt forráskódú ingyenesen letölthető. Az összetett és strukturálatlan adatok hatékony kezelése, az egyszerű méretezhetőség mellett, a NoSQL adatbázisok elsődleges előnyei. Ha elosztott környezetekről beszélünk, ahol az adat mérete jelentősen nagy a NoSQL adatbázis modell a legmegfelelőbb.
A felhasználó igények kielégítésére különféle NoSQL adatbázis kategória és alkategória jött létre, melyek közül a leggyakrabban használt fölosztások a Key-Value Store Database (kulcs-érték pár adatbázis), Document Store Database (dokumentum tároló adatbázis) Column Family Stores Database (oszlopcsalád tároló adatbázis) és a Graph Database (gráf adatbázis).
A Key-Value Store Database az egyik legegyszerűbb adatbázis a típusok közül viszont emellett hatékony és erőteljes modell. Az adatok kulcs-érték párok segítségével kerülnek eltárolásra, ahol a kulcs egy egyedi azonosító, ami egy értékre mutat. Egyszerű alkalmazásprogramozási felülettel (API) rendelkezik, és lehetővé teszi az adatok sémamentes tárolását. Az adatok bármilyen primitív típusúak vagy objektumok lehetnek, és a tárolók strukturált és strukturálatlan adatokat is tárolhatnak. A kulcs-érték tárolók magas párhuzamosságot, jobb teljesítményt és nagy lekérdezési, feldolgozási sebességet és nagy méretű tárolást tesznek lehetővé.
A Document Store Database, olyan adatbázisok, amelyek az adatokat dokumentumok formájában tárolja, amelyek két fő komponensből állnak, egy kulcsból és egy dokumentumból. Ezek a dokumentumok valamilyen szinten hasonlítanak a relációs adatbázisok rekordjaihoz, de rugalmasabbak és sémamentesek, tehát a dokumentum szerkezetét nem kell megadni a felhasználónak mielőtt adatokat vinne be az adatbázisba. A dokumentum szerkezetére vonatkozó szabályokat általában beépített ellenőrzőrendszer végzi. A dokumentumok általános formátumai az XML, PDF, JSON, amik egyedi kulcsokat használnak, melyek általában egyszerű karakterlánc vagy URI vagy elérési út. A JavaScript Object Notation (JSON) a dokumentum tároló adatbázisban használatos dinamikus sémával, ami azt jelenti, hogy a különböző dokumentumok eltérő számú mezőt tartalmazhatnak, amelyeket egyedi kulccsal lehet hozzáadni. Ezek az adattárak valamivel összetettebbek, mint a kulcsfontosságú értékek adattárolói, de nagyszerű teljesítményt nyújtanak. A dokumentum adatbázisok általában olyan alkalmazásokhoz használatosak, amelyekben az adatokat nem kell egységes szerkezetű táblázatban tárolni, hanem minden dokumentum azonos vagy eltérő tulajdonságokkal rendelkezhet. A legnébeszerűbb Document Store adatbázisok a MongoDB, CouchDB Amazon DynamoDB, MongoDB Atlas és Google Cloud Firestore.
A Column Family Stores Database, oszloporientált táblákban tárolják az adatokat, ellentétben a relációs adatbázisokkal, amelyek sororientált stílusban tárolják az adatokat. Ezek az adatbázisok oszlopcsaládokban tárolják az adatokat sorokként, amelyekben sok oszlop van egy sorkulcshoz társítva. Egy sorkulcs, amely egy egyedi értékű rekord azonosítására szolgál, és ugyanazt a szerepet tölti be, mint egy reláció elsődleges kulcsa, ezen kulcsok használata rendkívül gyors keresést tesz lehetővé az adatbázisban. Fő adattárolás béli előnyei a

méretezhetőség, rugalmasság, olvasás gyorsasága a lekérdezés közben, könnyű tömörítés, mivel az egy fájlban lévő adatok azonos típusúak, új oszlopok hozzáadásának egyszerűsége. Az oszlopos tároló adatbázisok ideális adattárolási módjai alkalmassá teszik adatbányászati és analitikai alkalmazásokra, mint például a Google Bigtable, Cassandra és HBase.
A Graph Database, ahogy a nevéből is adódik gráfelméleti koncepciót valósít meg, mely sémamentesen tárolja az adatokat gráfok formájában. A gráf adatbázis csomópontok és élek összessége, ahol a csomópontok entitások vagy objektumok, az élek pedig a csomópontok közti kapcsolatok. A gráf adatbázisok hatékony adattárolást biztosítanak kifejezetten a félig strukturált adatok számára. A lekérdezések bejárásként történő kifejezése gráf adatbázisokban, ami gyorsabb, mint a relációs adatbázisok. A gráf adatbázisok számos alkalmazáshoz használhatók, például közösségi hálózati alkalmazásokhoz, bioinformatikához, biztonsághoz és hozzáférés-vezérléshez, hálózat és felhőkezeléshez. A leggyakrabban használt gráf adatbázisok a Neo4j, a FlockDB, az InfroGrid és az IMS.
A NoSQL adatbázisok rengeteg előnye és forradalmi újításai mellett hátrányai is számba vételezhető. Az egyik ilyen, hogy a magas rendelkezésre állás és teljesítmény a konzisztencia kárára valósul csak meg. Ez azt jelenti, hogy a NoSQL adatbázisok kevésbé megbízhatóak, mint a relációs adatbázisok, mivel veszélyeztetik a teljesítmény megbízhatóságát. Emellett nehezen karbantartható. Komoly gondot okot az ilyen típusú adatbázisokban az adatfájlok titkosításának hiánya. [3]

Az adatok méretének exponenciális növekedése miatt lépett felszínre a 2000-es évek végén a Big Data, mely már alkalmas olyan mértékű adatok eltárolására, melyre az eddigi technológiákkal nem lehet megvalósítani. Napjainkban feltörekvő piaci ág a felhőszolgáltatások. Bérlési költséget ellenében, van lehetőség tárhely bérlésére. Egyre több cég tárolja adataikat felhőben, ugyanis így az adatbázis karbantartásával nem kell foglalkozni.

| Rank | | | DBMS | Database Model | Score | | |
|---|---|---|---|---|---|---|---|
| Apr 2022 | Mar 2022 | Apr 2021 | | | Apr 2022 | Mar 2022 | Apr 2021 |
| 1. | 1. | 1. | Oracle | Relational, Multi-model | 1254.82 | +3.50 | -20.10 |
| 2. | 2. | 2. | MySQL | Relational, Multi-model | 1204.16 | +5.93 | -16.53 |
| 3. | 3. | 3. | Microsoft SQL Server | Relational, Multi-model | 938.46 | +4.67 | -69.51 |
| 4. | 4. | 4. | PostgreSQL | Relational, Multi-model | 614.46 | -2.47 | +60.94 |
| 5. | 5. | 5. | MongoDB | Document, Multi-model | 483.38 | -2.28 | +13.41 |
| 6. | 6. | ↑ 7. | Redis | Key-value, Multi-model | 177.61 | +0.85 | +21.72 |
| 7. | ↑ 8. | ↑ 8. | Elasticsearch | Search engine, Multi-model | 160.83 | +0.89 | +8.66 |
| 8. | ↓ 7. | ↓ 6. | IBM Db2 | Relational, Multi-model | 160.46 | -1.69 | +2.68 |
| 9. | 9. | ↑ 10. | Microsoft Access | Relational | 142.78 | +7.36 | +26.06 |
| 10. | 10. | ↓ 9. | SQLite | Relational | 132.80 | +0.62 | +7.74 |

2.2. DB-Engines szerinti legnébszerűbb DB-k [21]

Manapság rengeteg adatbázis rendszer létezik. Az alábbi fejezetben a legyakrabban használt relációs és NoSQL adatbázis rendszerek kerülnek bemutatásra/ tárgyalásra.

### 2.2.1.2 Oracle Database

Az Oracle jelenleg a világ piacvezető relációs adatbázis rendszer. Az Oracle adatbázis egy egységként kezelt adatok gyűjteménye. „Az adatbázis célja a kapcsolódó információk tárolása

és visszakeresése. Az adatbázis-szerver az információkezelési problémák megoldásának kulcsa. Általánosságban elmondható, hogy egy szerver megbízhatóan kezel nagy mennyiségű adatot egy többfelhasználós környezetben, így sok felhasználó egyszerre érheti el ugyanazokat az adatokat. Mindezt nagy teljesítmény mellett érik el. Az adatbázis-kiszolgáló megakadályozza az illetéktelen hozzáférést, és hatékony megoldásokat kínál a hibák helyreállítására. Az Oracle Database az első olyan adatbázis, amelyet vállalati számítástechnikai célokra terveztek, és ez a legrugalmasabb és legköltséghatékonyabb módja az információk és alkalmazások kezelésének. A vállalati számítástechnika iparági szabványnak megfelelő, moduláris tárolók és szerverek nagy készleteit hozza létre. Ezzel az architektúrával minden új rendszer gyorsan kiépíthető az összetevők készletéből. Nincs szükség csúcsterhelésre, mert a kapacitás igény szerint egyszerűen hozzáadható vagy átcsoportosítható az erőforráskészletekből. Az adatbázis logikai és fizikai struktúrákkal rendelkezik. Mivel a fizikai és a logikai struktúrák elkülönülnek, az adatok fizikai tárolása a logikai tárolóstruktúrákhoz való hozzáférés befolyásolása nélkül kezelhető. ” [4]

#### 2.2.1.2 Microsoft Sql Server

A Microsoft SQL Server egy relációs adatbázis rendszer, melyet a Microsoft feljesztett ki. A lekérdezési nyelve a Transact-SQL, mely a Sysbase és a Microsoft együttműködésével született meg. A Microsoft SQL Server napjaink egyik leggyakrabban alkalmazott adatbázis rendszere, melyre abszolút rászolgált a rendkívül stabil biztonsági rendszere, gyors adatkezelése, optimális memóriahasználatával. [5]

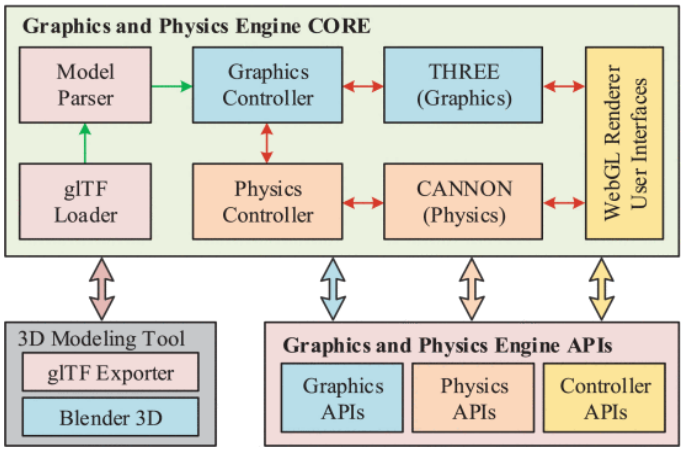
#### 2.2.1.4 Mongo DB

A Mongo DB, egy NoSQL, ezen belül is a Document Store adatbázis család egyik legismertebb tagja. A Mongo DB egy nyíl forráskodú adatbázisrendszer, melyet a 10gen nevezetű cég hozott felszínre 2009-ben. A Document Store adatbázis tárolási megvalósítások közül a Mongo DB JSON formátumú adatstruktúrát használ az adatok rendszerezésére. [6] Megvalósítja a NoSQL adatbázisok előnyeit, tehát jól skálázódik, képes strukturálatlan, félig strukturált adatok eltárolására. Több milliónyi adat kezelésére képes. Támogatja az indexelést tömbön és beágyazott objektumon keresztül. Saját lekérdezési nyelvel rendelkezik, a Mongo Query Language-el. Úttörőnek számít a biztonsági funkciók behozatalával, melyek az akkori NoSQL adatbázisokból hiányoztak. A Mongo BD egyre nagyobb népszerűségnek örvend vállalatok tekintetében, mert hatalmas mennyiségű big data eltárolására is képes. [7]

### 2.2.2. Számítógépes grafika

Számítógépes grafikákkal már az 1970-es évektől foglalkoztak. A számítógépes grafika (Computer Grapics) az informatika egyik rendkívül fejlett szakága, melyet számos helyen alkalmaznak. Napjainkban a vizualizációs technológia 2D-ről 3D-re változik. A számítógépes grafika jelenleg lenébszerűbb területei a VR (Virtual Reality), az AR (Actual Reality) és a Modellezés. A számítógépes grafika iránt egyre nagyobb érdeklődés figyelhető a szélesedő

ipari körökben. Legfőképpen játékokban, marketingbe, művészetekben, filmiparba, ipari vizualizációs, elemzési, tervezési és vezérlő alkalmazásokban és még számos egyéb területen. A számítógépes grafikai technológiák folyamatosan, exponenciális léptékben fejlődnek. A számítógépes grafika feladata a képadatok hatékony megjelenítésé (renderelése) a felhasználó számára. Manapság sok kapcsolódó 3D-s alkalmazás fut számos platformon, beleértve a webböngészőket is. Mivel a webalapú alkalmazások minden platformon futtathatók, beleértve a mobileszközöket is, ezért a webes 3D megjelenítés egyre nagyobb hírnévnek örvend. Ezek a megjelenítők gépközeliség tekintetében magas szinten helyezkednek el és általában az adott operációs rendszertől függő valamilyen alacsonyabb grafikai motorra fordulnak le. Például Windowson Vulcan, DirectX vagy OpenGL.
Az alábbiakban a grafikus megjelenítők egy egyszerű lehetséges architectúrális felépítése kerül rövid részletezésre. A grafikus megjelenető motorok alapvetően két részből állnak. A Grapics and Physics Engine CORE-ból (Engine Core) és a Grapics and Physics Application Programming Interfaces (Engine API-k). Tehát jól látható a [2.3] ábrán, hogy a WebGL-ek az Engine Core helyezkedik el, illetve, hogy az alacsonyabb szintű OpenGL API az Engine API rétegben foglal helyet. Mindegyik WebGL-t JavaScript segítségével ültetik be, ezért közvetlenül végrehajthatók webböngészőn. Illetve helyet foglal még a komponensek között a 3D Modelling Tool, mely a glTF állományok exportálását végzi más alkalmazásokba, ilyen




2.3 Rendszer architectúra [8]

Modelling Tool a sokak által ismert Blender.
Az Engine Core az ábrán látható hét darab részegységből tevődik össze. Az első ilyen részegység a **glTF Loader**, melynek feladata a nevéből is adódó glTF kiterjesztésű modellek

| glTF Format | File Extension | File Format |
|---|---|---|
| glTF Separate | .gltf, .bin, .imgType [a] | JSON, Binary, imgType [a] |
| glTF Embedded | .gltf | JSON |
| glTF Binary | .glb | Binary |

2.4 glTF fájlformátumok [8]

hatékony betöltése. Hatékonyan képes minimalizálni mind a modellek méretét, mind a feldolgozási időt. Valamint neki köszönhetően a modellek glTF formátumba exportálhatók és menthetők. Az aszinkron letöltés elérése érdekében a glTF Loader az úgynevezett JavaScript Promise segítségével valósul meg. A betöltés után egy konverzió valósul meg, mely során a betöltött adatok a modell típusától függően a grafikus vezérlő számára értelmezhető, szabványos formátumba kerül. Ezt a feladatot hivatott ellátni a rendszer Model Parser komponense. A **Model Parser** három különböző formátumot támogat. Az első a glTF Separate (.gltf + .bin + textúra), ez a glTF JSON béli fájlformátuma. A textúra lehet egy szabványos formátumú képfájl, .JPG vagy .PNG. Második a glTF Embedded (.gltf). Ez a formátum egyetlen JSON formátumú fájlba egyesíti az összes információt, beleértve a jelenetben lévő elem textúráit (BASE-64 formátumba konvertálják, és beágyazzák a fájlba). A harmadik a glTF Binary (.glb). Ez a fájltípus egy tömörített bináris fájlformátum. Ez biztosítja a legkisebb fájlméretet és a leggyorsabb kicsomagolási/dekódolási feldolgozási időt. Ezt akkor lehet célszerű használni, ha modellünk olyan típusú melynek textúrái nem megosztottak
A harmadik modul a **Graphics Controller**, melynek feladata a garfikus könyvtárhoz való összeköttetés megvalósítása. Ez a modul felelős a hibák kiszűréséért és a függvényhívási optimalizációkért. A következő elem a **Physics Controller**, mely feladata klasszifikálja és betölti a testek különböző típusait. **Renderer and User Interface**, a modul a webböngészők szabványos DOM alapul. Kezeli a webtörzs és a <canvas> elemeit a WebGL-megjelenítéshez. Számos egér és billentyűzet esemény támogatott. A Canvas és Three modulokat a későbbiekben részletezném. [8]
A továbbiakban részletezésre kerül néhány korábban már említett web és nem web alapú grafikai motor.

### 2.2.2.1 OpenGL

Az OpenGL grafikai rendszer, egy szoftver interfész (API) a grafikus hardverekhez. Habár léteznek nála újabb fejlesztések, a mainapig rengetegen használják és nagy úttörőnek tekinthető a grafikai rendszerek tekintetében.
Az API több száz eljárásból áll és funkciók, amelyek lehetővé teszik a programozó számára, hogy meghatározza a shadereket, objektumokat és kiváló minőségű grafikus képek. Sok OpenGL hívás vezérli a geometriai objektumok rajzolását, például pontokat, vonalakat és sokszögek. A rajz egyes részeinél (például élsimításkor vagy több mintavétel van használatban) támaszkodik keretpuffer meglétére és tulajdonságaira. Egyes parancsok kifejezetten a framebuffert kezelik.

A programozó számára az OpenGL olyan parancsok halmaza, amelyek lehetővé teszik a specifikációt shader programokat vagy shadereket, a shaderek által használt adatokat és az állapotvezérlő szempontokat. Az OpenGL kívül esik a shaderek hatókörén. Az adatok jellemzően a geometriát két részre osztják vagy három dimenziót és textúra képeket, míg a shaderek szabályozzák a geometriát a geometria feldolgozása, raszterezése és a töredékek

megvilágítása, árnyékolása raszterezéssel jön létre, aminek eredményeként a geometria a framebufferbe kerül. Egy tipikus OpenGL-t használó program az ablak megnyitására irányuló hívásokkal kezdődik a framebuffer, amelybe a program rajzolni fog. Ezután hívások indulnak a kiosztáshoz egy OpenGL-környezetet, és társítsa az ablakhoz. Ha egy kontextus ki van osztva, Az árnyékolók, geometria és textúrák meghatározásához OpenGL-parancsok készülnek, amelyeket követnek parancsokkal, amelyek geometriát rajzolnak úgy, hogy a geometria meghatározott részeit átadják a shadereknek. A rajzparancsok egyszerű geometriai objektumokat adnak meg, mint pl pontok, vonalszakaszok és sokszögek, amelyeket a shaderekkel tovább lehet manipulálni. Vannak olyan parancsok is, amelyek közvetlenül vezérlik a framebuffert a és az olvasással pixel írása.

