Szegedi Tudományegyetem

Informatikai Intézet

SZAKDOLGOZAT

Tóth Fanni

2023

Szegedi Tudományegyetem
Informatikai Intézet

Térbeli grafikával támogatott turisztikai információs webalkalmazás Angular környezetben

Szakdolgozat

Készítette:
Tóth Fanni
Gazdaságinformatikus BSc
szakos hallgató

Témavezető:
Dr. Tanács Attila
egyetemi adjunktus

Szeged
2023

Feladatkiírás

A feladat egy olyan, modern webes technológiai környezetben készített webalkalmazás készítése, amely egy kiválasztott kistérség turisztikai látnivalóiról ad 3D modellezés segítségével interaktív tájékoztatást. Jelenjen meg a térség vaktérképe egy virtuális terepasztalon, amelyen kattintható objektumok jelképezzék a látnivalókat és az utazási lehetőségeket. Az egyes elemekre kattintással bővebb szöveges és képi leírást kapjunk.

A feladat célja a technológiai környezet kialakítása a 3D modellezéssel, információs panelek megvalósításával, de nem elvárás, hogy teljes körű turisztikai információval fel legyen töltve. Elegendő a teszteléshez szükséges mennyiségű információ elérhetővé tétele. A 3D modellek származhatnak saját forrásból, vagy jogszerűen felhasználható meglévő forrásokból.

Tartalmi összefoglaló

- ***A téma megnevezése:***

Térbeli grafikával támogatott turisztikai információs webalkalmazás Angular környezetben

- ***A megadott feladat megfogalmazása:***

A feladatom egy olyan webalkalmazás létrehozása volt, ami bemutatja a Lillafüredi kisvasútat. Két elkülöníthető rész létrehozása volt a feladat, egy a vonatoknak, egy pedig a nevezetességeknek és a térképnek, ahol információt lehet kapni ezekről.

- ***A megoldási mód:***

A webalkalmazás érdekesebbé tételéhez Blender vonat modelleket készítettem és ezeket helyeztem el a webalkalmazás egyik oldalán a vonatokról szóló információkkal. Valamint a másik oldalra Blender-ben elkészített terepasztal tettem, Blender-ben elkészített megállókkal. A megállókra való kattintáskor megjelenik az információ a környékbeli nevezetességekről, jegyárakról, árakról.

- ***Alkalmazott eszközök, módszerek:***

Angular keretrendszerrel készítettem el a webalkalmazást, valamint a Three.js segítségével importáltam be a Blenderben elkészített modelleket. A kódot a Visual Studio Code-ban készítettem el.

- ***Elért eredmények:***

Elkészült a webalkalmazás a Lillafüredi kisvasútról, amin négy oldal található. Az első a kezdőoldal, ami a navigációt segíti elő, a második egy útmutató oldal, ami elmagyarázza a webalkalmazás másik két oldalán található funkciókat. A harmadik oldalon található a terepasztal és a megállók modellje. A megállókra kattintva megjelennek az információk az adott megállóról. A negyedik oldalon a vonatokról találunk információt, valamint ezen az oldalon megtekinthetőek a vonatok modelljei, amit a felhasználó elmozgathat.