A megjelenítő motor lehetővé teszi az interaktív háromdimenziós programok megalkotását. Az OpenGL számos programozási nyelvbe alkalmazzák. Legyakoribban C++-ban, C-ben, Java-ba, C#-ba, de más nyelvekbe is lehetőség van integrálni. Az OpenGL egyik nagy előnye, hogy mindegyik platformon remekül funkcionál. A rendszer alkalmas lehet bonyolultabb játékok, alkalmazások fejlesztéséhez, de kisebb programok grafikus felületének megalkotására is használható. Értelemszerű WeGL-ek szembeni sebesség különbség az OpenGL javára, hiszen itt megspórolunk egy absztrakciós szintnyi fordítási időt. Sokkal jobban testre szabható, optimalizálható a renderelés egy webessel szemben. Szintén előny, hogy nagyon részletes dokumentációval rendelkezik, ez megkönnyíti a felmerülő problémák utánajárását. Kiemelkedő renderelési teljesítményre képes, ha a programunkban a teljesítményorientáltság az elsődleges szempont. Lényegesen kevesebb overheadet generál a többi API-hoz képest, emelet magas a funkcionalitása és viszonylag magas grafikai megvalósításra képes. Erősségei közé sorolandó a rendszer stabilitása, nagyon stabil, kiforrott, jól bevált grafikai rendszer. Az előnyeiből következik a hátránya is, hiszen sokkal bonyolultabb a kód szintaxis, mint egy WebGL-nél, sokkal komplexebb, több odafigyelést, háttér tudást igényel. [9]

#### 2.2.2.2 Cannon.Js

„A Cannon.js egy merev testszimulációs könyvtár, amelyet például játékokban használnak. A programozók felhasználhatják játékobjektumaik valósághű mozgását és interakcióját. A Cannon.js JavaScript nyelven íródott, és bármely böngészővel használható renderelő- vagy játékmotorral együtt használható.” írja a gyártó. [10]
A Cannon, tehát egy nyílt forráskódú API, mely jól használható WebGL alapú alkalmazásokhoz. Más alternatívokhoz képest kompaktabb, teljesítményét tekintve erősebbnek és könnyebben érthető, tanulható. Támogatja a 3D alkalmazásokhoz szükséges összes geometriát, például síkot, gömböt, téglatestet, hengert, részecskét és magassági mezőt.

#### 2.2.2.3 Three.js

A Three.js egy JavaScript könyvtár és API, amelyet animált 3D számítógépes grafika létrehozására és megjelenítésére használnak WebGL segítségével. A Three.js a JavaScript 3D könyvtárak és keretrendszerek legpopulárisabb tagja, melyet Ricardo Cabello fejlesztett ki 2010-ben. A Three.js a 3D modellezés megvalósítása mellett rengeteg más alkalmazhatósága van. Például segítségével előállíthatók VR tartalmak, 3D animációk, koordináta geometria feladatok. Fő előnyei közé sorolható egyszerűsége, ugyanis A WebGL-alkalmazások vagy más könyvtárak nélküli elkészítéséhez hatalmas mennyiségű kódra van szükség, és a programozóknak világosan meg kell érteniük a WebGL specifikáció részletes részleteit. Hátránya a rendszer grafikus megjelenítési sebessége. Ez a probléma webes jellegéből származtatható, hiszen platform függően kell kiválasztani az adott operációs rendszerhez legmegfelelőbb engine-t. Ez a konverzió értelemszerűen a sebesség kárára megy. [8]

#### 2.2.2.4. Babylon.Js

A Babylon.Js egy keretrendszer, mely lehetőséget ad 3D alkalmazások, modellek, videók, játékok megvalósítására webes környezetben. A Babylon.Js az egyik legismerteb grafikai modellező webes keretrendszer, melynek népszerűsége és felhasználóinak száma napról napra növekszik. Különlegessége, hogy nyílt forráskódú mivoltát tekintve, maguk a felhasználóknak is lehetőségük van a keretrendszer bővítésére. A keretrendszer a TypeScript nyelvet preferálja. Nagyon felhasználóbarát rendszer, könnyen tanulható, értelmezhető. Az enginen keresztül lehetőségünk van entitásokat kezelni, specifikálni, amik majd renderelésre kerülnek. Tudjuk specifikálni az egyes objektumokat, a fényeket, a kamerákat, a textúrákat, a felületeket, az effekteket. Előnye egyértelműen a rendszer egyszerűségében rejlik. Fő hátránya a többi WebGL-hez hasonlóan a sebesség. [11]

### 2.2.3. Programnyelvek

A programnyelv kiválasztása nagyon fontos tényező. Fontos, hogy olyan nyelvben fejlesszünk szoftvert, amely az adott probléma megoldására a legalkalmasabb. Fontos szempont a kód áttekinthetőségét, újrafelhasználhatóságát. Ezen elvárásokat az objektumorientált programozási nyelvek elégítik ki a legjobban. Az alábbiakban ezért néhány objektumorientált programozási nyelv rövid tárgyalására kerül sor.

#### 2.2.3.1. Java

A Java vitathatatlanul a hálózati számítástechnika és az Internet nyelvévé vált. Habár a Java programnyelvet manapság legelterjedtebben webfejlesztésben használják, azért vannak kiemelkedően jó Java asztali alkalmazások, például a fentebb részletezett Sweet Home 3D szoftver. Hála a Java Remote Method Invocation (RMI) API-nak, az objektum orientáltságnak, a platformtól független adattípusoknak, az UNICODE karakterlánc kódolásának és a biztonsági

modellnek a magas szintű támogatásának. Rendkívül fontos szempont, hogy Java nyelv mivoltából adódva könnyen karbantartható és újra felhasználható kódot eredményezhet. Szintén előny, hogy rendkívül nagy platformok közti mobilitást és hordozhatósága. A Java-ban rejlő sajátos probléma, amely a teljesítménykritikus alkalmazásokban észrevehető, az a Java szemétgyűjtő (GC – Garbage Collector) hatása. A Java futásideje, a Java 3D futásideje és az alkalmazás kódja mind objektumokat hoz létre. Ezek az objektumok végül „szeméttévválnak, és a Java Virtual Machine (JVM) GC összegyűjti őket. Amíg a GC fut, érezhető a rendszer lassulása, ami több renderelt keret eldobásához vezethet. [12]

#### 2.2.3.2 C#

Manapság a C # az egyik hanem a legnépszerűbb programozási nyelv a világon. A C # -ot a Microsoft fejlesztette ki .NET-keretrendszerében, majd később az ECMA szabványként hagyta jóvá (ECMA-334). A C # programozási nyelv általános célú, OOPS-alapú programozási nyelv. A C # fejlesztőcsapatot "Anders Hejlsberg" vezette 2002-ben. A C # programozási nyelv az egyik olyan nyelv, amelyet a (CLI) közös nyelvi infrastruktúrához terveztek. A C # első verziója 1.0, a .NET framework 1.0-val, a Visual Studio pedig 2002. [13] A C# felépítése meglehetősen hasonlít a Java-hoz és fordítva, ezért erősségeikben és gyengeségeikben is hasonlítanak. A C# nagy erőssége az áttekinthetőség. Előnyei közé sorolható a nagyon részletes és könnyen áttekinthető dokumentáció és nyelv multiplatformitása. Van beépített 3d renderer, ez főleg a C# specifikus fejlesztők számára jelenthet nagy előnyt. Hátrány, hogy a WPF rég volt frissítve, kezd elavulttá válni. Hardver közeli konfigurációk programozása meglehetősen bonyolult vagy nincs rá lehetőség, ez komoly gondot okozhat hardver közeli fejlesztőknek.

### 2.2.4. Optimalizálási stratégiák

Az optimális tér eléréshez célszerű valamilyen optimalizálási stratégia mentén elindulni. Az optimalizáló algoritmusok mindig valamilyen keresési téren hajtják végre a keresést. Ezen belül próbálja meg megtalálni az optimális elemet. Jelen esetünkben a kereséséi tér a bútorok és azok pozícióí egy adott szobán belül. Az optimum megtalálására, rengeteg féle stratégia létezik, ezek közül, választottam ki párat, amiket érdemesnek találtam a feladat megoldására.

#### 2.2.4.1 Hegymászó algoritmusok

Az egyik legrégebbi és legegyszerűbb módszer. Egy létező keresési módszer módosításával valósul meg. Működése egyszerű, röviden leírható. A keresési tér egy véletlenszerű pontját választja ki kiindulási pontként, majd ezt követően választ egy másik pontot, ami, ha jobb mint az előző akkor az lesz az új legjobbnak ítélt pozíció, majd ezt véges sokszor elismétli egészen addig amíg már nem talál jobbat.
A hegymászó algoritmusból többféle megvalósítás is létezik, a fő különbség közöttük, hogy milyen formában vizsgálják az aktuális pont környezetét. [14]

### 2.2.4.1.1 Sztochasztikus

A sztochasztikus megoldás a klasszikus megvalósítása a módszernek. A fentiekben leírtakhoz hűen véletlen pozícióból indul. Az aktuálisan vizsgált pontot nevezzük most **p**-nek. Először tehát p-t kiválasztjuk a keresési tér egy véletlenszerű pontjára. Majd ezt követően egy ciklust futtatunk, amíg a megállási feltétel nem teljesül. A megállási feltétel, lehet egy fitnesz korlát, időkorlát, iterációszám korlát stb. Az algoritmus az alábbi ábrán látható.

```
1: függvény HillClimbingStochastic(𝕊, d_s, ϵ, f, StopCondition)
2:     p ←rnd 𝕊
3:     ciklus amíg ¬StopCondition()
4:         q ←rnd {x ∈ 𝕊 | d_s(x, p) = ϵ}
5:         ha f(q) < f(p) akkor
6:             p ← q
7:         elágazás vége
8:     ciklus vége
9:     vissza gpm(p)
10: függvény vége
```

2.5 Sztohasztikus hegymászó algoritmus pszeudokódja [14]

Megjelenik egy q változó, ennek epszilon sugarú környezetében választ véletlenszerűen egy új pozíciót, ha ez újonnan választott pont jobb az aktuális p pontnál, akkor ő lesz az új p pont, ha nem akkor új véletlenszerű pontot választ.
A módszer indeterminisztikus, tehát a kimenet értéke nem feltétlenül egyezik meg többszöri futtatás után, ha ugyanabból a kezdeti pontból indítjuk az algoritmust. [14]

### 2.2.4.1.2 Steepest Ascent

A fő különbség az előző algoritmushoz képest a q pont kiválasztásának módja Az elázóhoz képest itt a epszilon környezeten belül nem egy véletlen pontot választunk ki hanem az adott környezet legjobb elemét. A változtatásnak köszönhetően az esetleges lokális optimumba való beragadás is fölfedezhető. Az algoritmus pszeudo kódja az alábbi képen látható.

```
1: függvény HillClimbingSteepestAsc(𝕊, d_s, ε, f, StopCondition)
2:     p ←rnd 𝕊
3:     stuck ← hamis
4:     ciklus amíg ¬stuck ∧ ¬StopCondition()
5:         q ←argmin f(x) {x ∈ 𝕊 | d_s(x, p) = ε}
6:         ha f(q) < f(p) akkor
7:             p ← q
8:         különben
9:             stuck ← igaz
10:        elágazás vége
11:    ciklus vége
12:    vissza gpm(p)
13: függvény vége
```

2.6 Steepest Ascent hegymászó algoritmus pszeudo kódja [14]

Ez a megoldás véges számú iteráció után beakad, ezzel véget ér az algoritmus, ennek a vizsgálatára szolgál a stuck változó. Ez a megoldás determinisztikus, hiszen a módszer ugyanabból a pontból való futtatása, midig ugyanazt a kimenetet adja visszatérési értékként. Fontos megjegyezni, hogy az epszilon környezeten belüli legjobb elem kiválasztása a gyakorlatban ilyen formában nem kivitelezhető, hiszen előfordulhat, hogy az intervallumon belüli elemek száma végtelen. Ebben az esetben szükséges valamilyen intervallumon belüli bontást alkalmazni. Az algoritmus kevesebb ciklus iteráció alatt megtalálja a megoldást, viszont az egyes ciklus iterációk futási idejű költsége sokkal nagyobb az előző megvalósításhoz képest, hiszen minden egyes környezetvizsgálat esetén meg kell vizsgálnunk az összes elemet, vagy bontott elemeket. Összeségében nem biztos, hogy futási időben kedvezőbb. [14]

### 2.2.4.1.3 Random restart

Működése megegyezik a fentebb említett technikákkal, annyi különbséggel, hogy egymás után Megjelenik egy V halmaz melyeben az addigi bejárt pontok találhatók, hogy meg egyszer ugyanoda ne léphessen, amolyan féle adatcache.

```
1: függvény HillClimbingRandomRestart(𝕊, d_s, ϵ, f, StopCondition)
2:     V ← ∅
3:     ciklus amíg ¬StopCondition()
4:         p ←rnd 𝕊 \ V
5:         stuck ← hamis
6:         ciklus amíg ¬stuck ∧ ¬StopCondition()
7:             q ←argmin f(x) {x ∈ 𝕊 | d_s(x, p) = ϵ}
8:             ha q ∉ V ∧ f(q) < f(p) akkor
9:                 p ← q
10:                V ← V ∪ {q}
11:            különben
12:                stuck ← igaz
13:            elágazás vége
14:        ciklus vége
15:    ciklus vége
16:    vissza gpm(p)
17: függvény vége
```

2.7 Random Restart hegymászó pszeudo kódja [14]

Olyan szempontból előrelépés az eddigi hegymászó módszerekkel szemben, hogy általános jellemző a lokális optimumon belül való beakadás, erre kivédési módszert jelenthet ez az algoritmus, mégpedig, úgyhogy egyszerűen több különböző pontból indítjuk el a keresését. [14]

### 2.2.4.1.4 Értékelés

Előnye, hogy alapvetően a hegymászó módszerek kis számításigényű algoritmusok. Ebből és az egyszerű megvalósításából adódóan, gyors kipróbálást, tesztelést tesz lehetővé, hosszú implementálást nem igényel. Ebből adódóan célszerű lehet kombinálni már meglévő keresési algoritmusok végső stádiumaként. Szintén kiemelendő, hogy az algoritmus minden állapotában képes helyes megoldást adni, ez nem jelenti azt, hogy ez a legjobb megoldás, viszont ez nem minden feladatnál föltétlen célunk, néhol elegendő, ha a legjobb megoldásnak egy jó közelítése.

Hátránya, hogy legtöbbjükre jellemző a lokális pontba beakadás, bár ezt különböző metrikákkal lehet enyhíteni, de megszüntetni nem. Garantáltan helyes megoldást csak abban az esetben kapunk, ha a kereséséi tér pontjaiból monoton csökkenő út vezet a globális minimumba. Szintén hátrány, hogy a vizsgálandó epszilon környezetet nehéz eldönteni mi alapján válasszuk meg.

Gondoljunk bele, ha túl nagy az epszilon értéke, lehetséges, hogy folyamatosan átugorja a globális optimumot, ha az érték túl kicsi, a lokális optimumokba való beakadás gyakorisága jelentősen növekszik. [14]

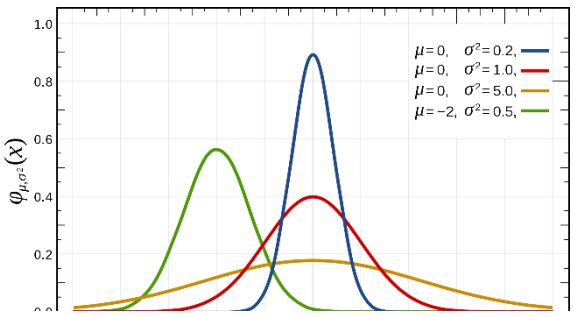
### 2.2.4.2 Véletlen Optimalizálás

Alapvetően megpróbája kiküszöbölni, a hegymászó algoritmus adta hátrányosságokat. Kiküszöböli az említett epszilon megválasztásának nehézségét. Kiküszöböli a lokális optimumba való ragadást. Működik többdimenziós keresés esetében is. Alapelveiben teljes mértékben megegyezik a hegymászónál látottakkal, azonban oly módon más, hogy a lépték mértéke nem fix, egy matematikai eloszlásfüggvény alapján lesz meghatározva.

1: **függvény** RANDOMOPTIMIZATION($\mathbb{S}, f, C, \mu, \sigma$, STOPCONDITION)
2: $p \xleftarrow{\text{rnd}} \mathbb{S}$
3: **ciklus amíg** $\neg$STOPCONDITION()
4: $step \leftarrow \begin{bmatrix} RND_n(\mu_1, \sigma_1) \\ RND_n(\mu_2, \sigma_2) \\ \dots \\ RND_n(\mu_{dim(\mathbb{S})}, \sigma_{dim(\mathbb{S})}) \end{bmatrix}$
5: $q \leftarrow p + step$
6: **ha** $\forall c \in C \mid c(q) \geq 0$ **akkor**
7: **ha** $f(q) < f(p)$ **akkor**
8: $p \leftarrow q$
9: **elágazás vége**
10: **elágazás vége**
11: **ciklus vége**
12: **vissza** $gpm(p)$
13: **függvény vége**

2.8 Random optimalizálás pszeudokód [15]

Az algoritmus ugyanúgy kezdődik, mint a hegymászó módszereknél látottak, egy p véletlenszerű pontból indul. Ezt követően viszont már összetetteb mivel, itt nem egy fix



2.9 Normál eloszlás függvény [22]

távolságot lépünk, hanem egy normál eloszlás alapján számított mértékkel. Az alábbi ábrán látható a normál eloszlás függvény.
Megvizsgáljuk a kiszámított értékkel módosított pont, jobb-e, mint az aktuális, ha igen, akkor tovább lépünk, ha nem akkor maradunk az aktuális pontban és új távolságot számolunk az eloszlás alapján. Azért lesz előnyös számunkra, hogy normál eloszlással állapítjuk meg a következő lépési pontot, ugyanis a normál eloszlás függvény nem korlátos, így mindig lesz valamekkora esély egészen nagy pozitív vagy negatív irányú lépésre, ezáltal van rá esély, hogy egy lokális optimumból kitudunk ugrani, ezzel feloldva a beakadást.
[15]

#### 2.2.4.2.1 Értékelés

A módszer rendelkezik a hegymászó módszerek kapcsán megismert előnyökkel, viszont nagyobb eséllyel képes kilépni egy lokális optimumból. Elméletben végtelen idő alatt, mindig megtalálja a globális optimumot. Indeterminisztikus módszer.
[15]

### 2.2.4.3 Particle Swarm Optimalizáció

A Particle Swarm Optimalizáció egy biológiailag inspirált algoritmus, mely a madárrajok mozgására épül. A madár rajoknál megfigyelhető, hogy képesek, a tökéletes együtt mozgásra és élelemkeresésére, úgy, hogy nincs közöttük egy kiemelet vezető, aki az egész folyamatot levezérli.
A módszer egyetlen egy populáción alapul. Minden egyed rendelkezik egy saját pozícióval és mozgási iránnyal és sebességgel. A pozíció a megoldáshalmaz egy eleme, a mozgási irány a tovább mozgás útjának iránya, a sebesség a tovább mozgás mértéke. Minden pozícióhoz rendelünk egy fitnesz értéket és a raj célja a legalacsonyabb fitnesz érték megtalálása, elérése. Fontos, hogy az egyedek nincs irányító elem, mindenki egyenértékű. Mindenkinek elérhető a raj által elért addigi legjobb optimum értéke mindenki ismeri a saját lokálisan elért optimumát. A lokális optimum ennél a módszernél minden egyed addigi elért legjobb eredménye. Az egyedek egy véletlenszerű pozícióból indulnak a mozgási irányuk a globális és a lokális optimumok függvényében. Ezeket súlyozva figyelembe véve allapítja meg a sebességüket.
Az alábbi ábrán látható az algoritmus pszeudo kódja.

```
1: függvény ParticleSwarmOptimization(𝕊, f, StopCondition)
2:     P ← InitializePopulation(𝕊)                    ▷ Position+velocity
3:     Evaluation(P, g^opt, f)
4:     ciklus amíg ¬StopCondition()
5:         CalculateVelocity(P, g^opt)
6:         ciklus ∀p ∈ P
7:             p ← p + p^velo
8:         ciklus vége
9:         Evaluation(P, g^opt, f)
10:    ciklus vége
11:    vissza gpm(g^opt)
12: függvény vége
```

```
1: függvény Evaluation(P, g^opt, f)
2:     ciklus ∀p ∈ P
3:         ha f(p) < f(p^opt) akkor
4:             p^opt ← p
5:             ha f(p) < f(g^opt) akkor
6:                 g^opt ← p
7:             elágazás vége
8:         elágazás vége
9:     ciklus vége
10: függvény vége
```

2.10 Particle Swarm Optimalizáció [16]

A kiindulási pozíciók után minden egyednek kiszámoljuk a fitnesz értékét majd ennek megfelelően módosítjuk a lokális és globális optimumot. A sebesség meghatározásához sebesség vektort használunk, ennek megállapítására figyelembe kell venni az aktuális pozíciót, a sebességét, majd ennek megfelelően módosítjuk a pozíciót. Majd ezt a folyamatot egészen addig ismételjük, míg nem teljesül valamilyen megállási feltétel. [16]

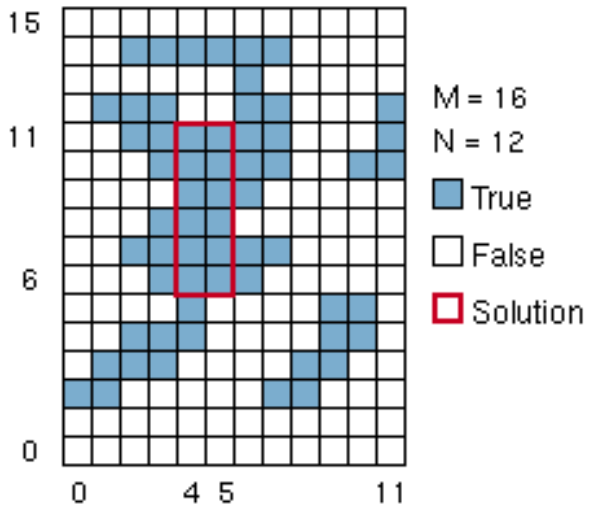
#### 2.2.4.3.1 Értékelés

Előnye , hogy nagy keresési terek esetén hasznos lehet, ugyanis a sok egyszerre indított egyed részletesen tudja megvizsgálni a kereséséi teret. Nagyon jól párhuzamosítható, ugyanis az egyedek mozgása független egymástól.
Hátránya viszont, hogy sok paraméterrel rendelkezik, ezért nehezen konfigurálható. [16]

## 2.2.5. Maximális téglalap alakú részhalmaz kiszámítási módszerek

Ahhoz, hogy el tudjuk dönteni, hogy mi számít számunkra jó bútor elrendezésű szobának, valamilyen jósági értéket (fitneszt) kell rendelni minden egyes elrendezési lehetőséghez. Ezt a fejezetrészt nevezhetnénk, fitnesz számító algoritmusoknak is. Feladatunk szempontjából annál nagyobb egy adott elrendezés jósága, minél nagyobb a szoba maximális téglalap alakú rézhalmaza, melyben nem található meg egyetlen objektum sem. Máshogy kifejezve minél nagyobb az üresen hagyott téglalap alakú tér. Mivel a fitneszt kiszámító függvény hívása igen gyakori, így kiemelt fontosságú futási idejének minimalizálása. Többek között ezért is lesz majd több megoldás is olvasható, a nagyobb futási idejű algoritmusoktól a kisebbek felé haladva.

Fontos megjegyezni, hogy a teret egy logikai tömbként kezeljük, ahol 1 $cm^2$-t feleltetünk meg a tömb egy elemeként. Az üres mező kap logikai igaz értéket, ahol valamilyen objektum tartózkodik kap logikai hamis értéket. Ezt szemlélteti az alábbi ábra.




2.11 Probléma szemléltetése [20]

Az ábrán Solution-nel jelölt résztábla kiszámítására az alábbi három módszer szolgál megoldásul.

### 2.2.5.1 Nyers erő módszere

Az első ilyen módszer a nyers erő módszerre, vagy angolul Brute Force. Nyilván való, hogy egy adott (T) tér felosztható véges számú részhalmazokra. Ha végig tudnánk menni T tér összes létező részhalmazán, majd ezeknek méreteit kiszámolva. melyek nem tartalmaznak objektumot (hamis logikai értéket) végrehajtanánk egy maximumkiválasztást akkor végeredményként megkapnánk, az adott T tér legnagyobb területű téglalap alakú rézhalmazát. A Nyers erő módszere pontosan ugyanezt csinálja. Maga az algoritmus pszeudo kódja az alábbi ábrán látható. [17] [18] [19]

```
main algorithm:
   for ll.x = 0 .. N
      for ll.y = 0 .. M
         for ur.x = ll.x .. N
            for ur.y = ll.y .. M
               if area(ll, ur)>area(best_ll, best_ur)
               and all_ones(ll, ur)
                  best_ll = ll; best_ur = ur
end main algorithm
define area(ll, ur)
   if ll.x>ur.x or ll.y>ur.y // If ur is left of or
      return 0               // below ll: return 0
   else
      return (ur.x-ll.x+1)*(ur.y-ll.y+1)
define all_ones(ll, ur)
   for x = ll.x .. ur.x
      for y = ll.y .. ur.y
         if b[x, y]==0
            return false
   return true
```

2.12 Brute Force algoritmus [20]

Ahol a **b** az N x M elemű logikai tömb.

**best_ll** az aktuálisan legjobb megoldás halmaz béli elem, bal alsó sarkának koordinátáit tartalmazó pár (pair) adatszerkezet.

**best_ur** az aktuálisan legjobb megoldás halmaz béli elem, jobb felső sarkának koordinátáit tartalmazó pár (pair) adatszerkezet.

a **ll** az aktuálisan vizsgálandó részhalmaz bal alsó koordinátáit tartalmazó pár adatszerkezet.

az **ur** az aktuálisan vizsgálandó részhalmaz jobb felső koordinátáit tartalmazó pár adatszerkezet.

az **area** függvény aktuálisan vizsgált részhalmaz méretét

az **all ones** függvény visszaadja, az aktuálisan vizsgált részhalmaz elemén belül található-e objektum (hamis logikai érték), ha igen, akkor hamis a visszatérési értéke, hanem akkor igaz.

Mégis milyen az algoritmus futási ideje? MxN lehetséges választási lehetőség van az ll-re. Mindegyik választásnak egy és M×N között van egy és MxN megfelelő jobb felső sarka: Ez azt jelenti, hogy átlagosan O(MxN) ur választási lehetőség lesz minden ll-re. Ezért a b részhalmazok teljes száma O(M2xN2) nagyságrendű. Ezeknek az altábláknak az átlagos elemszáma szintén O(MxN), és bár az algoritmusnak csak az altáblák töredékét kell beolvasnia, mégis konstruálhatunk olyan eseteket, amikor a b altáblánként elvégzett műveletek átlagos száma O(MxN). Ezért ennek az algoritmusnak a legrosszabb esetben O(M3xN3) a komplexitása, ami nagyon rossz.
[17] [18] [19] [20]

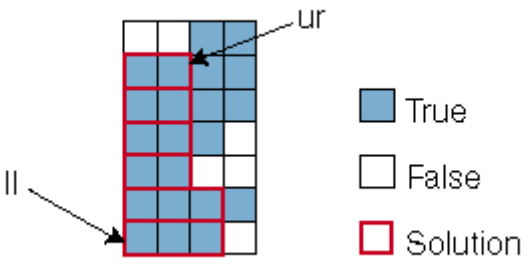
### 2.2.5.2 Emlékező módszer

Bár nem hatékony, az első algoritmus jó alapot biztosít a továbbiakban. Keresési irány bevezetésével jelentős teljesítményjavulás elérése lehetséges. Az első listában szereplő algoritmus bármilyen sorrendben fel tudja sorolni a résztáblákat, és mégis megtalálja a helyes megoldást. Ehelyett ki lehet használni azt, hogy ha egy kis téglalap hamis logikai értéket tartalmaz, akkor az összes környező téglalap is tartalmazni fog egy hamis logikai értéket. Tehát minden lehetséges bal alsó sarokhoz téglalapot adok. Ez a növekedési folyamat egyszerűen az egyes számok jobb felső sarkát tartalmazó téglalap meghatározását eredményezi: a nyers erő módszerénél használt **all_ones** függvényre többé nem lesz szükség.

```
// The cache starts with all zeros:
c[0 .. M-1] = 0
main algorithm:
   for 11.x = N-1 .. 0
      update_cache(11.x)
      for 11.y = 0 .. M-1
          ur = grow_ones(11)
          if area(11, ur)>area(best_11, best_ur)
             best_11 = 11; best_ur = ur
end main algorithm
define update_cache(x)
   for y = 0 .. M-1
      if b[x, y]!=0
         c[y] = c[y]+1
      else
         c[y] = 0
define grow_ones(11)
   ur = (11.x-1, 11.y-1) // Zero area ur-choice
   y = 11.y-1
   while y+1<M and b[y]!=0
      y = y+1; // Scan a new layer
      x = min(11.x+c[y]-1, x_max) // Use the cache, Luke!
      x_max = x
      if area(11, (x, y))>area(11, ur)
         ur = (x, y)
```

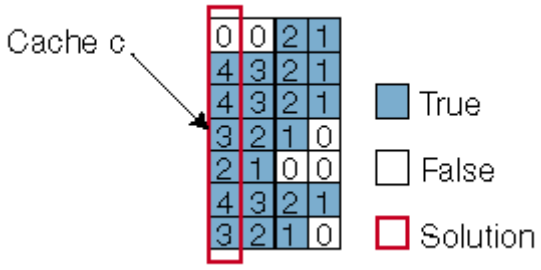
2.13 Emlélkező módszer algoritmus [20]

A jelölések javarészt megegyeznek a nyers erő módszerénél használtakkal. Az egyik újdonság az előbbihez képest az ábrán látható **grow_ones** függvény, mely az algoritmus magját képezi. Ez a funkció a lehető leghosszabb összefüggő láncot keresi, először a megadott bal alsó sarokból jobbra haladva a külső while ciklusban. Ennek a vizsgálatnak a jobb szélét rögzíti az **x_max**, amikor a külső ciklus következő iterációjában egy réteget próbál hozzáadni, akkor nem próbál meg szélesebb téglalapot beolvasni, mint az első (mert az értelmetlen lenne). Az itt látható ábra a bal alsó ponton növesztett egymást követő rétegek mutatja.




2.14 Grow Ones szemléltetés [20]

A másik fontos érdemi újítás a cache bevezetése. Erre azért van szükség ugyan is a **grow_ones** függvény gyakran ugyanazt az egyesekből álló területet vizsgálja újra és újra. Az ábrán például az **ll** jelű elem három egyesből álló sávot indít. Majd ezután az algoritmus meghívja a **grow_ones**-t az **ll**-től egy pozícióval jobbra lévő **p** elemre. Vegyük észre, hogyha az algoritmus emlékezett volna, hogy az **ll**-től jobbra három logikai igaz érték van, akkor plusz ciklus futás könnyen észlelhette volna, hogy **p** egy két logikai igaz értékből álló sorozat bal oldali vége.



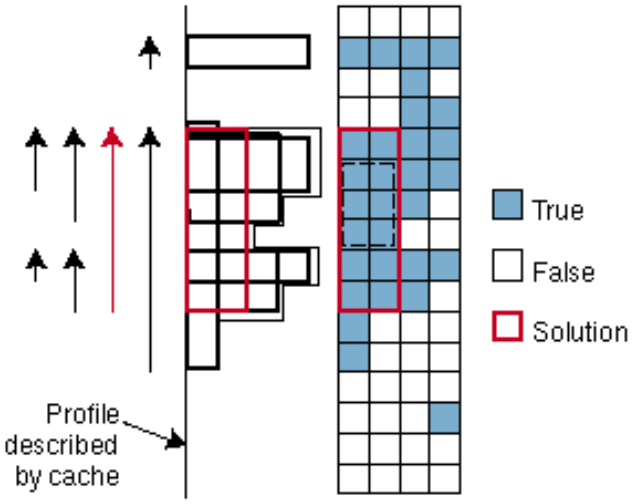

2.15 Cache szemléltetés [20]

Ahhoz, hogy az algoritmus megjegyezze a **b** tömb minden egyes elemének jobb oldalán lévő egyesek számát, bevezetésre kerül egy cache. Az ábra a cache egymást követő tartalmát mutatja a végső maximális területű téglalap, a megoldás környezetében. Ennek a gyorsítótárnak az egyik oszlopról a másikra való frissítése nem túl költséges művelet, de a gyorsítótár kezdeti értékeit nullára állítjuk és a gyorsítótár feltöltését a b legjobboldalibb oszlopától indítjuk. Ez megmagyarázat, arra, hogy legkülső ciklus iránya miért változott az ellenkezőjére. A gyorsítótárat oszloponként egyszer frissíti az **update_cache** eljárás.

Mindezen újításokat kombinálva azt kapjuk, hogy az” emlékező” algoritmus futási ideje $O(M^2 \times N)$, ami nagyon jelentős javulás a nyers erő módszeréhez képest. [17] [18] [19] [20]

### 2.2.5.3 Kihasználó szerkezet

A köztes eredmények gyorsítótárazása vagy annotálása egy általános technika, amely számos algoritmushoz alkalmazható. Ez egy olyan technika, amely gyakran könnyen integrálható meglévő algoritmusokba. Végső soron azonban a legtöbb optimális algoritmus a probléma jellegétől függ és ez a alól a maximális téglalap sem kivétel.




2.16 Kihasználaható előny szemléltetése [20]

Az előző algoritmus lassítja, hogy minden egyesekből álló téglalapot megvizsgált, amely beleillik a megoldás résztáblába. Néhány ilyen téglalapot azonban nem szükséges megvizsgálni. Például az ábrán látható szaggatott téglalap a gyorsítótár aktuális tartalma által leírt résztáblán belül van, de a táblában vannak nagyobb téglalapok, amelyek a szaggatottat körülveszik. Elegendő tehát, csak azokat a téglalapokat megvizsgálni, amelyek a résztáblán belül vannak, de nem tartalmazza őket egy másik téglalap sem. Ezeket a téglalapokat az ábra bal felében megvastagított vonallal, melyben a megoldás téglalapja piros színnel jelölt.

Az ábra bal szélén lévő nyilak a téglalapok függőleges hosszát szemléltetik. A legfontosabb megfigyelés itt az, hogy ezek a kiterjedések egymásba illeszkednek. Ha a nyílvégeket alulról felfelé haladva bejárjuk, látható, hogy egy nyíl, amely egy másiknál előbb kezdődik, nem fog később befejeződni. Érdemes azt is szemügyre venni, hogy a nyilak onnantól kezdődnek, ahol egy téglalap kiszélesedik, és ott érnek véget, ahol egy másik szűkül. Tehát egy vagy több egymásba ágyazott téglalap megnyitásához el kell érünk a gyorsítótárat, amely jelentős futási időt generál. Azonban a téglalapokat úgy is tudjuk csökkenteni, hogy mindeközben a

gyorsítótár értéke nem változik, ebben az esetben nem nyílnak meg fölöslegesen új téglalapok, és nem záródnak be a megnyitottak.

Ennek a megvalósítására tökéletes alapjául szolgál a verem adatszerkezet. Ez az adatszerkezet tökéletesen megfelel az előbb említett bezárások (push) és visszatöltések kezelésére (pop). A végső algoritmusomban használt verem nyomon követi, hogy hol volt téglalap szélesedés, és

```
// The cache starts with all zeros:
c[0 .. M-1] = 0 // One extra element (closes all rectangles)
main algorithm:
   for x = N-1 .. 0
      update_cache(x)
      width = 0 // Width of widest opened rectangle
      for y = 0 .. M
         if c[y]>width // Opening new rectangle(s)?
            push(y, width)
            width = c[y]
         if c[y]<width // Closing rectangle(s)?
            do
               (y0, w0)= pop()
               if width*(y-y0)>area(best_ll, best_ur)
                  best_ll = (x, y0)
                  best_ur = (x+width-1, y-1)
               width = w0
            until c[y]>=width
            width = c[y]
            if width!=0 // Popped an active "opening"?
               push(y0, width)
```

2.17 Kihasználó szerkezet algoritmus [20]

mi volt a szélesség a profilszélesítés előtt. Ez lehetővé teszi az algoritmus számára, hogy egy szélesebb téglalap bezárásakor visszanyerje a már megnyitott keskenyebb téglalapok szélességét. Az alábbi képen olvasható az algoritmus.

Futási időt tekintve a memória sokszori olvasása, írása és visszatöltése a futási idő kárára megy, azonban ez az algoritmus összeségében O(M) memóriát igényel, és akkor is működik, ha az adott b tömb elemeit adjuk meg bemeneti paraméterként, szóval az imént tárgyalt megközelítések közül futási idő szempontjából a leghatékonyabb. [17] [18] [19] [20]

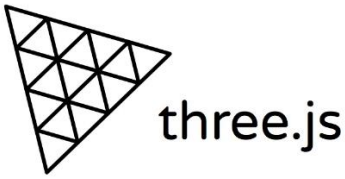
## 2.2.6. Összegzés

Az előzőekben megvizsgálásra kerültek a megvalósításhoz szükséges elemek egyes alternatívái. Tárgyalásra kerület a különböző adatbázisrendszerek, a számítógépes grafika, ezen belül, a webes és a nem webes grafikai megjelenítők, valamint a programnyelvek.

Adatbázis rendszerek közül a választásom a MongoDB-re esett, mivel a projektemben nincs szükség strukturálatlan adatok tárolására, viszont szükséges lehet az adatok gyors keresésére és visszaadására, amire a MongoDB tökéletesen megfelel.

Grafikai megjelenítésre a Three.Js-t tartom a legalkalmasabbnak, főként egyszerűsége miatt. A projektben nem lesz szükség komolyabb grafikai elemek megjelenítésére. A renderer sebesség a rendszerben nem jelent gondot, a renderert objektumok száma nem fog problémát okozni.

Programozási nyelvként a C#-ra esett a választásom, itt elsődleges szempont a nyelvel való több éves tapasztalatom volt, ami a másik alternatívákról nem mondható el.

| | |
|---|---|
| **Adatbázis rendszer** |  |
| **Grafikai megjelenítő** |  |
| **Programozási nyelvek** |   |
| **Optimalizálási módszer** | Particle Swarm optimalizáció |
| **Fitnesz számítási módszer** | Kihasználó szerkezet |

2.1 Használt módszerek összefoglaló táblázat

# 3. Követelmény specifikáció

A programban szobákat lehet létrehozni, törölni és a benne lévő bútorokat modifikálni, méret pozíció alapján. A program képes egy megadott specifikációk alapján létrehozni egy helykihasználás szempontjából optimális elrendezést. Ha a szoba berendezése elkészült a programban lehetőség van legenerálni egy 3D-s modellt, ahol megtekinthetnénk az elkészült szobát. Ezzel vizuálisabb látványvilág alapján lehet megállapítani, hogy a szoba megfelel-e az elvárásoknak. A programtól elvárt funkcionális és nem funkcionális követelmények részletes leírását foglalja magába az alábbi két táblázat.