- ***Kulcsszavak:***

Angular, Blender, Three.js, 3D modellezés, webalkalmazás

Tartalomjegyzék

Feladatkiírás **3**

Tartalmi összefoglaló **4**

Tartalomjegyzék **5**

BEVEZETÉS **7**

1. RENDSZERTERV **9**

1.1. Követelmény specifikáció **9**

1.2. Rendszerterv **9**

1.3. Blender **12**

1.4. Angular **12**

1.5. Three.js **13**

2. BLENDER HASZNÁLATA **14**

2.1. Elkészült Blender modellek **14**

2.2. Konkrét modellek elkészítése **14**

2.3. Modellek exportálása **19**

3. ALKALMAZÁS FELÉPÍTÉSE ANGULARRAL **22**

3.1. App.module **22**

3.2. Nav modul **23**

3.3. Info modul **24**

3.4. Map modul **24**

3.5. Not-Found modul **26**

3.6. Splash modul **26**

3.7. Trains modul **27**

4. THREE.JS **30**

4.1. Blender modellek betöltése **30**

4.2. Raycast **31**

5. FELHASZNÁLÓI ÚTMUTATÓ 33

5.1. Kezdőoldal és Útmutató 33

5.2. Térkép 34

5.3. Vonatok 36

Irodalomjegyzék 38

Nyilatkozat 39

Köszönetnyilvánítás 40

Mellékletek 41

BEVEZETÉS

A motivációm a weboldal elkészítése mögött az volt, hogy szerettem volna egy olyan grafikus weboldalt készíteni, ami jobban leköti a felhasználók figyelmét. Úgy gondolom, ha egy oldalon több a 3D-s modell, akkor sokkal figyelemfelkeltőbb. Manapság egyre több weboldal készül el, amiben megtalálhatóak 3D-s elemek.

Az utóbbi időben sokat túráztunk a barátokkal és többször néztünk meg kisvasutakat is. Szerettünk volna előzetesen informálódni a menetrendekről, megállókról és a környéken található nevezetességekről, esetleg az ott használható vonatokról. Viszont ezeken a weboldalakon nem volt túl sok információ, nem voltak olyan látványosak, és több oldalról kellett összekeresgélni a környéken számításba eső nevezetességeket. Ezeken az oldalakon voltak ugyan képek a vonatokról, viszont ezek csak 2D-ben ábrázolták őket. Illetve találtam egy-két weboldalt, ahol elkészítették több vonat modelljét, de ezek már régebben készültek, ami miatt nem voltak olyan jó minőségűek, és nem volt az összes vonat bemutatva. Ami a kisvasutak térképét illeti, csak fényképet találtam. A nevezetességek pedig nem voltak összegyűjtve megállók szerint. Tehát úgy gondoltam, ezeket a funkciókat egy oldalon kellene megjeleníteni.

Az implementáláshoz a Three.js[1] könyvtárat használtam, mivel tanultam már a Számítógépes grafika órán korábban és tudtam, hogy ezzel a Blender modelleket könnyedén be lehet importálni a weboldalba [1,2]. A másik ok amiért a Three.js-t használtam, hogy fel tudjam használni a `raycast`-ot, ami hasznosnak bizonyult a dolgozatom során.

Az implementáláshoz az Angular keretrendszert[2] használtam [3]. Azért ebben a keretrendszerben írtam a kódot, mert tudtam, hogy nagyobb kódot szét tud bontani modulokra, amik kezelhetőbbé teszik a weboldalt. Ennek köszönhetően öt fő modulra fel tudtam bontani az alkalmazásomat. Az Angular másik előnye az Angular Material komponens, amit a kód írása közben is sokszor használtam. Ennek a segítségével nem kell sokat foglalkozni a formázással, mivel van több alapértelmezett design eleme.

Ugyan a Three.js-ben is lehet modellezni, én a modellek elkészítéséhez a Blender 3D modellező programot[3] használtam [4, 5]. Az egyetemen a Számítógépes grafika tárgyunk egy része is erről a programról szólt, de korábbról is ismertem már. Úgy gondoltam ez megfelelő

[1] Hivatalos honlap: https://threejs.org/
[2] Hivatalos honlap: https://angular.io/
[3] Hivatalos honlap: https://www.blender.org/

lesz, mivel nincsen hasonló modellező program, ami ingyenes lenne, és amivel 3D modelleket lehetne ilyen jó minőségben elkészíteni. Az előbb említett pozitív tulajdonságok miatt nagyon népszerű ez a program és emiatt sok információt meg lehet találni a Blender használatáról. A modellek textúrázását is a Blender-ben valósítottam meg, innen csak ki kellett exportálnom a modelleket, amit a későbbiekben ki is fejtek.