## 3.1. Funkcionális követelmények

| ID | Mint… | Képesnek kell lenni… | Tehát ezért… |
|---|---|---|---|
| 1 | felhasználó | Teljes bútorkészlet adatbázis lekérésére. | Az összes bútor lekérése a bútorkészlet adatbázisból. |
| 2 | felhasználó | Teljes bútorkészlet adatbázis szűrésére. | A szűrési feltételeknek megfelelő bútorok lekérése a bútorkészlet adatbázisból |
| 3 | felhasználó | Teljes bútorkészlet adatbázisban való keresésre. | A keresendő elem/elemek megkeresésé az adatbázisban és megjelenítése a grafikus menüsorban. |
| 4 | felhasználó | Szoba méreteinek megadására. | A szoba elmentése és megjelenítése a felülnézeti grafikus felületen. |
| 5 | felhasználó | Szoba törlésére. | Az addig megadott méretek, lehelyezett bútorok eltávolítása. A törlés megjelenítése a grafikus felületen. |
| 6 | felhasználó | Objektumok lehelyezésére. | Az objektum hozzáadása a szobához a megfelelő pozícióba, a felülnézeti grafikus felületen való megjelenítés. |

| | | | |
|---|---|---|---|
| 7 | felhasználó | Objektumok eltávolítására. | Az objektum eltávolítása a szobából, a változás megjelenítése a felülnézeti grafikus felületen. |
| 8 | felhasználó | Bútor kiválasztására. | A kiválasztott bútor kijelölése. |
| 9 | felhasználó | Kiválasztott bútor lehelyezésére. | A kiválasztott bútor megjelenítése a grafikus felületen. |
| 10 | felhasználó | Kiválasztott bútor átméretezésére. | A kiválasztott bútorhoz tartozó menüsor és az észlelt változások megjelenítése. |
| 11 | felhasználó | Kiválasztott bútor forgatására. | A módosítások mentése és megjelenítése a felülnézeti grafikus felületen. |
| 12 | felhasználó | Kiválasztott bútor törlésére. | A kiválasztott bútor eltávolítása a szobából. |
| 13 | felhasználó | Kiválasztott bútor mozgatása. | A módosítások elmentése és megjelenítése felülnézeti grafikus felületen. |
| 14 | felhasználó | A kiválasztott bútor átméretezésére a grafikus felületen és pontos méret megadásra a felületen kívüli bemeneti mezők segítéségével. | A kiválasztott bútor módosítása mid a grafikai mind a 3d dimenziós megjelenítő felületen. |

| | | | |
|---|---|---|---|
| 15 | felhasználó | Az optimális helykihasználású szobaterv lekérése. | A legenerált szoba megjelenítése a felülnézeti grafikus felületen. |
| 16 | felhasználó | A felülnézeti szobaterv háromdimenzióban való megtekintésre. | A felülnézeti szobaterv háromdimenziós modellbe való legenerálása és grafikus felületen való megjelenítése. |
| 17 | felhasználó | A háromdimenziós szobaterv forgatására. | A háromdimenziós modell megadott szögben való forgatása és az elforgatott szobaterv legenerálása és megjelenítése a grafikus felületen. |
| 18 | felhasználó | A háromdimenziós szobaterv elmentésére. | A szobaterv elmentése az adatbázisba. |
| 19 | felhasználó | Az elmentett háromdimenziós szobatervek kilistázására. | Az elmentett háromdimenziós szobatervek listázása a grafikus felületre. |
| 20 | felhasználó | Az elmentett háromdimenziós szobatervek közüli választásra és annak betöltésére. | A kiválasztott, mentett szobaterv betöltése a grafikus felületre. |
| 21 | felhasználó | A kétdimenziós, felülnézeti szobaterv elmentésére. | A szobaterv elmentése az adatbázisba. |
| 22 | felhasználó | Az elmentett kétdimenziós, felülnézeti szobaterv betöltésére. | A kiválasztott, mentett szobaterv betöltése a grafikus felületre. |

3.1. Funkcionális követelmények

## 3.2. Nem funkcionális követelmények

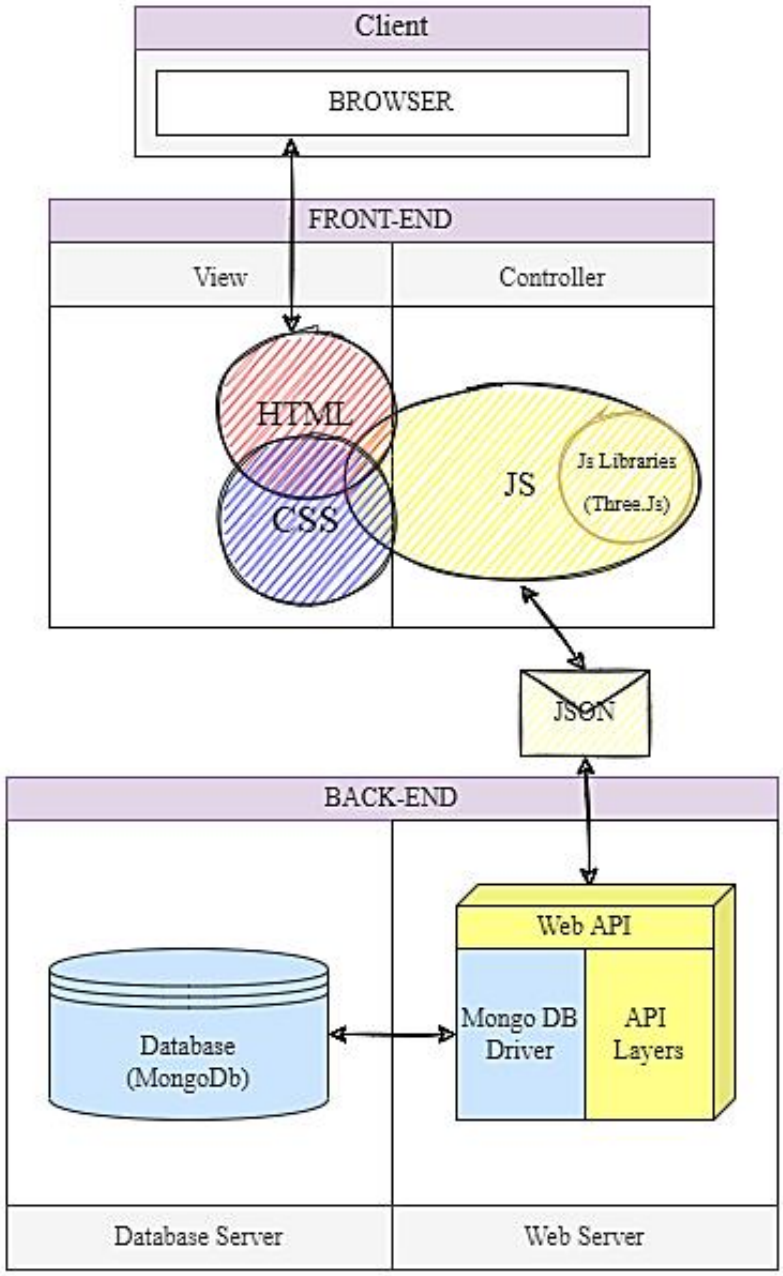
| | |
|---|---|
| Teljesítmény és méretezhetőség | A rendszer az eredmények visszaadására gyorsan képes, viszont nagyobb terhelés mellett nagyobb válaszidőre lehet számítani. |
| Hordozhatóság és kompatibilitás | A szoftver Windows operációs rendszeren futtatható. A futás nem ütközik más futó alkalmazásokkal, folyamatokkal. |
| Megbízhatóság, rendelkezésre állás, karbantarthatóság | A rendszerben ritka a hibák fellépése, viszont, ha hiba keletkezik az a rendszer leállásához vezet. |
| Biztonság | A rendszeradatok védelmére az alap tervezési minta adta védelemén kívül egyéb védelem nem áll rendelkezésre. |
| Lokalizálás | A rendszer megfelel a helyi lokalizációnak. |
| Kihasználhatóság | A rendszer használata a felhasználók számára egyszerű, könnyen elsajátítható. |

3.2. Nem funkcionális követelmények

# 4. Készítendő rendszer megtervezése

## 4.1. Rendszerterv

A rendszer alapvetően két fő részegységre bontható. Back-end és Front-end. Ebben a fejezetben ezen két rész és ezen belül elhelyezkedő részegységek kerülnek tárgyalásra, szekvenciális diagrammok, komponens diagrammok és folyamat, ábrák segítségével. A rendszer terve, komponensei, folyamatainak betekintése kívülről befelé haladva kerül tárgyalásra.

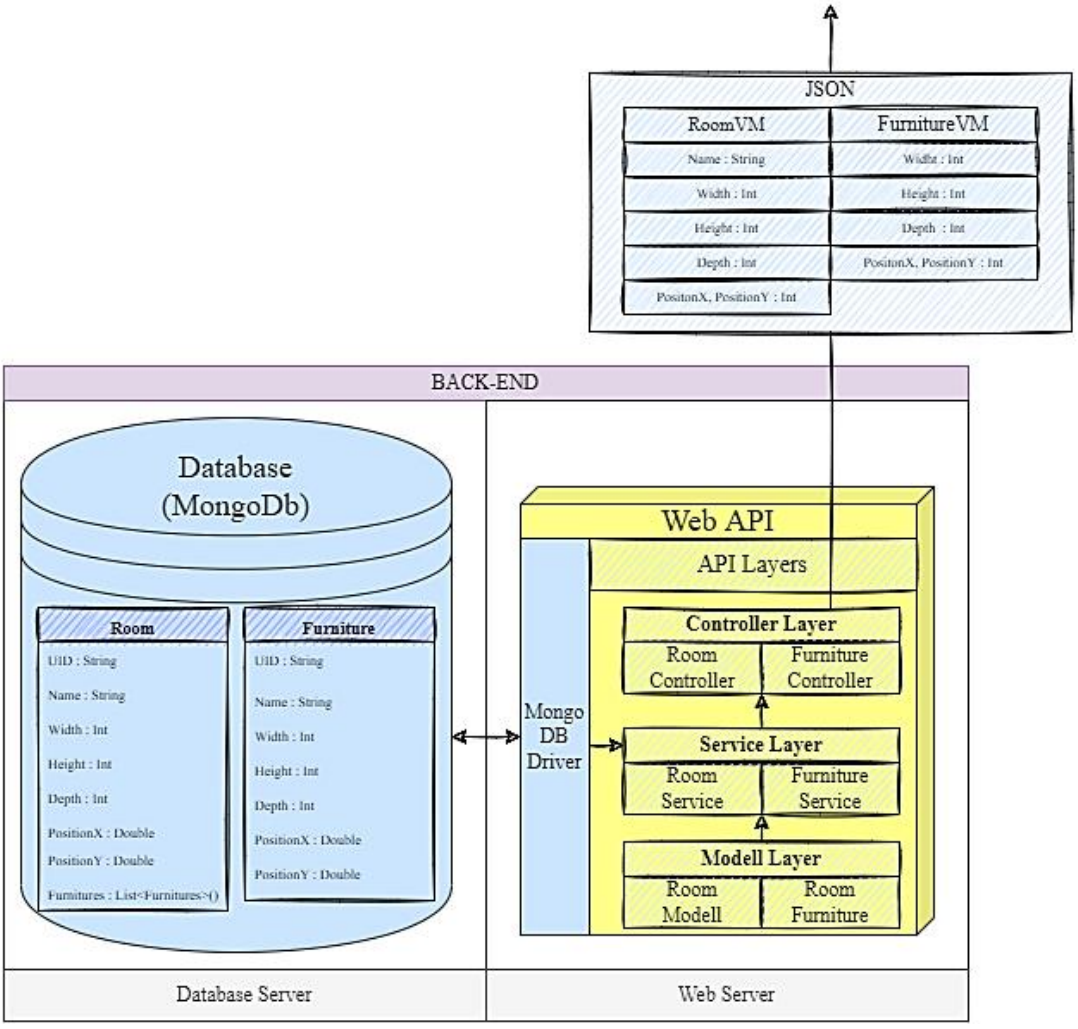



4.1. A rendszer komponens diagrammja

Mindenekelőtt fontos, hogy teljes betekintés nyerjünk a rendszer egészéről. Ezt a tervet ábrázolja a 4.1.-es ábra. A Back-End elsődleges feladatai az adatok menedzselése és a kliens felől érkező kérések kiszolgálása. A Front-end fő feladata az adatok megjelenítése és felhasználói interfészek biztosítása az adatok begyűjtésére. Napjainkban nem ritka, hogy a legtöbb cég ebben a felosztásban fejleszt. Főként nagyobb volumenű webalkalmazások esetén. Ebben a rendszertervezési mintában, általában a fejlesztők szintén ezen két nagy részegység mentén szeparálódnak. Azokat a fejlesztőket, akik mind Back-end, mind front-end feladatokat is ellátnak, nevezzük FullStack fejlesztőknek. A Back-end és a Front-end közti kommunikáció JSON packagek küldésével valósul meg. Majd ezeket a packageket földolgozva jelennek meg az adatok böngészőn keresztül a felhasználó számára.

### 4.1.1. Back-End

Elsőként az alkalmazás Back-end rétege kerül kialakításra, ez jelenik meg a 4.2-es ábrán. Ez a réteg további két felé bomlik. A Database Serverre és a Web Serverre. A Database Server felelős az adatok hatékony eltárolásáért. A Web Serverrel az egyéni driveren keresztül kommunikál, a mi jelen esetben a MongoDb driver.




4.2. Back-End részletes komponens és class diagramm

Az adatbázis Json fájlokat tárol, Az adatbázisban szobák (Room) és azon belül bútorok(furniture) tárolódnak. Egy szoba és azon belüli bútor tárolásának felépítése a 4.3.-as ábrán látható. Egy szobában több bútort is el lehet tárolni.

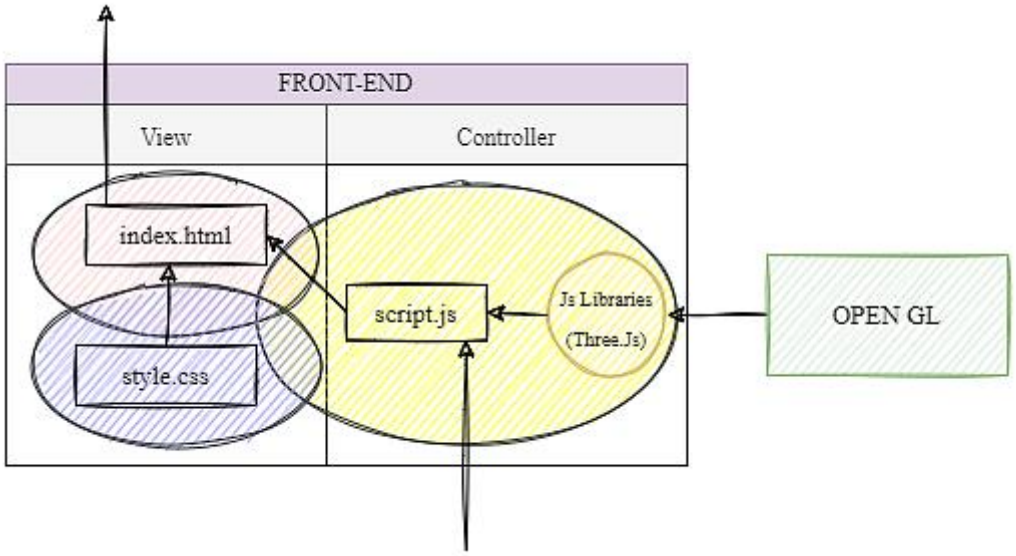

```
[{
        "id": "string",
        "name": "string",
        "widthX": 0,
        "widthY": 0,
        "height": 0,
        "furnitures": [{
                "id": "string",
                "name": "string",
                "width": 0,
                "depth": 0,
                "height": 0
                        }]
}]
```


4.3. Room és Furniture Json formátumban

A Service lekéri az adatbázistól az adatokat és elvégzi a Modellek és a View Modellek közti, konverziót, majd továbbítja az adatokat a Controller felé. Külön Service osztály kezeli a szobákat (RoomService) és a bútorokat (FurnitureService). A Controller Dependency Injectionnel megkapja a Servicet, majd az ő feladata a HttpGet, Post, Put, Delete hívások kiszolgálása. A Servicehez hasonlóan itt is két külön kontroller kezeli a szobákat (RoomController) és a bútorokat (FurnitureService). Majd ezt követően a szükséges műveletek elvégzése után JSON formátumban (4.3. ábra) küldi ki a választ, amivel aztán a Front-end tud dolgozni.

## 4.1.2. Front-End

A Front-End három fő részből tevődik össze. Az egyik a Html ami a megjelenítésért felelős. A Css ami a kinézetrért és a JavasScript, a Controller ami a működésért felel.

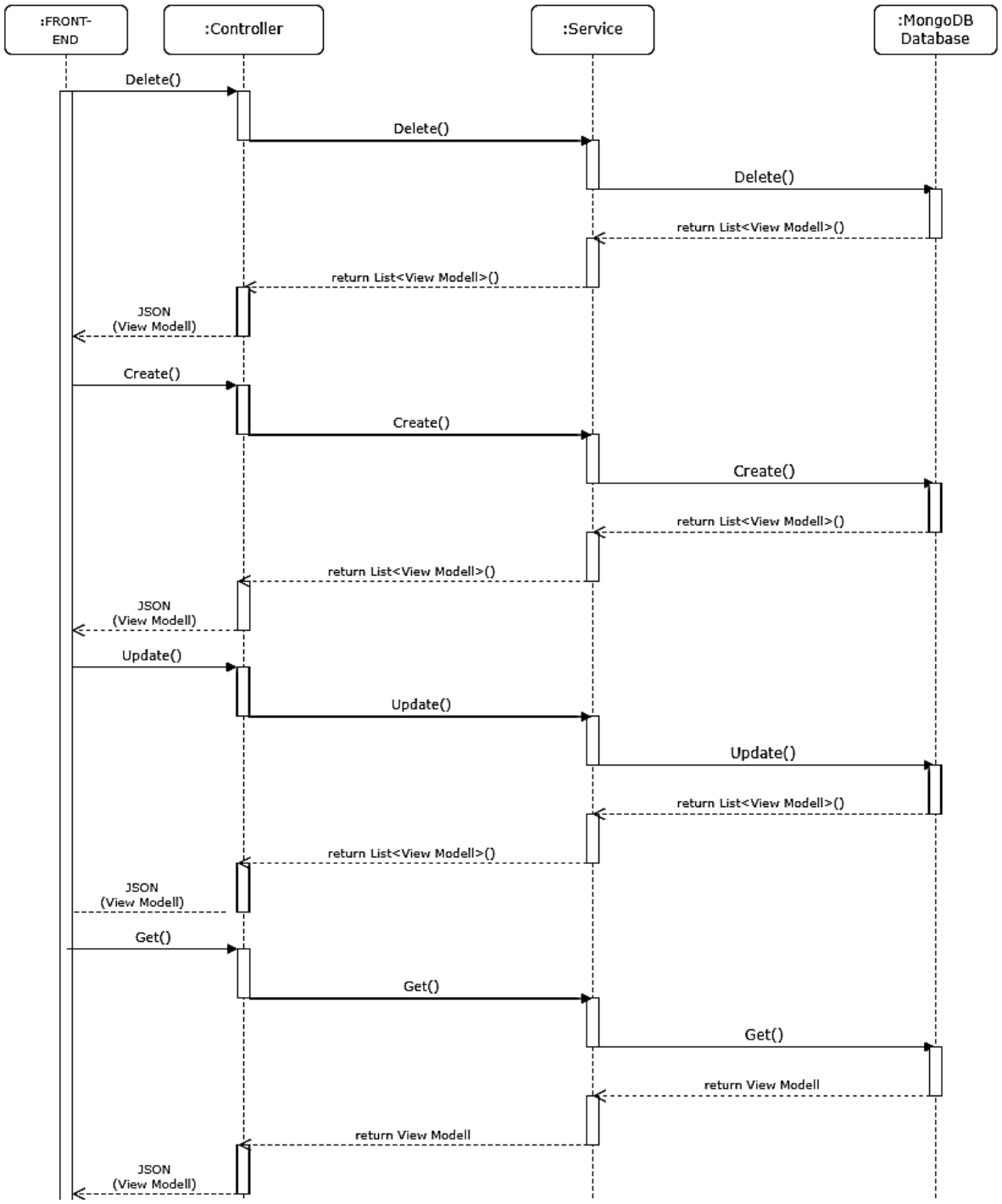



4.4. Front-End részletes komponens diagramm

Az index.html határozza meg a böngészőben megjelenő html elemeket. A style.css felel a html elemek megjelenéséért. A script.js az összeköttetés a Back-enddel, valamint ő valósítja meg a html elemek működését és a three.js könyvtár segítségével a 3d megjelenítést. A three.js oprációs rendszertől függően választja ki a renderelési közegét. Jelen esetben az oprációs rendszer a Microsoft Windows 10, ezért a renderelési környezet az OpenGL lesz. Valamint loggolást végez a felhasználó által végzett változtatásokról, amelyeket a felhasználó tud véglegesíteni, ezzel tehermentesítve az adatbázist és gyorsítani a felhasználó élményt.