A weboldalon a kezdőoldal, útmutató oldal és két fő oldal található. A kezdőoldal funkciója, hogy elnavigáljon a másik oldalakra, az útmutató oldalon pedig információt lehet szerezni a weboldal használatáról. Az egyik főoldalon meg lehet tekinteni a Lillafüredi kisvasút térképét, illetve a megállókat, ahol információt kaphatunk, hogy melyik megállóhoz milyen nevezetességek és természeti látványosságok vannak a legközelebb. A másik főoldal a Lillafüreden valaha közlekedő vonatokat mutatja be. Információt lehet kapni minden egyes vonatról. Ezen az oldalon megtekinthető minden vonat 3D-s modellje, amit a felhasználó minden irányba elmozgathat.

1. RENDSZERTERV

Ebben a fejezetben arról lesz szó hogyan fogalmaztam meg a feladatot, mire volt szükség az elkészítéséhez, és hogyan terveztem meg a weboldalt.

1.1. Követelmény specifikáció

- Turisztikai weboldal a Lillafüredi kisvasútról
- Könnyen lehet információt találni a:
    - Menetrendekről
    - Jegyárakról
    - Térképről
    - Nevezetességekről
    - Vonatokról
    - Vonatok kinézetéről
- Vizuális weboldal
- Legyen bevonva a felhasználó
- Legyenek elkülönítve a részek
- Első oldal:
    - Megjelenjen a térkép és a nevezetességek
    - Térkép valósághűen ábrázolja a ma is használt sínhálózatot
    - Nevezetességek legyenek csoportosítva
    - Eredeti weboldalak legyenek elérhetőek az oldalról
- Második oldal:
    - Összes itt használt vonatról legyen információ
    - Vonatok modelljei megtekinthetőek legyenek
    - Valósághűek legyenek a modellek

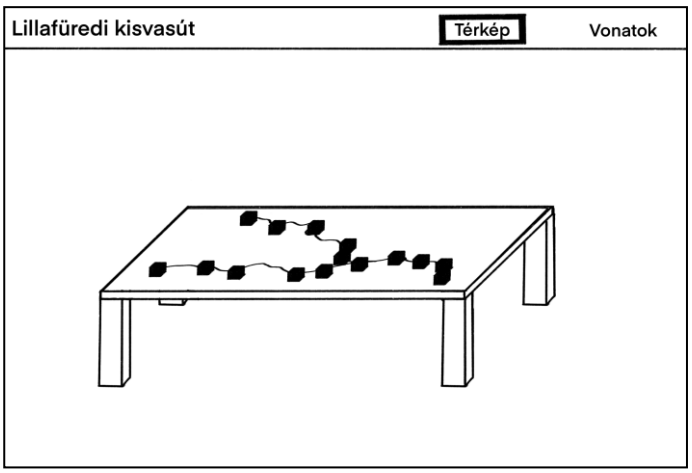
1.2. Rendszerterv

A tervezéskor elsőként a weboldal kinézetére koncentráltam. Előbb megterveztem, mik kerüljenek a képernyőtervekre. Mivel a két résznek jól elkülöníthetőnek kellett lennie, ezért

két oldalt létrehozására volt szükség, egy a térképnek és a nevezetességnek, egy pedig a vonatoknak.

Mivel több oldal készítésére volt szükségem, így ezekhez érdemesnek találtam egy navigációs oldalt is létrehozni, ami a kezdőképernyő lesz. Valamint szükség volt, hogy minden oldal tetején legyen egy egységes navigációs sáv is létrehozva, azért, hogy ne kelljen mindig visszamenni a kezdőoldalra mikor oldalt akar a felhasználó váltani.