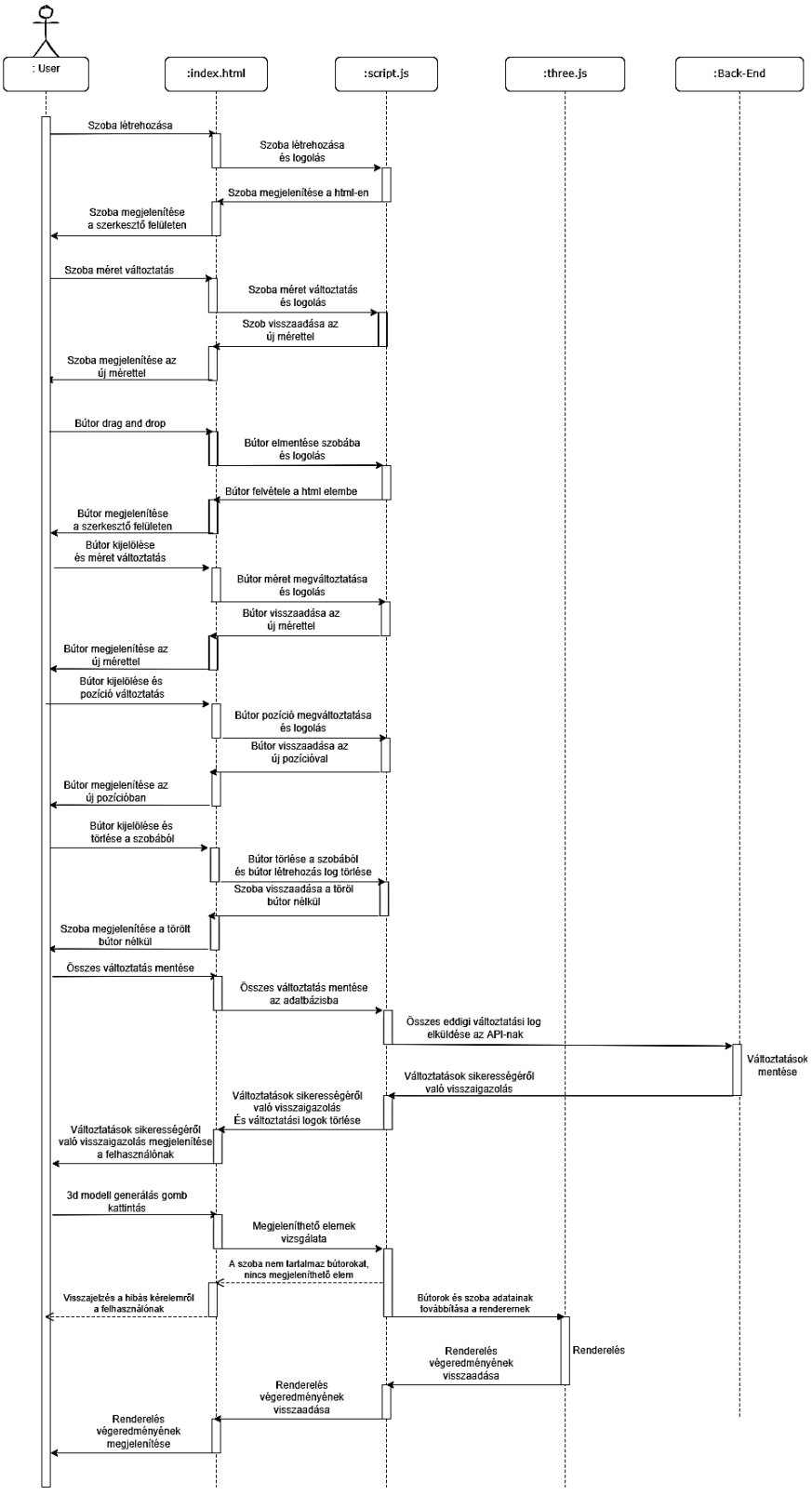
## 4.2. Funkciólista

A Web API az alapvető CRUD feladatok ellátását biztosítja, mely szekvenciális diagrammja a 4.4-es ábrán látható.




4.5. Back-End CRUD szekvencia diagramm

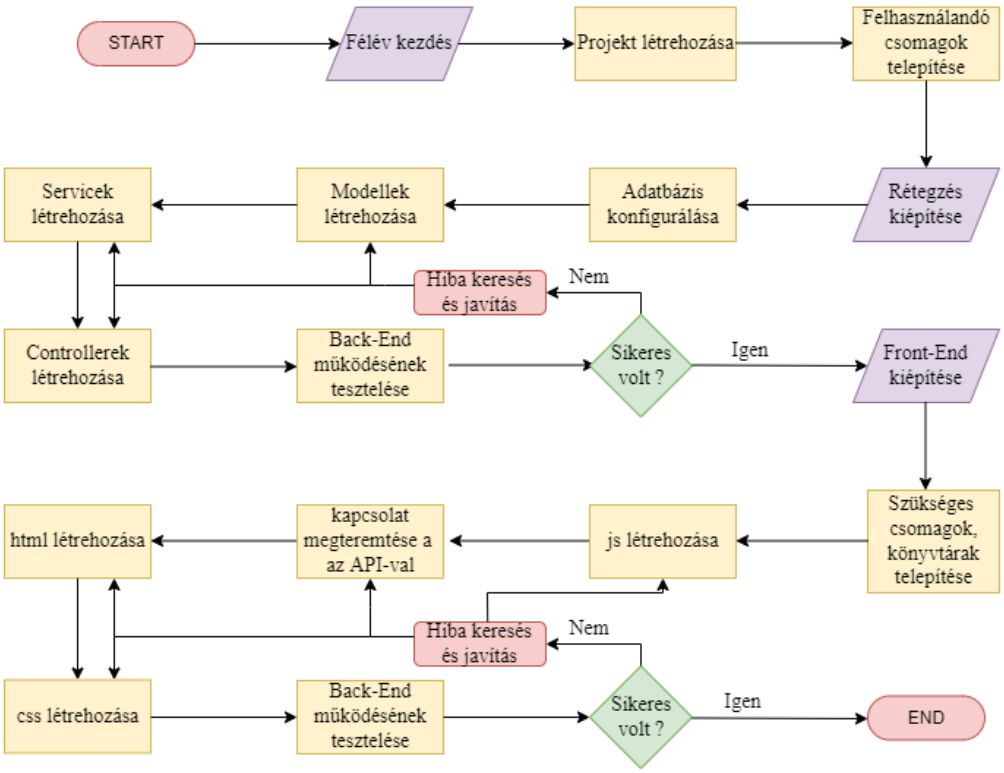
Az alábbiakban a Front-End szekvenciális diagrammja látható, mely a felhasználó és a Front-End közti kommunikációs folyamatokat szemlélteti.




4.6. Front-End szekvencia diagramm

Minden egyes felhasználói input azonnali elmentése az adatbázisba nagyon költséges feladat, ezért a Front-End, loggolást készít a változtatásokról, melyek a mentés gomb lenyomását követően kerülnek bele ténylegesen az adatbázisba. Ezt követően az addigi loggok törlődnek. Ez leolvasható a 4.5.ös szekvencia diagrammról.

# 4.3. Fejlesztés tervezett menete




4.7. Fejlesztés menete

# 5. Fejlesztés

Alapvetően a fejlesztés három fő és ezen belül is két alrészre bontható. részre bontható. Az első az adatbázis konfigurálása, a második a Back-End rétegek kialakítása, a harmadik a Front-End és UI felület kialakítása. Ezen belül a html canvas műveletek implementálása, valamint a 3D megjelenítő felület canvassal való szinkronizálása.

## 5.1. Adatbázis konfigurálása

Az első fontos lépés az adatbázis konfigurálása. Itt történik meg a táblák és a relációk létrehozása.

Kezdetben Microsoft Sql Server adatbázissal dolgoztam, majd rájöttem, hogy a szobák és bútorok tábla teljesen függetlenek egymástól, így a relációs adatbázis adta előnyöket nem lehet kihasználni. Ezért váltottam később a MongoDB Atlasra. Mivel a MongoDB Atlas bizonyos adatmennyiség felett csak fizetős szolgáltatásként vehető igénybe, így váltottam a teljes mértékben ingyenes azonban lokális verziójára a MongoDB Compassra. Az adattáblák felvétele meglehetősen egyszerű feladatnak bizonyult, kisebb utánajárással nem okozott komolyabb nehézséget.

## 5.2. Back-End rétegek kialakítása

A következő lépés a rétegek kialakítása volt, ahol az alapvető CRUD műveltek és a Front-End, Back-End közti kommunikáció került kialakításra. Maga az implementálás előzetes ismereteim alapján egészen gördülékenyen haladt.

Ebben a fejezetben kiemelném, hogy a rendszer föl lett készítve a párhuzamos, egyidőben való használatra. Erre szükség lehet abban az esetben, ha az adatbázison egy időben több felhasználó is változtat. Ez webalkalmazások tekintetében egy valós, létező probléma, amire célszerű odafigyelni.

Ez esetemben azt jelenti, hogy az API kihasználja a MongoDb adatbázis nyújtotta magas szintű adat párhuzamosságot. Ennek értelmében az adatbázisba író/felülíró és olvasó metódusok async-awaitek. Ebből összesen három van:

Az első a **PushFurnitureAsync** metódus, amely ez adatbázisba való egyidejű írásért felelős.

```
private async Task<UpdateResult> PushFurnitureAsync(string roomId, Furniture furniture)
{
    var filter = Builders<Room>.Filter.Eq(r => r.Id, roomId);
    var update = Builders<Room>.Update.Push(r => r.Furnitures, furniture);
    return await _room.UpdateOneAsync(filter, update);
}
```

5.1 PushFurnitureAsync metódus implementálása

A második a **PullFurnitureAsync** metódus, amely ez adatbázisba való egyidejű olvasásáért felelős.

```
private async Task<UpdateResult> PullFurnitureAsync(string roomId, Furniture furniture)
{
    var filter = Builders<Room>.Filter.Eq(r => r.Id, roomId);
    var update = Builders<Room>.Update.Pull(r => r.Furnitures, furniture);
    return await _room.UpdateOneAsync(filter, update);
}
```

5.2 PullFurnitureAsync metódus implementálása

A harmadik a **UpdateFurnitureAsync** metódus, amely ez adatbázisba való egyidejű fölülírásáért felelős.

```
private async Task<UpdateResult> UpdateFurnitureAsync(string roomId, string furnitureId, Furniture furniture)
{
    var filter = Builders<Room>.Filter.Where(r => r.Id == roomId && r.Furnitures.Any(i => i.Id == furnitureId));
    var update = Builders<Room>.Update.Set(r => r.Furnitures[-1], furniture);
    return await _room.UpdateOneAsync(filter, update);
}
```

5.3 UpdateFurnitureAsync metódus implementálása

# 5.3. Front-End és UI felület kialakítása

A következő lépés a frontend kialakítása volt. Itt nehézséget a JavaScript nyelvben való teljes járatlanságom okozta. Először is meg kellet ismerkednem a nyelvel utána vághattam csak neki a feladatnak. Ebben a fejezetrészben a Front-End-en kialakítása során fölmerülő, általam fontosnak gondolt problémák és a rá született megoldások kerülnek tárgyalásra.

## 5.3.1. Front-End optimalizálása

A gördülékenyebb megjelenítés és a felhasználó élmény növelése érdekében fontos a felület optimalizálása. Ennek érdekében fontos a gyakorta használt műveleteknél a futási idő csökkentése. Ezért a fejlesztés során fölfedeztem pár olyan pontját a programnak, ahol érdemesnek találtam valamilyen optimalizálást bevezetni.

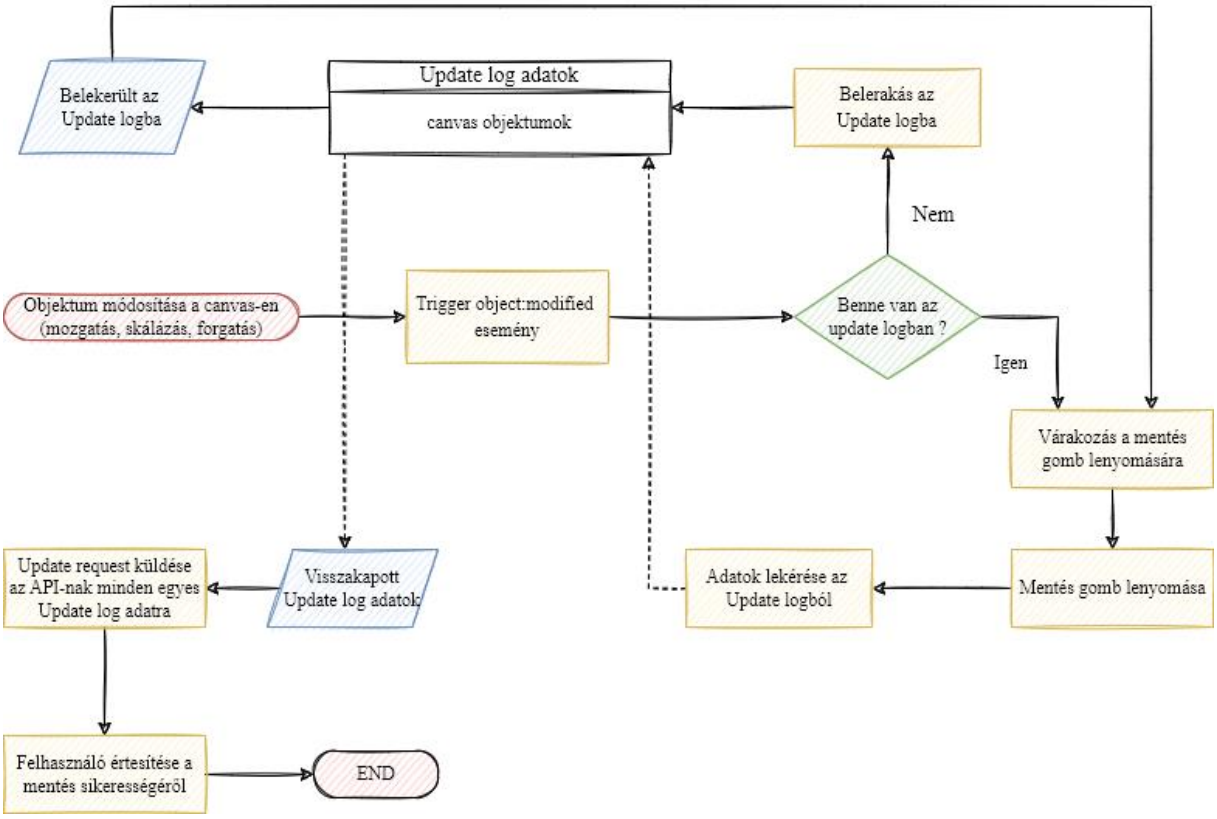
### 5.3.1.1 Mentés optimalizálása

Az egyik ilyen művelet, ahol jelentős futási időbeli csökkenés érhetünk el a mentés. Idáig a mentés úgy működött, hogy a mentés gomb lenyomásával a canvas-en lévő összes objektummal ki lett küldve egy API update hívás. A probléma itt az, hogy a hívás akkor is megtörténik, ha nincs is rá szükség. Gondoljunk csak bele, ha n darab objektum van lehelyezve a canvas-re és a felhasználó csak az egyiken módosít, abban az esetben is n darab API hívás történik. Ez egy borzasztó költséges művelet, még egy felhasználó esetében is. Abban az esetben, ha több felhasználóról beszélünk ezzel a megoldással az API hívások száma n*n, amely közel sem elhanyagolható.

Ennek a problémának megoldására született az az ötlet, hogy csak abban az esetben indítsunk update API hívást, ha szükséges. Tehát ha n darab canvas objektumból csak egyen módosít a felhasználó abban az esetben 1 darab API hívás történjen az n helyett.

Tehát a futási idő legjobb esetben lecsökkenthető O(n)-ről O(1) -re. Legrosszabb esetben marad O(n). Összeségében ezzel a módosítással a futási idő tehát O(n)-ről O(max(n)) -re változik.

Az alábbi ábrán ennek a műveletnek a folyamat ábrája látható.

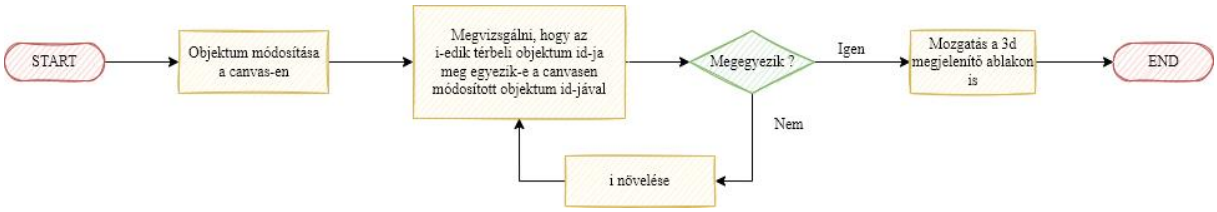



5.4. ábra Save művelet optimalizálásának folyamat ábrája

### 5.3.1.2 Canvas és 3D megjelenítés összhangba hozása és a folyamat optimalizálása

A felhasználó élmény növelése és a felület egyszerűsítése érdekében úgy döntöttem, hogy a User Interface-ről lekerül a 3D renderer gomb. Eddig úgy történt a térbeli megjelenítés, hogy a canvas-en való módosítás után rá kellet nyomni a 3D renderer gombra majd a felület jobb oldalán megjelent a renderert szoba. Helyette úgy gondoltam, hogy sokkal szebb megoldás, ha a 3D megjelenítést dinamikusan valósítom meg. Tehát ahogy a canvas módosul úgy vele egy időben a 3D megjelenítés is módosuljon.

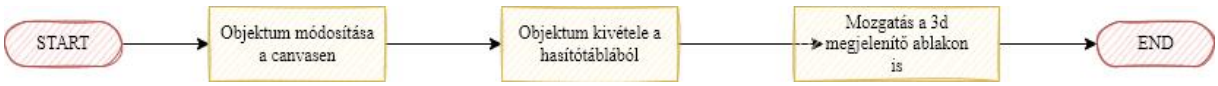
Ehhez azonban szükséges, hogy a canvas-en lévő objektum esetében megkeressük, hogy melyik térbeli objektumnak felel meg. Ez uuid alapján beazonosítható. Ezzel kapcsolatban viszont felmerül még egy optimalizálási probléma. Mégpedig az objektum megkeresése a 3D scene-en. Ennek egy megoldási folyamata látható az alábbi ábrán.




5.5. ábra Optimalizálatlan 3d scene-ben való megjelenítés folyamat ábrája

Ez a fajta megoldás legrosszabb esetben végig iterál minden egyes térbeli objektumon minden egyes alkalommal amikor egy millimétert is mozgatunk az objektumom a canvas-en. Ezt azért érezzük, hogy borzasztó költséges művelet. Így a futási idő módosítás esemény hívásonként O(max(n)).

Ennek a problémának a megoldására született az a az ötlet, miszerint tároljuk el egy hasító táblába, hogy melyik canvas objektumhoz melyik térbeli objektum tartozik. Pontosan azért célszerű a hasító tábla használata, mivel a benne való keresés O (1). Ugyan memória használatot növeli, viszont ezt az áldozatot, úgy gondolom megéri vállalni. Ezzel jelentős futási idejű csökkenést érhetünk el, ugyanis a futási idő módosítás esemény hívásonként O(max(n)) -ről O(1) -re módosul. Ez esetben az alábbi módon változik a folyamat ábra.

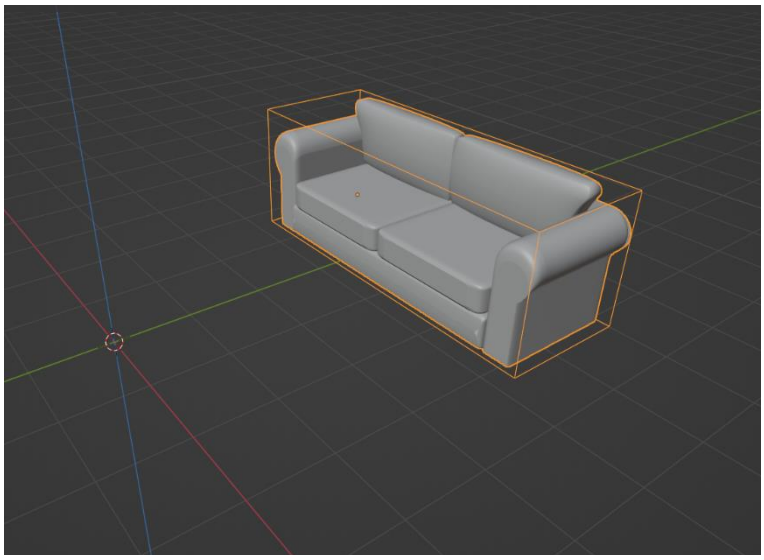



5.6. ábra Optimalizált 3d scene-ben való megjelenítés folyamat ábrája

Jól látható, az elöl tesztelő ciklust fölváltotta egy hasítótábla.

## 5.3.2. 3D modellek gyűjtése és skálázásuk

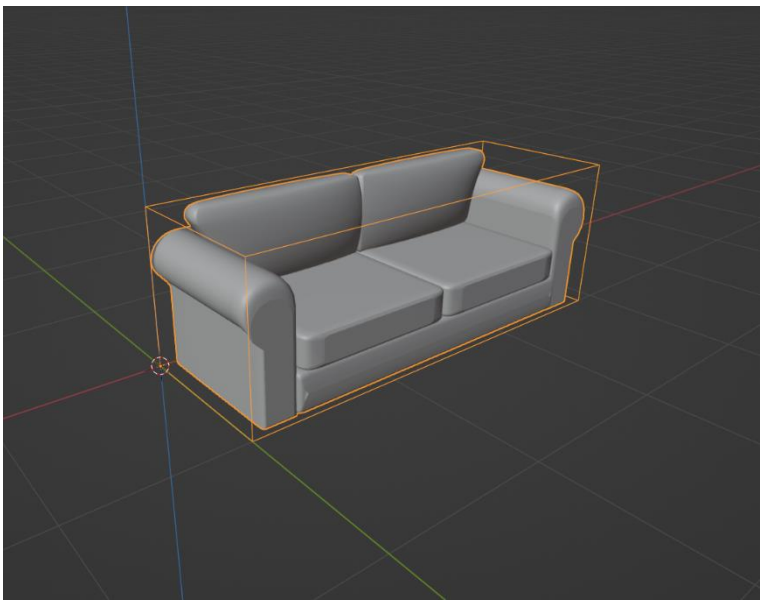
A 3D modellek gyűjtése és importálása önmagában nem hangzik nehéz feladatnak. Viszont ahhoz, hogy ezeket a modelleket megfelelően tudjuk importálni a three.js segítségével el kell végezni néhány konverziót, ami közel sem volt triviális. Ahhoz, hogy a megfelelő koordinátába helyeződjön le egy térbeli modell, ahhoz két feltételnek kell teljesülnie. Az egyik, hogy az objektumot legkisebb térfogatban körülvevő téglatest (vagy másnéven bounding box – az 5.3-as ábrán narancssárgával körvonalazott téglatest) sarkának pontosan az origóra kell illeszkednie (Az origó az 5.3-as ábrán piros-fehér szaggatott vonallal jelölt kör). A másik, hogy pontosan ugyanakkora méretűnek kell lennie szélességre, hosszúságra, magasságra, mint a canvas-en lévő modell (ebből adódik egyébként a skálázhatósági probléma, amire majd későbbiekben kitérek). Az alábbi ábrán egy rosszul elhelyezett és skálázott modell látható.



5.7 Rosszul pozícionált és skálázott modell

Az interneten fellelhető ingyenes modellek többségével az a nagy probléma, hogy nem felel meg ennek a két feltételnek, tehát az egyik megoldás, hogy valamilyen szoftveres segítséget kell segítségül hívni a helyes pozíció és méret beállításához. A másik megoldás a three.js-ben való modifikálás, viszont utána járva ez egy sokkal nehezebb kivitelezni, így az első opciót választottam.

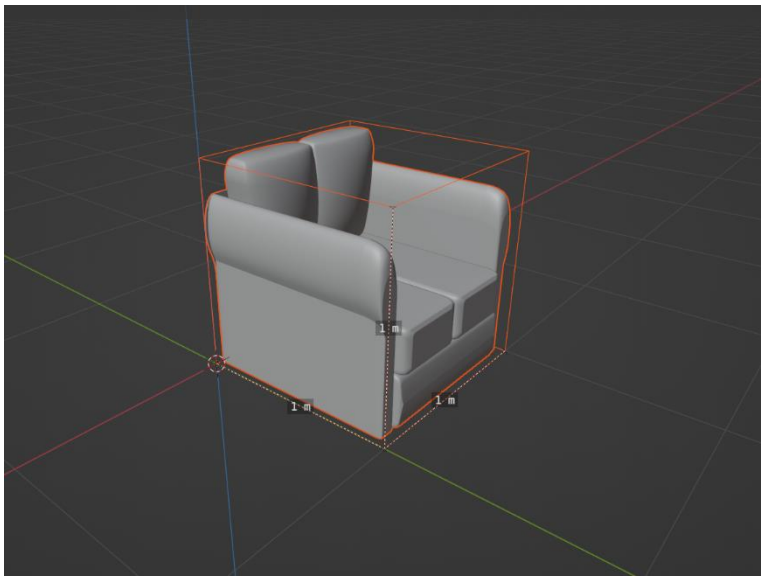
A helyes pozíció beállítására kezdetben az egyik leghíresebb ingyenes 3d modellező szoftvert használtam, a Blendert. Rengeteg próbálkozás után kiderült, hogy a Blender közelítőlegesen igen, viszont pontos pozícionálásra nem alkalmas. Feladatom szempontjából a közelítőleges pozícionálás nem opció, hiszen a bútorok elrendezésénél vagy az ütközés vizsgálatánál minden milliméter számít. Mindezek mellet a Blenderben pontos átméretezésre nincs lehetőség csak skálázni lehet, amely segítségével a pontos átméretezés nem megoldható. Más alternatívák után kellet hát kutatnom, amelyben ezek a funkciók elérhetők. Hosszas keresgélés után a választásom az Adobe Dimension-re esett, melyben lehetőség van cm-re pontos átméretezésre és pozícionálásra. Az Adobe Dimension-ben való átpozícionálás után az alábbi képen a térbeli objektum látható már a helyes pozícióban (tehát bounding boxának sarkával az origóba.illesztve).



5.8 Rosszul skálázott, helyesen pozícionált modell

A másik probléma a skálázhatóságot illetően adódott. Hiszen a bútorok átméretezésére az egyetlen opció three.js-ben a skálázás. Nem lehet tehát fix méretet megadni, csak egy adott méretet valahányszorosára növelni vagy csökkenteni. Így muszáj valamilyen megoldást találni, hogy a skálázás segítségével lehessen megoldani az átméretezhetőségi problémát.

Végül az a megoldás született, hogy minden egyes háromdimenziós modell szélessége, magassága és hosszúságát pontosan 100cmre állítom be. Ez látható az 5.5-ös képen.




5.9 Helyesen skálázott és pozícionált modell

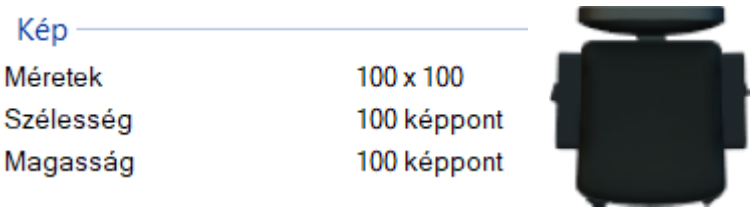
Majd amikor megjelenítésre kerül, akkor a fölskálázódik x,y,z irányba, a bútor méret osztva 100-al. (a bútorok mérete cm-ben tárolódik az adatbázisban). A mentés műveletnél pedig ugyanezt a konverziót kell végrehajtani. Ez látható az alábbi ábrán, ahol a furnitureModel maga a térbeli objektum a data pedig az adatbázisba elmentett adat.

```
furnitureModel.scale.set(data.depth / 100, data.height / 100, data.width / 100);
```

## 5.3.3. Canvas objektumok skálázása

Az előző fejezetrészben tárgyalt skálázhatósági probléma a canvas objektumok átméretezésénél szintén előjön. Az általam használt JavaScript könyvtárban a Fabric.js-ben szintén nincs lehetőség egzakt átméretezésre. A méretbeli modifikálásra az egyetlen opció a skálázás. Itt a probléma annyiban tér el, hogy nem 3d-s modellekről beszélünk, hanem .PNG fájlokról, aminek csak szélessége és hosszúsága van.

Itt szintén azt a megoldást alkalmaztam, hogy minden egyes .PNG fájl-t egységes 100 cm-es szélességre és hosszúságra lett beállítva. Fontos megjegyezni, hogy 1 pixel-t 1 cm-nek feleltetek meg.




5.10 PNG fájl 100x100 pixelre átskálázva

Fabric.js-en belül a skálázási konverzió a térbeli modellekhez hasonlóképpen az alábbi képlettel áll elő:

**skálázás mértéke = adatbázisba elmentett bútor hossza / 100**
**hosszúság = hosszúság * skálázás mértéke**

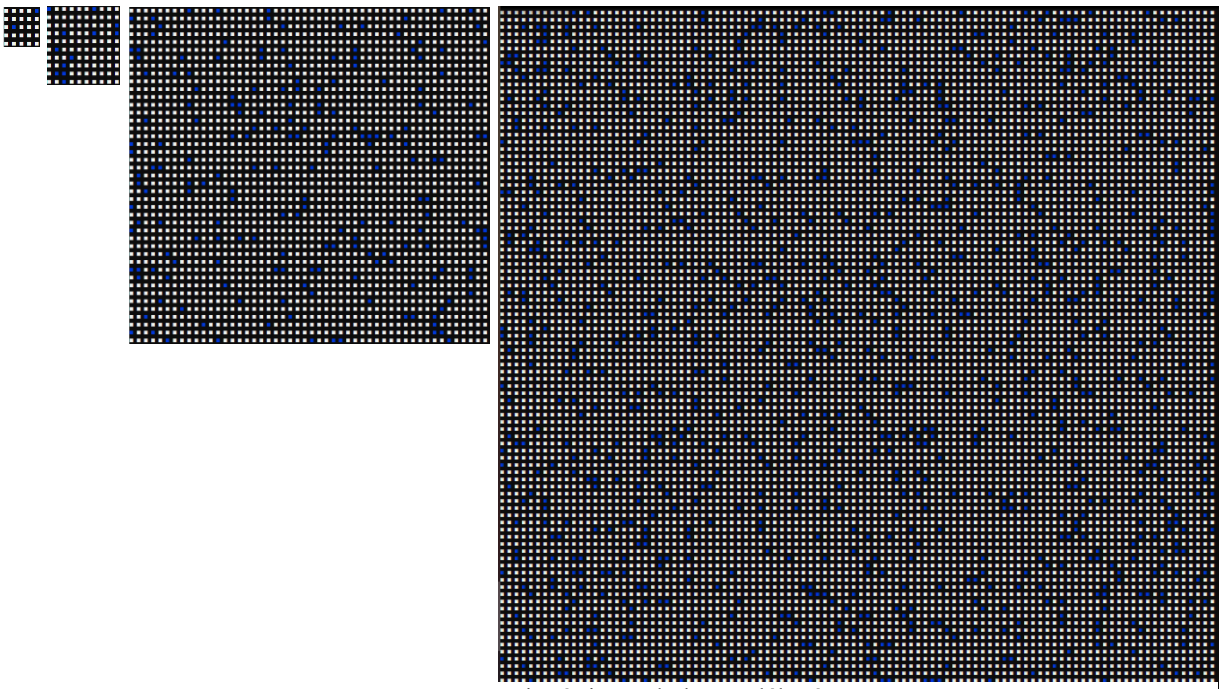
Mentésnél pedig az objektum hossza az alábbi képletből áll elő:

**adatbázisba elmentett bútor hossza = 100 * skálázás mértéke**

## 5.4. Maximális téglalap alakú terület meghatározási módszereinek implementálása

A 2.2-es fejezetszámú használt módszerek és technikák résznél, szó esett különféle terület optimalizálási algoritmusokról. Ezeket ültettem gyakorlatba és teszteltem futási idejüket NxN méretű logikai tömbökön folyamatos N növelést követően. Érdemesnek tartottam ezeket megvizsgálni, hogy az elméletben foglaltak gyakorlatba is átültessem, ezzel számszerű bizonyítékot adva teljesítő képességükre.

Felmérésem során az N a következő értékeket vette föl {5, 10, 50, 100, 256, 378, 500, 1000, 5000, 10000}. A logikai tömb értékeit random sorsortam ki. Hamis érték fölvételére 5% eséllyel, igaz érték fölvételére 95% eséllyel. A bemeneti tömb közül szemléltetés képen az N = 5,10,50,100 méretűek láthatóak az alábbi ábrán. Az igaz logikai értékek fehérrel, míg a hamisak sötétkékkel szerepelnek.



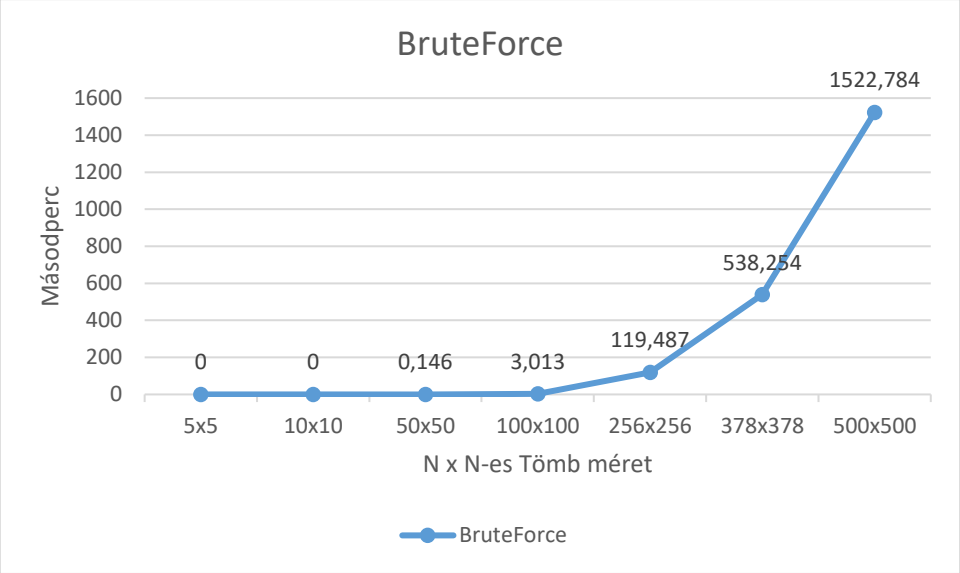
5.11 Bemeneti mátrix tömbök szemléltetése

Esőként a **nyers erő** módszerét mértem föl melynek, a már C#-ban implementált verziója az alábbi ábrán látható.

```
static void BruteForce(ref (int, int) best_ll, ref (int, int) best_ur)
{
    (int, int) ll;
    (int, int) ur;

    for (ll.Item1 = 0; ll.Item1 < b.GetLength(0); ll.Item1++)
    {
        for (ll.Item2 = 0; ll.Item2 < b.GetLength(1); ll.Item2++)
        {
            for (ur.Item1 = 0; ur.Item1 < b.GetLength(0); ur.Item1++)
            {
                for (ur.Item2 = 0; ur.Item2 < b.GetLength(1); ur.Item2++)
                {
                    var asd1 = area(ll, ur);
                    var asd2 = area(best_ll, best_ur);
                    if (asd1 > asd2 && all_ones(ll, ur))
                    {
                        best_ll = ll;
                        best_ur = ur;
                    }
                }
            }
        }
    }
}

1 reference
static bool all_ones((int, int) ll, (int, int) ur)
{
    for (int x = ll.Item1; x <= ur.Item1; x++)
    {
        for (int y = ll.Item2; y <= ur.Item2; y++)
        {
            if (b[x,y] == false)
            {
                return false;
            }
        }
    }
    return true;
}

7 references
static int area((int, int) ll, (int, int) ur)
{
    if (ll.Item1 > ur.Item1 || ll.Item2 > ur.Item2)
        return 0;
    else
        return (ur.Item1 - ll.Item1 + 1) * (ur.Item2 - ll.Item2 + 1);
}
```

5.12 BruteForce C# implementáció

Az alábbi algoritmus teljesítményét mérve az alábbi grafikont kapjuk.