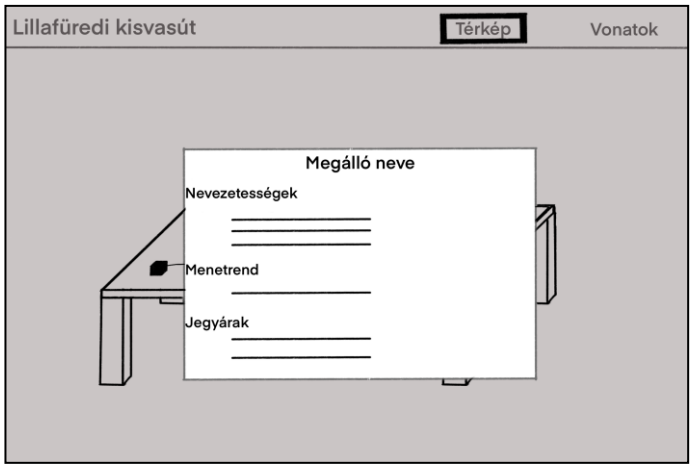
Az első oldalnak a térképes oldalt választottam (1.1. ábra), hogy előbb megismerje a felhasználó a kisvasutat és a nevezetességeket, illetve, hogy egyből azt lássa, hogy merre tud utazni.



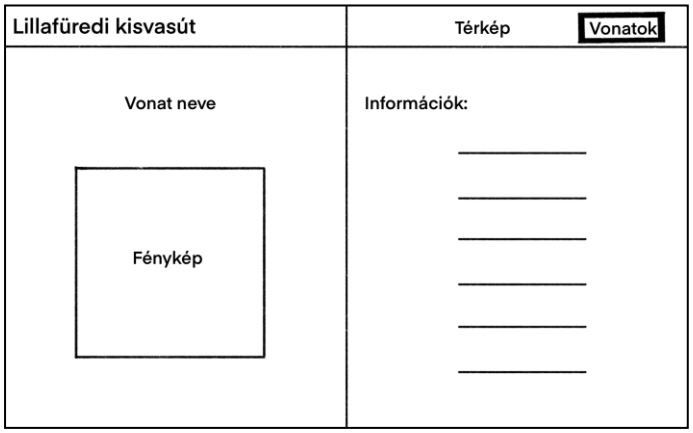
1.1 ábra: Térkép oldal képernyőterve

Ezen az oldalon a térképet 3D-ben valósítottam meg, a végső terv egy Blender-ben elkészített terepasztal lett. A megállókat is 3D elemként terveztem meg, hogy könnyen észrevehető legyen melyik hol helyezkedik el. A térképet is interaktívvá alakítottam később, úgy hogyha a felhasználó kattint valamelyik megállóra, annak a megállónak az információi jelenjenek meg amire kattintott. A nevezetességeket pedig elkülönítve kellett megjeleníteni, így a dialógusablak bizonyult a legjobb döntésnek (1.2. ábra). Minden megállóhoz a környéken található nevezetességeket ki kellett írni, ezeknek az eredeti oldalát be kellett linkelni, valamint a menetrendet és az árakat minden megállónál meg kellett jeleníteni.

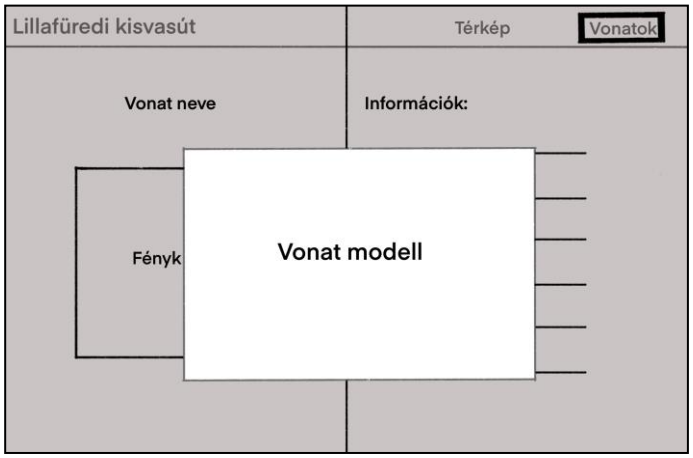
A második oldal a vonatokról szól. Kettéosztottam az oldalt, így egyből látja a felhasználó az információk mellett, hogy néz ki a vonat (1.3. ábra). A vonatokat is egyesével terveztem megjeleníteni azért, hogy jobban átlátható legyen az oldal. Elkülönítettem a modelleket, hogy jobban fókuszban legyenek, erre is a dialógusablak bizonyult a legjobbnak (1.4. ábra). A modellt mindenképpen interaktívvá akartam tenni. A felhasználó is körbe tudja mozgatni a vonatot úgy ahogyan ő szeretné. Illetve a modell magától is mozogni fog.



1.2 ábra: Dialógusablak képernyőterve a Térkép oldalon



1.3 ábra: Vonatok oldal képernyőterve



1.4 ábra: Vonat modell megjelenítése dialógusablakon

Majd arra koncentráltam, hogy milyen technológiákkal tudom ezeket a terveket a legjobban megvalósítani. Mivel korábban ismertem már a Blender-t és tudtam, hogy jó minőségben, gyorsan lehet dolgozni benne, a modellek elkészítéséhez ezt a programot választottam.

Az egyetemen korábban megismerkedtem a Three.js-szel. Innen tudtam, hogy ezt a könyvtárat könnyedén lehet használni a Blender-rel együtt. A modellező programban elkészített modelleket egyszerűen be lehet importálni a Three.js-be.

Végül a weboldal implementálásához kerestem egy programozási nyelvet. Tudtam, hogy meg lehet valósítani HTML-lel, Javascript-tel, de olyat kerestem, aminek vannak bővítményei, amik a segítségemre lesznek. Tehát keretrendszert kellett keresnem és a legjobbnak az Angular tűnt. Az egyik pozitívuma az, hogy külön modulokra lehet felosztani az alkalmazást. Ez nagy segítségemre volt, mivel több fájlból terveztem felépíteni az alkalmazást. A másik indok amiért az Angular hasznos volt, az az Angular Material. Ennek segítségével nem kell sok elem formázásával foglalkozni, mivel az Angular Material pont erre jó, hogy vannak alapértelmezett formázásai beépítve.

1.3. Blender

A Blender egy grafikai modellező program. Nagyon elterjedt manapság, mivel felsőkategóriás modelleket, animációkat, valamint játékokat lehet vele elkészíteni, de a többi hasonló kategóriás szoftverrel ellentétben ez ingyenesen hozzáférhető bárki számára. Mivel önkéntesek fejlesztik, nekik köszönhetően gyorsan tud fejlődni a Blender, a felhasználók igényeihez mérten. Mostmár több vállalat is támogatja így nagyobb ütemben tud fejlődni.

A Blender tökéletes kezdőknek, mivel nem kell megvásárolni a szoftvert, valamint egy kis gyakorlással nagyon könnyen bele lehet tanulni. Mivel nagyon elterjedt, ezért rengeteg információ, segítség és oktató videó található az. Valamint sok területet lefed, például modellezés, játékkészítés, animációk készítése, ezért sokféleképpen lehet használni.

Sokszor felhozzák negatívumnak, hogy sok gyorsbillentyű kód van a Blender-ben, ami nehézséget okoz. Viszont ezek segítségével gyorsabban lehet dolgozni, ha valaki ismeri már ezeket a gyorsbillentyűzeteket.

1.4. Angular

Az Angular egy HTML és Typescript alapú keretrendszer. Az Angular Typescript nyelven íródott, aminek az alapja a JavaScript. Egy Angular alkalmazás fő elemei a komponensek, modulok, valamint a service-ek. Az Angular komponensek, `NgModulok`-ba szerveződnek. Az `NgModules` funkcionális halmazba gyűjti a kapcsolódó kódot.