5.13 BruteForce algoritmus mérési eredményei

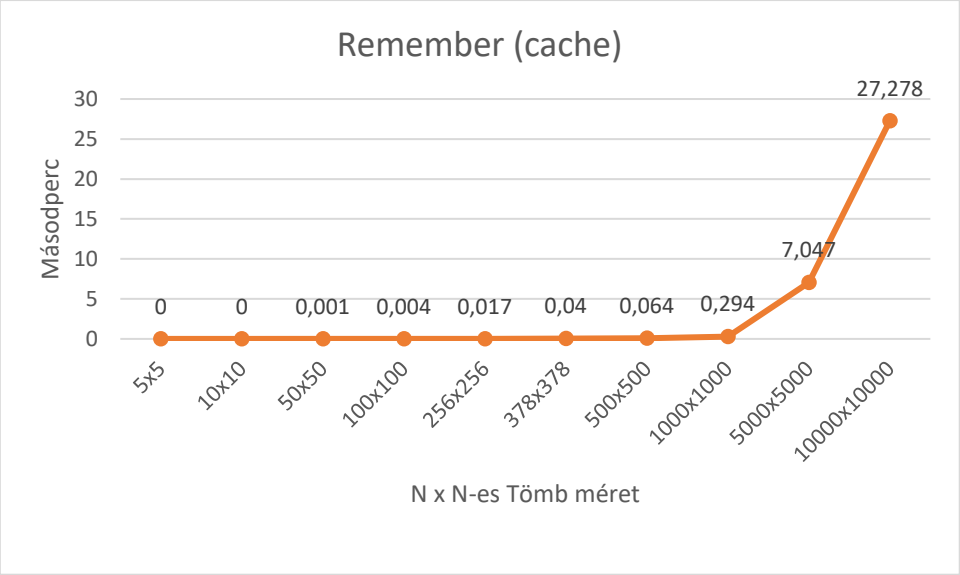
Megfigyelhető, hogy a futási idő az N növekedésével exponenciálisan nő, oly mértékben, hogy a 378x378-as tömbméretnél az algoritmus 8,9709 percig futott, míg 500x500 méretnél ez az érték már 25.36667 perc volt. A várható exponenciális növekedés miatt az 1000x1000, az 5000x5000 és a 10000x10000-es tömbméretekkel már nem futtattam.

Másodjára az **emlékező** (Remember) algoritmus futtattam, melynek implementációja az alábbi ábrán látható.

```
static void Remember(ref (int, int) best_ll, ref (int, int) best_ur)
{
    (int, int) ll;
    (int, int) ur;

    for (ll.Item1 = N - 1; ll.Item1 >= 0; ll.Item1--)
    {
        update_cache(ll.Item1);
        for (ll.Item2 = 0; ll.Item2 <= M - 1; ll.Item2++)
        {
            ur = grow_ones(ll);
            if (area(ll, ur) > area(best_ll, best_ur))
            {
                best_ll = ll;
                best_ur = ur;
            }
        }
    }
}

2 references
static void update_cache(int ll_X)
{
    for (int y = 0; y <= M - 1 ; y++)
    {
        c[y] = b[ll_X, y] ? ++c[y] : 0;
    }
}

1 reference
static (int, int) grow_ones((int, int) ll)
{
    (int, int) ur = (ll.Item1 - 1, ll.Item2 - 1);
    int x_max = N;
    int y = ll.Item2 - 1;

    while (y + 1 < M && b[ll.Item1, y + 1] != false)
    {
        y++;
        int x = Math.Min(ll.Item1 + c[y] - 1, x_max);
        x_max = x;
        if (area(ll, (x, y)) > area(ll, ur))
            ur = (x, y);
    }
    return ur;
}

7 references
static int area((int, int) ll, (int, int) ur)
{
    if (ll.Item1 > ur.Item1 || ll.Item2 > ur.Item2)
        return 0;
    else
        return (ur.Item1 - ll.Item1 + 1) * (ur.Item2 - ll.Item2 + 1);
}
```

5.14 Remember metódus C# implementációja

A futási eredmény mérése során érdekes meghökkentő eredmények születtek. Ugyanis ahogy az ábrán is látszik, már az 500x500-as tömbön mért futási idő 23793,5-szeres növekedést jelent a nyers erő algoritmusához képest. Ez az algoritmus is exponenciálisan nő viszont nagyon jelentős különbség van a függvény felfutása között.



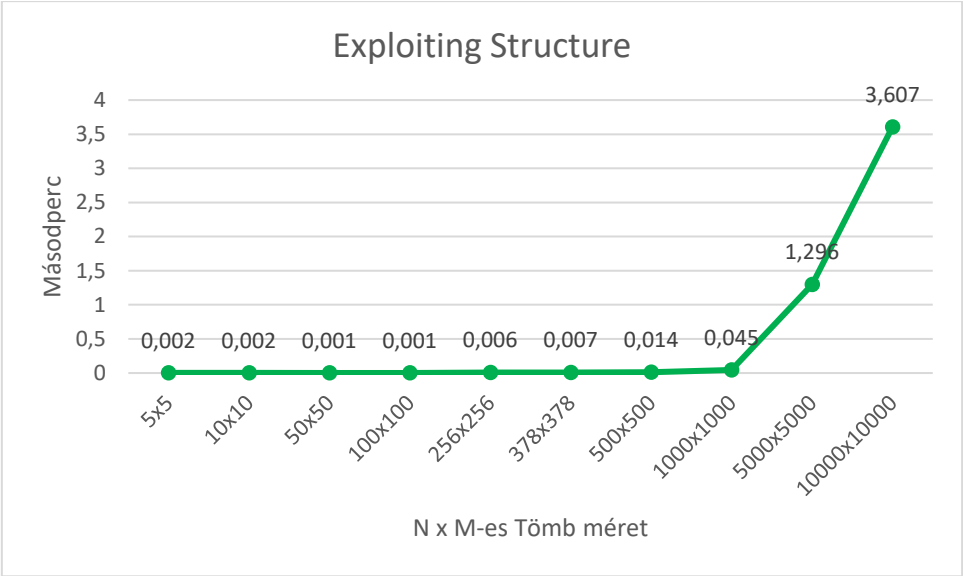

5.15 Remember metódus mérési eredményei

Végül a kihasználó szerkezet (exploiting structure) került utoljára elemzésre. Mely az előző algoritmusokhoz képest mutat némi teljesítménynövekedést. Az algoritmus megvalósítása az alábbi képen látható.

```
static void ExploitingStructure(ref (int, int) best_ll, ref (int, int) best_ur)
{
    int best_area = 0;

    for (int x = 0; x < N; ++x)
    {
        int width = 0;
        update_cache(x);
        for (int y = 0; y < M; ++y)
        {
            if (c[y] > width)
            {
                s.Push((y, width));
                width = c[y];
            }
            if (c[y] < width)
            {
                int y0, w0, area;
                do
                {
                    (y0, w0) = s.Pop();

                    area = width * (y - y0);
                    if (area > best_area)
                    {
                        best_area = area;
                        best_ll = (y0, x);
                        best_ur = (y - 1, x - width + 1);
                    }
                    width = w0;

                } while (c[y] < width);

                width = c[y];

                if (width != 0)
                    s.Push((y0, w0));
            }
        }
    }
}
```

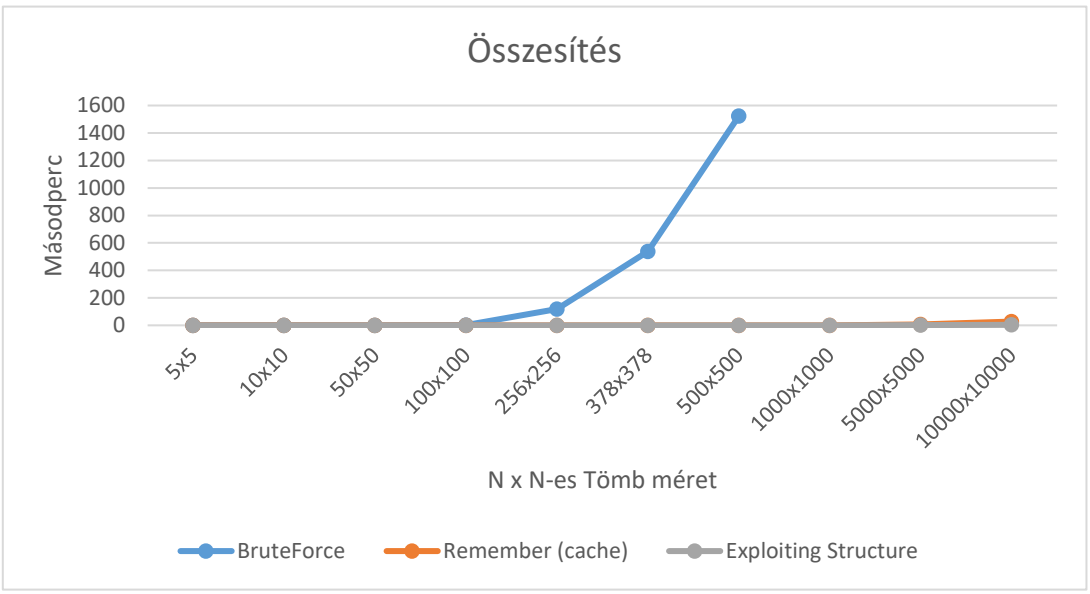
5.16 Exploiting Structure C# implementációja

A függvény futási ideje, még így is exponenciális viszont a háromfajta variáció közül még így is ez magasan a leggyorsabb. A futási időről készült grafikon az alábbi ábrán látható.



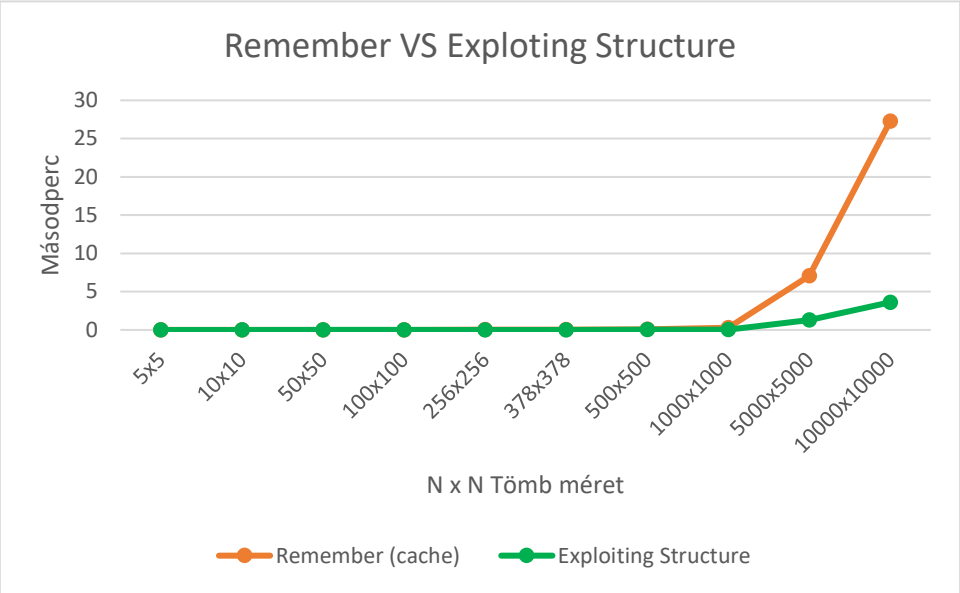

5.17 Exploiting Structure mérési eredményei

Végül összesítsük az eredménykét, a drasztikus különbségek demonstrálása érdekében.




5.18 Összesített mérési eredmények

Az alábbi grafikon a három módszer futási idejét összesíti egy grafikonon, ahol megfigyelhető, hogy a nyers erő módszere az N növekedésével drasztikusan elmarad futási időben a másik két módszertől.




5.19 Remeber és Exploiting Structure összehasonlítása

Szintén érdemes megfigyelni, hogyha csak a Remember és az Exploting Structure módszert hasonlítjuk össze, megfigyelhető, hogy az Exploiting Structure függvénynek drasztikusan kisebb a felfutása. Bár a módszerek közti futási idő különbség a végtelenbe tartva egyre nagyobb, esetünkben nem valószínű, hogy bizonyos értékek fölött is alkalmazásra kerül az algoritmus.

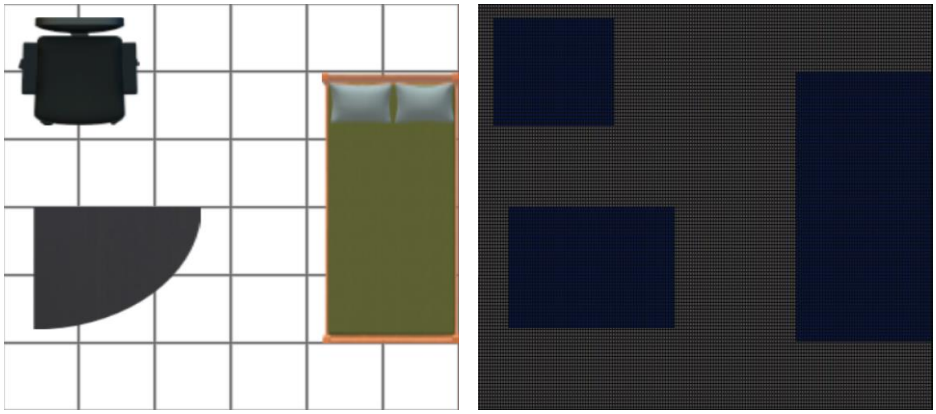
## 5.5. Bit map konverzió

A maximális téglalap alakú teret már megtudjuk határozni, viszont, ahhoz, hogy ezt a feladatunkban is tudjuk alkalmazni, bemeneti logikai mátrix tömbként (, amit mostantól bit map-ként fogok rövidíteni) egy olyan bitmap átadása szükséges, amely pontosan akkora méretű, mint az adott szoba és pontosan ugyanabban a pozícióban, abban a méretben szerepelnek benne a bútorok, ahogy utoljára mentve lettek. Ezt a konverziót a Back-End végzi el, így semmi másra nincs szükségünk csak az aktuális szoba azonosítójára és onnantól fogva minden adat rendelkezésünkre áll a konverzió lebonyolításához. Maga a konverzió egyébként nem túl bonyolult, az implementációja az alábbi képen látható.

```
static bool[,] BitMapCreator(Room room)
{
    bool[,] BitMap = new bool[room.WidthX, room.WidthY];

    foreach (var f in room.Furnitures)
    {
        for (int x = f.PositionX; x < f.PositionX + f.Width; x++)
        {
            for (int y = f.PositionY; y < f.PositionY + f.Depth; y++)
            {
                BitMap[x, y] = false;
            }
        }
    }

    return BitMap;
}
```

5.20. Bitmap konverzió C# implementáció

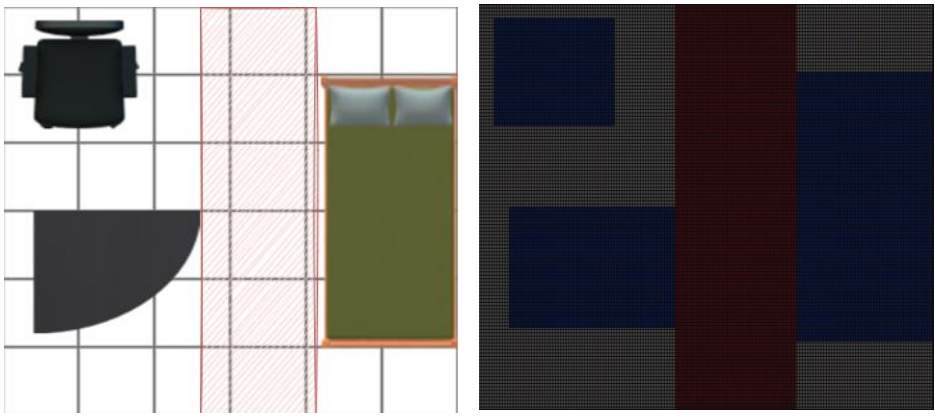
A BitMapCreator függvény futtatását követően az alábbi ábrán látható a konverzió bemenete



5.21 BitMap konvezió bemenete

és a hozzá tartozó elvárt kimenet.

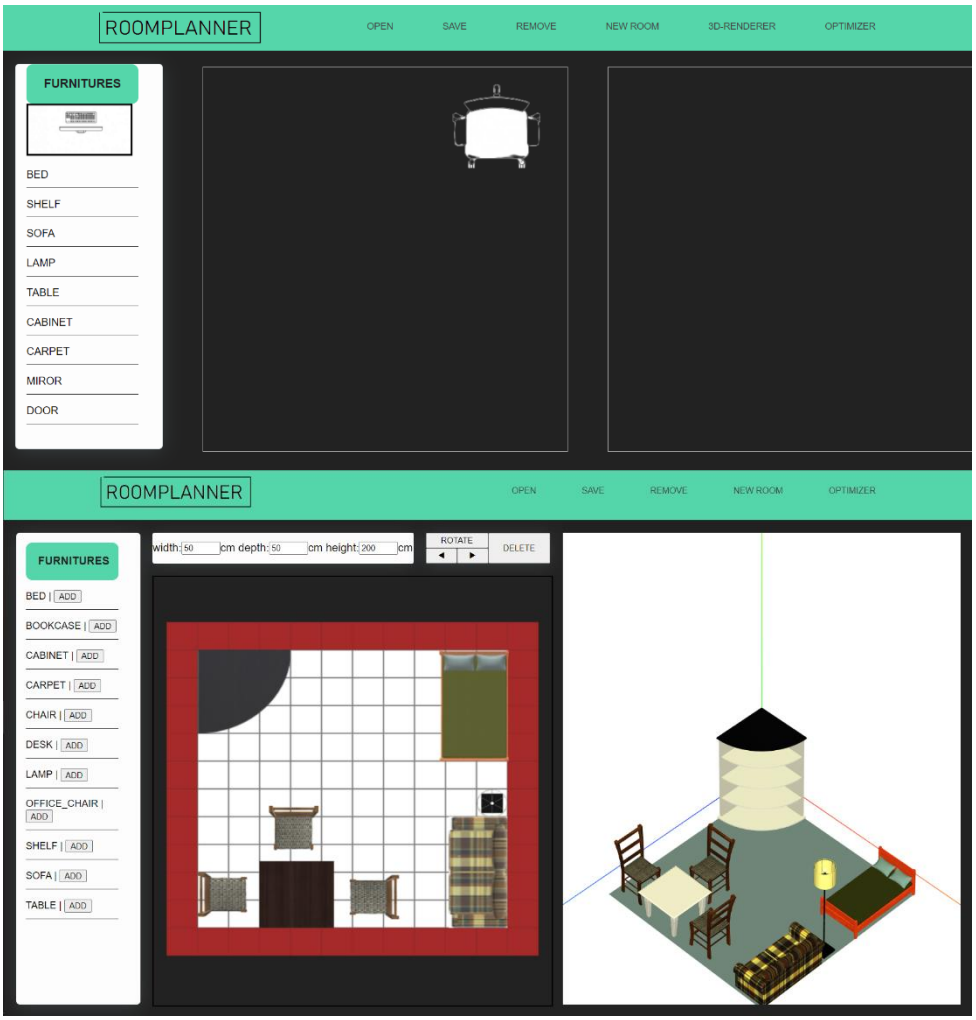
Ezen a bitmapon futtatva a maximális téglalap alakú területet kereső algoritmust az alábbi kimenetet kapjuk.



5.22 BitMap konverzió kimenete

# 5.7. UI kinézete

A User Interface jelenlegi kinézete az 5.7-es kép alsó felében látható, a felső felében pedig a pilot verzió. A kép azt hivatott bemutatni, hogy a UI milyen változtatásokon ment keresztül a Pilot verzióhoz képest.