Egy Angular komponens négy fájlból épül fel. Az első egy HTML fájl, ami a frontend kódot tartalmazza. A következő fájl egy CSS fájl, ahol a formázás található. A következő két fájl Typescript fájl, az egyik egy „.spec” résszel egészül ki, ezt használják tesztelésre, a másik

pedig a HTML kód mögötti logikát tartalmazza. Valamint található egy .module fájl. Minden alkalmazásban van legalább egy gyökérmodul, de általában több modul is található egy Angular alkalmazásban. Ezeknek a segítségével könnyedén tudjuk az alap keretrendszer eszköztárát bővíteni. Vannak olyan komponensek, ahol található még egy `routing` fájl, ez segíti az alkalmazáson belüli navigációt.

Valamint az Angular-ban található az Angular Material, ami a felhasználói felület komponenskönyvtára, amely segítségével könnyebben lehet az alkalmazást formázni. Sok beépített eleme van, például a naptár, a kártyák, gombok. Ezeket fel lehet használni eredeti formájukban, de könnyen személyre is lehet szabni őket.

1.5. Three.js

A Three.js egy JavaScript alapú könyvtár, amely képes `GPU` alapú játékok és más grafikus alkalmazások futtatására közvetlenül a böngészőből. A Three.js könyvtár számos funkciót és `API`-t biztosít 3D-s jelenetek böngészőben történő rajzolásához. `WebGL` segítségével jeleníti meg a böngészőben a számítógépes grafikákat. Azért jó ez a könyvtár mert úgy lehet megjeleníteni összetettebb animációkat, hogy nincs szükség önálló alkalmazásra vagy bővítményigényre.

Sok lehetőség rejlik benne, mivel sok tulajdonsága van, például különböző kamerák, ortografikus vagy perspektivikus. Fényekből is sok fajtát lehet használni, reflektor-, vagy pontfényt. Különféle anyagok is rendelkezésre állnak, mint például a `Phong` vagy a `Lambert`. A jelenetekhez futásidőben tudunk objektumokat hozzáadni és kitörölni. Segédprogramokat tudunk használni, ilyenek a 3D matematikai függvények és mátrixok. Illetve segédprogramok segítségével Three.js-kompatibilis `JSON`-fájlokat lehet létrehozni: Blender, openCTM, FBX, MAX, OBJ, és ezek segítségével könnyen tudunk exportálni, importálni.

2. Blender használata

Ebben a fejezetben arról fogok írni, hogy mit csináltam a Blender grafikai programban. Az alfejezetek között lesz egy olyan, amiben bemutatom az összes modellt, amit a webalkalmazáshoz készítettem, a következő alfejezetben két modell részletes elkészítését leírom, valamint arról lesz még szó hogyan kell a modelleket kiexportálni, hogy jól jelenjenek meg a modellek egy Three.js jelenetben.

2.1. Elkészült Blender modellek

A webalkalmazásomhoz a Blender-ben 12 db különböző vonatot, 1 db megállót, 1 db terepasztalt készítettem el, amin található egy sínhálózat, valamint egy hátteret modelleztem a térkép mögé (2.1. ábra). A modellek nagy változatát a mellékletben helyeztem el kép formátumban.

Mindegyik modell textúrázását a Blender-ben készítettem el az `UV mapping`-gel. A vonatoknál olyan képeket kerestem a textúrázáshoz, ami az eredeti vonatról készült. Viszont a megállókhoz egy tetszőleges házról készült képet választottam, a hátteret is hasonlóan készítettem el. A terepasztalhoz keresnem kellett egy terepasztal fű textúrát, valamint egy képet fa mintáról. A síneket pedig fém, illetve fa textúrákkal készítettem el.

A vonatokat képek alapján modelleztem le, mivel nem álltak rendelkezésemre tervrajzok, tehát nem méretarányosak, de amennyire lehetett a valóságot tükrözik.

Az eredeti megállók elég különbözően néztek ki, tehát van amelyiknél egy épület a megálló, de van, ahol csak egy tábla jelzi a megállót. Emiatt, hogy sok féle megálló jelzés van a valóságban, én egy egységes házat készítettem el megállónak. Ezeket a házakat a terepasztalra helyeztem később.

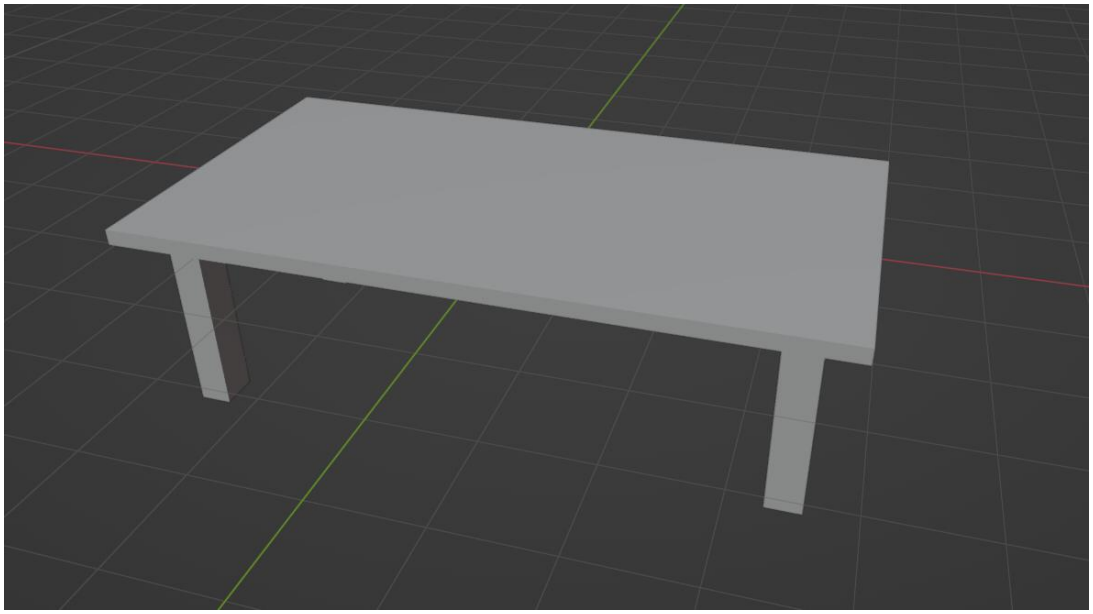
A térkép végül egy terepasztalként készült el, amin a ma is használt síhálózat van lemodellezve.

2.2. Konkrét modellek elkészítése

Két modell elkészítését fogom bemutatni. Az első modell, amit részletesen bemutatok az a terepasztal lesz. A terepasztal két részből épül fel. Az egyik az asztal, a másik pedig a sínhálózat.