5.23 UI kinézet változás a pilot verzióhoz képest

Mint látható a UI sok változáson ment keresztül, a html kód teljes egészében refaktorálásra került. Az addigi pixel szerinti elhelyezések dinamizálva lettek, grid szerkezetek váltották le őket, így a böngésző ablak méretezésére is jól tud reagálni az alkalmazás. Eredetileg drag and droppos bútorbehúzásra lett tervezve a webalkalmazás, viszont ez idő hiánya miatt nem valósult meg, helyette, bekerült minden bútor mellé egy add gomb. Felhasználó barátibb megoldás lenne a drag and drop viszont szerintem az átláthatóságon nem sokban ront. A bútorok könnyebb modifikálása érdekében belekerült még egy rendszersáv , amely a bútorok méretét monitorozza és a bútorokat ezen keresztül lehetőség van számbeli formátumban is módosítani. Valamint a magasság módosítását máshogy nem is lehetett kivitelezni, személy szerint ezt találtam a legelegánsabb megoldásnak. Szintén bevezettem egy forgatás gombot két nyíllal mellyel jobbra

és balra lehet forgatni. Hozzá lett adva egy törlés gomb is, amivel lehet már bútort is törölni, a sikerességét egy fölugró ablak jelzi.

A UI kialakításánál fontos szempont volt az egyszerűség, az áttekinthetőség és az átláthatóság, mivel alapvetően otthoni felhasználásra, mindenki számára egyértelművé és kezelését könnyen elsajátíthatóvá szerettem volna tenni.

# 6. Tesztelés

A tesztelési fázis fontos részét képezi a programnak, ugyanis itt bizonyosodhatunk meg róla, hogy amit lefejlesztettünk valóban működik. A Back-End CRUD és optimalizáló algoritmusát unit tesztel, a Front-End-et főként manuális tesztekkel teszteltem. A tesztelést öt különböző pont alapján értékeltem. Az első a követelmény, ami azt foglalja magába, hogy mit várunk el a tesztelendő funkciótól. A Tesztleírás leírja a konkrét tesztesetet, hogy milyen szcenárióknak következhetnek be és hogy mit tesztelünk. A hívás lecsapódása, hogy az adott metódus hívás a program mely pontját éri el. A mérés és cél, arra ad választ, hogy minek kell lennie az elvárt kimenetnek, az eredmény pedig azt mutatja meg, hogy ez valóban teljesült-e.

| | Követelmények | Tesztleírás | Hívás lecsapódása | Mérés és cél | Eredmény |
|---|---|---|---|---|---|
| **1** | A felhasználó legyen képes új szoba létrehozására. | Új szoba mentése az adatbázisba az interfészen megadott adatokkal. | API és Front-End | Új szoba megjelent az adatbázisba, a canvas-en és a 3d-s grafikai felületen. | ✓ |
| **2** | A felhasználó legyen képes már meglévő szoba törlésére. | Kiválasztott szoba törlése a legördülő listából, a canvas-ről, a 3d-s grafikai felületről és az adatbázisból. | API és Front-End | Szoba törlődött az adatbázisból, a legördülő listából, a canvas és a 3d-s grafikai megjelenítő elemei törlődtek. | ✓ |
| **3** | A felhasználó legyen képes már meglévő szoba megnyitására. | Ha volt addigi szoba megnyitva akkor az bezárul és helyére betöltődik a kiválasztott szoba az adatbázisbéli | API és Front-End | Ha volt megnyitott szoba akkor az elemei törlődtek a canvas-ről és a 3d-s grafikai megjelenítőről. Helyére az API-tól | ✓ |

| | | | | | |
|---|---|---|---|---|---|
| **4** | A felhasználó legyen képes bútor hozzáadásra a szobához | Ha van megnyitva szoba akkor a bútor jelenjen meg a szerkesztő és a 3d-s felületen, ha nem akkor hibaüzenet | API és Front-End | Ha van megnyitva szoba, akkor a bútor hozzáadódik az adatbázison belül az adott szobához, ha nincs akkor hibaüzenet. | ✓ |
| **5** | A felhasználó legyen képes kiválasztott bútor törlésére | Bútor törlése az adatbázisból, a canvasról, a 3d-s grafikai felületről. | API és Front-End | A bútor törlődött az adatbázisból a canvasról, a 3d-s grafikai felületről. | ✓ |
| **6** | A felhasználó legyen képes a bútor méretének módosítására a canvas-en | A bútor skálázódik a megfelelő méretre canvasen és a 3d-s grafikai felületen | Front-End | A bútor mérete megváltozott a felhasználó által beállítottakra, a változások megjelentek a felületen és | ✓ |
| **7** | A felhasználó legyen képes a bútor méretének módosításra a szerkesztő sávon | A kijelölt bútor skálázása a beállított méretre | Front-End | A bútor mérete módosult és ez megjelenik a canvas-en és a 3d-s grafikai felületen és belekerül a log tömbbe. | ✓ |
| **8** | A felhasználó legyen képes a bútor 90°-os forgatására az óramutató járásával megegyező | A kijelölt bútor elforgatása | Front-End | Megváltozik a bútor, szöge és ez a változás megjelenik a canvas-en és a 3d-s felületen, valamint belekerül az objektum a log | ✓ |
| **9** | A felhasználó legyen képes a bútor 90°-os forgatására az óramutató járásával ellentétes | A kijelölt bútor elforgatása | Front-End | Megváltozik a bútor, szöge és ez a változás megjelenik a canvas-en és a 3d-s felületen, valamint belekerül az objektum a log | ✓ |

| | | | | | |
|---|---|---|---|---|---|
| **10** | A felhasználó legyen képes a bútorok áthelyezésére | Módosul a kijelölt bútor pozíciója | Front-End | Módosul a kijelölt bútor pozíciója és ez a canvas-en és a 3d-s felületen is megjelenik, valamint az objektum belekerül a log | ✓ |
| **11** | A felhasználó legyen képes a szoba elmentésére | A szobában elmentésre kerül minden módosítás | API | A log tömbben lévő összes módosítás elmentésre kerül az adatbázisba, majd a log tömb tartalma törlődik | ✓ |
| **12** | A bútorok az oldal betöltését követően megjelennek a menüsávban | A bútorok lekérése az adatbázisból és megjelenítése a menüsávon | API és Front-End | Bútorok lekérdezése az adatbázisból visszaadása és megjelenítése a menüsávban | ✓ |
| **13** | Felhasználó legyen képes az optimális bútorelrendezés lekérdezésére | Optimalizálási eljárás futtatása | API | Visszakapott megoldás megjelenítése és validációja amennyire lehetséges | ✓ |
| **14** | Üzleti logika tesztelése, szimulált környezetben (CRUD tesztek) | Unit tesztek futtatása | API | Üzleti logika kód lefedettség nagyobb mint 80% | ✓ |
| **15** | A webalkalmazás képes legyen visszajelzés küldésre a felhasználónak | Felugró ablakok helyességének ellenőrzése | Front-End | Helyes hibaüzenetek | ✓ |

| | | | | | |
|---|---|---|---|---|---|
| **16** | Lehessen a felülnézetes szerkesztőfelületen zoomolni | Canvasan egér görgetés hatására nagyítás megy végbe | Front-End | Canvas elemek mérete megnő a UI-on | ✓ |
| **17** | Lehessen a felülnézetes szerkesztőfelületen egér mozgással kamerát manipulálni | Egérmozgatás hatására szerkesztő felület kamerapozíciója megváltozik | Front-End | Megváltozik a kamera pozíció | ✓ |
| **18** | Több bútor kijelölése egyszerre | Több bútort lehessen egyszerre modifikálni | Front-End | Kijelölt bútorok egyszerre történő modifikálása megjelenik a UI-on és belekerül az update logba | ✗ |
| **19** | 3d megjelenítő kamera pozíciójának változtatása | Kamera pozíciója kurzor események hatására megváltozik. | Front-End | Egér pozíció változásra reagál a 3d megjelenítés | ✗ |

5.24 Tesztelés összefoglaló táblázat

# 7. Értékelés

Szakdolgozatom során a cél egy webalkalmazás létrehozása volt, amely alkalmas szobák berendezésére, figyelve arra, hogy a felhasználói igényeket a lehető legjobban kielégítse. A fejlesztés első szakaszában egy CRUD műveletek ellátására alkalmas C# Back-End API-t próbáltam meg kialakítani. Kezdetben az adatbázist Microsoft Sql Serverrel próbálkoztam, viszont rádöbbentem, hogy fölöslegesen bonyolítaná túl a fejlesztést, hiszen nincs szükségem adattáblákra és közöttük lévő kapcsolatra ezért váltottam a Mongo DB felhő alapú szolgáltatására a MongoDB Atlasra, mely valamiért csak 1GB-es internet sebességgel működött megfelelően, egyébként folyamatosan timeout-olt minden egyes api kérésnél. Hosszas utánajárás után azt találtam, hogy ez egy bug amivel tisztában vannak a fejlesztők, ezért váltottam a Mongo DB lokális verziójára a MongoDB Compassra. Azóta nem tudom, hogy még mindig fönn áll-e ez a probléma, viszont mivel lokális verzió remekül bevált már nem is volt akkora prioritású változtatás. Alapjáraton szerettem volna felhő alapú adatbázist, ez sajnos nem valósult meg de a jövőben mindenképp megszeretném ezt a lépést megtenni. Ezek után a Front-End implementáláshoz fogtam neki, melyet JavaScriptben írtam. Kezdetben a JavaScript nyelv tanulásával indítottam, ugyanis semmilyen tapasztalatom nem volt a nyelvel kapcsolatban. Így hát az alapok elsajátítása után állhattam csak neki a feladatnak. Utólag jó döntésnek gondolom, egészen mélyen sikerült megismernem a nyelvet és az egyes könyvtárait. A nyelv és az egyes JavaScript könyvtárak elsajátítása után egészen gyorsan haladat a felülnézetes szerkesztőfelület fejlesztése. A 3d grafikus felület fejlesztésével, már annál inkább több gond adódott, mivel a Three.Js dokumentációból alapján írt kódok sem működtek és nem tudtam nagyon sokáig rájönni, hogy mi lehet a probléma. Sok keresgélés után megtaláltam, hogy a könyvtáron belül kellet módosítani egy hibás osztály hivatkozást, minek után ez a probléma megoldódott. A következő nehézség az 5.5-ös fejezetben részletezett 3d-s modellek skálázása és hibás elhelyezésével adódott. Hosszas utána járás után ezt is sikerült megoldani az Adobe Dimension segítségével. Sajnálom, hogy a háromdimenziós kameramozgatás nem tudott megvalósulni idő hiányában viszont szerintem sokat növelne a felhasználói élményen ezért ezt mindenképp szeretném a jövőben megvalósítani. Eredetileg a renderelést egy dedikált gomb segítségével terveztem megoldani. Ez a kezdeti funkciókban meg is valósult viszont ez a rendezre gördülékenységét és felhasználó barátiságát nagy mértékben hátráltatta, azért végül a renderelést sikerült dinamikusan megoldani, hogy a felülnézeti szerkesztő felülettel egyszerre mozogjon. Örülök neki, hogy ezt a funkciót az optimalizálásra odafigyelve sikerült bevezetni. Szerintem rengetget hozzáad a felhasználó élményhez. Szerettem volna, hogy a Front-End valamilyen tervezési mintát kövessen, ami végül nem valósult meg, idő és tapasztalat hiányába. Ezt nagyon sajnálom hiszen szükség lenne flexibilitás szempontjából erre. Ez a továbbfejlesztési prioritásban úgy gondolom, hogy első helyen kell hogy helyet foglaljon. Ezután következett a fejlesztésben az optimális helykihasználásra algoritmust találni, ekkor született meg a fejemben az 5.4-es fejezetben tárgyalt megoldás, amivel, mint kiderült, már az 1990-es években foglalkoztak, mint probléma és próbáltak rá megoldást keresni. Így ezt az algoritmust a Haladó Algoritmusok kurzuson tanult módszerekkel gördülékenyen tudtam implementálni. Hátulütője, hogy nem képes figyelni az objektumok felhasználhatóságának megőrzésére, viszont ez a továbbiakban egy elsőszámú prioritást élvező probléma, hiszen így előfordulhat, hogy olyan optimálisnak vélt elrendezést kapunk eredményül, mely a

gyakorlatban nem használható. Úgy gondolom viszont, hogy a mostani algoritmus egy nagyon jó alapot ad ennek, melynek a továbbfejlesztési potenciálja igen magas és még hasznosabb funkcióvá válhat. Továbbiakban ennek a funkciónak a továbbfejlesztése fogja adni a fő fókuszt viszont ez már a dolgozat keretein belül nem fog megtörténni.

# 8. Továbbfejlesztési lehetőségek

Továbbfejlesztés terén felmerült dolgok közül elsősorban a Front-End kód rétegzés teljes refaktorálása lenne, mivel nem sérti a Single Responsibilty elvét. A jövőre nézve a további fejlesztések megvalósítására úgy gondolom, hogy ez egy elengedhetetlen feltétel.

A prioritási rangsorban a 3d grafikai kameramozgatásának implementálása lenne, hiszen a felhasználói élmény növeléséhez úgy gondolom, hogy nagyon nagy mértékben hozzájárulna.

Szintén jó fontos lenne, hogy a szobák ne csak téglalap alakúak lehessenek, ugyanis a gyakorlatban gyakran előfordulnak más alakkal rendelkező terek.

Nagyon jó lenne kipublikálni valamilyen szerver bérlésen keresztül, hogy mindenki számára elérhető legyen.

Jó dolognak gondolnám, hogy az egyes bútorokat ne csak 90°-ba lehessen forgatni, hanem bármilyen szögbe. Ez eddig a tér optimalizálás bonyolultsága miatt nem valósulhatott meg.

Ebből következően hasznos lenne a téroptimalizálási eljárást továbbfejleszteni, tökéletesíteni, mivel ezt egy elég egyedi funkciónak gondolom, sok potenciált látok benne.

Figyelmet kellene még fordítani az apró UI hibajegyek kijavításával, ugyanis egy szemfüles felhasználó, el tudja érni a rendszer crash-t és a könyvtárakból adódó hibajegyeket.

Hosszútávú jövőt nézve jó lenne kiterjeszteni több szobás kompozíciók megalkotására. Valamint hosszútávú terv még a React.Js bevezetése, rengeteg hasznos UI funkció bevezetésére, amely nagymértékben egyszerűsítené a felhasználói felületet.

# Irodalomjegyzék


[1] eTesk, „SweetHome3D,” 28 03 2021. [Online]. Available: http://www.sweethome3d.com.

[2] eTesk, „SweetHome3D,” 28 03 2021. [Online]. Available: http://www.sweethome3d.com/support/forum/index.

[3] H. Mohamad A., „Relational and NoSQL Databases: The Appropriate,” The Institute of Electrical and Electronics Engineers, Muscat, Oman, 2022.

[4] „docs.oracle.com,” [Online]. Available: https://docs.oracle.com/cd/B19306_01/server.102/b14220/intro.htm. [Hozzáférés dátuma: 18 04 2022].

[5] A. R. Mustafina és I. S. Vershinin, „Performance Analysis of PostgreSQL, MySQL, Microsoft SQL Server Systems Based on TPC-H Tests,” Institute of Electrical and Electronics Engineers, Sochi, Russian Federation, 2021.

[6] L. Chaokui és Y. Wu, „The distributed storage strategy research of remote sensing image based on Mongo DB,” Institute of Electrical and Electronics Engineers, Changsha, China, 2014.

[7] . S. Vrinda és G. Sachin, „Basic NOSQL Injection Analysis And Detection On MongoDB,” Institute of Electrical and Electronics Engineers, Bhopal, India, 2019.

[8] E. Nitikorn, N. Santi és L. Wanchak, „The Graphics and Physics Engines for Rapid Development of 3D Web-based Applications,” Institute of Electrical and Electronics Engineers, Pattaya, Thailand, 2019.

[9] S. Mark és A. Kurt, in *The OpenGL Graphics System: A Specification*, The Khronos Group Inc., 2019, pp. 2-3.

[10] H. Stefan, „cannonjs.org,” [Online]. Available: https://github.com/schteppe/cannon.js/wiki.

[11] . M.-M. Julien, Babylon.Js Essentials, Birmingham - Mumbai: Packt Publisihng, 2016.

[12] D. Selman, „Java 3D Programing,” in *Java 3D Programing*, Manning Publications, 2002, pp. 4-7.

[13] Microsoft, „Microsoft,” 09 04 2021. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/api/system.windows.media.media3d?view=net-5.0.

[14] S. Sándor, „Hegymászó módszer,” in *Haladó Algoritmusok*, Budapest, 2022, pp. 15-19.

[15] S. Sándor, „Véletlen Optimalizálás,” in *Haladó Algoritmusok*, Budapest, 2022, pp. 27-29.

[16] S. Sándor, „Particle Swarm Optimalizálás,” in *Haladó Algoritmusok*, Budapest, 2022, pp. 89-92.

[17] B. Stroustrup, „The C++ Programming Language,” in *Third edition,* Reading, Addison-Wesley, 1997.

[18] T. H. Cormen, E. L. Charles és L. R. Ronald, „Introduction to Algorithms,” New York, McGraw-Hill, 1990.

[19] M. Orlowski, „A New Algorithm for the Largest Empty Rectangle Problem,” in *Algorithmica,* 1990, pp. 65-73.

[20] D. Vandevoorde, „Dr.Dobb's,” 1 április 1998. [Online]. Available: https://drdobbs.com/database/the-maximal-rectangle-problem/184410529#disqus_thread. [Hozzáférés dátuma: 24 11 2022].

[21] „DB Engines,” [Online]. Available: https://db-engines.com/en/ranking.

[22] „Normal distribution,” [Online]. Available: https://en.wikipedia.org/wiki/Normal_distribution.

[23] eTesk, „Archive3d,” 28 03 2021. [Online]. Available: https://archive3d.net/.

[24] eTesk, „SweetHome3DGallery,” 28 03 2021. [Online]. Available: http://www.sweethome3d.com/hu/gallery.jsp.

[25] s. business, „SmallBusiness,” 28 03 2021. [Online]. Available: https://www.thebalancesmb.com/best-interior-design-software-5024948#best-free-option-roomstyler-3d-home-planner.

[26] Ikea, „Ikea home and kitchen planner,” 07 04 2021. [Online]. Available: https://kitchenplanner.ikea.com/au/UI/Pages/VPUI.htm.

[27] Floorplanner, „Roomstyler,” 07 04 2021. [Online]. Available: https://roomstyler.com/3dplanner.

[28] Floorplanner, „Roomstyler,” 07 04 2021. [Online]. Available: https://roomstyler.com/press.

[29] Floorplanner, „Floorplanner,” 07 04 2021. [Online]. Available: https://floorplanner.com/.

[30] AutoCAD, „AutoDesk,” 07 04 2021. [Online]. Available: https://www.autodesk.hu/products/autocad/overview?term=1-YEAR.

[31] Google, „SketchUp,” 07 04 2021. [Online]. Available: https://www.sketchup.com/products/sketchup-pro.

[32] C. C. Home, „c-sharpcorner,” 09 04 2021. [Online]. Available: https://www.c-sharpcorner.com/uploadfile/mheydlauf/creating-a-simple-3d-scene-in-wpf/.

[33] „Hibernate,” 12 05 2021. [Online]. Available: https://hibernate.org/orm/.

# Ábrajegyzék

# Táblajegyzék