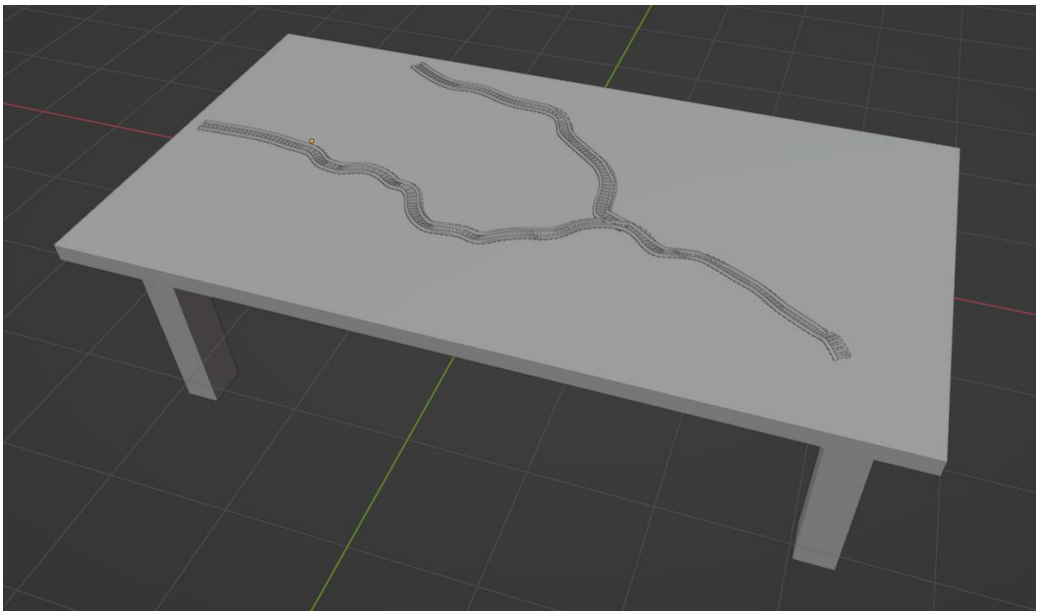
2.1. ábra: Blenderben elkészült összes modell

Az asztalnál egy kockából indultam ki, amit a Shift+A gyorsbillentyűzettel adtam a jelenethez. Majd megnyomtam a Numpad-en az 1-es gombot, és így 2D-ben láttam a modellt, így, amikor formáztam nem deformálódott el a modell. Objektum módban az S billentyű lenyomásával skáláztam a kockát. Ha csak az egyik irányba szerettem volna, akkor az S lenyomása után annak a síknak a betűjét nyomtam le, amelyiket szerettem volna módosítani. Ezután megnyomtam a 7-es billentyűzetet a Numpad-en, ekkor felülről láttam az objektumot, most innen nézve skáláztam a modellt. A lábait is kockából hoztam létre, aminek a méreteit skálázással készítettem el (2.2. ábra).



2.2. ábra: A terepasztalból az asztal modellje

Ahhoz, hogy jó legyen a sínhálózat formája, beimportáltam egy képet a Blender-be, Shift+A, `Images`-zel. A sínhez létrehoztam egy vonalat, Shift+A, `Curve`, `Path` kiválasztásával. Megformáltam Edit módban, hogy milyen alakú sínt szeretnék a korábban beimportált kép alapján. Ezután elkészítettem a formáját, ehhez létrehoztam egy kockát, 1-gyes lenyomásával oldalról láttam, ahol Szerkesztő módban Ctrl+R segítségével `Loop Cut`-okat hoztam létre. A két középső területet S-sel kisebbre skáláztam, majd, beletettem még több `Loop Cut`-ot, hogy szebb legyen a formája. Majd jobb gomb `Convert to Curve`. Kijelöltem a vonalat és jobb oldalt az `Object`-nél kiválasztottam az előbb elkészített sín formáját és így a sín azon a vonalon ment ahogy az előbb a vonalat beállítottam. Ezután egy kockából elkészítettem a talpfát. Skáláztam, hogy arányos legyen a sínhez, és beállítottam a `Modifier`-eknél az `Array`-t, így a talpfa ismétlődött egymás után. Az `Array`-nél pedig átállítottam a `Constant Offset`-re. Új `Modifier`-t adtam hozzá, a `Curve`-t, majd kiválasztottam a `Curve Object`-nél a sínt. Ezután az `Array Modifier`-ben a `Fit Type`-ot átállítottam `Fit Curve`-re, és kiválasztottam a sínt. Erre azért volt szükség, hogy ne egyenes vonalban ismétlődjenek a talpfák, hanem a sín mentén. Szükségem volt a sínpár másik sínjére is, tehát létrehoztam egy `Path`-t, majd a `Modifier`-eknél hozzáadtam a `Curve`-t és a már kész sínt választottam ki alapnak, így a már kész sín vonalát vettem fel ez a sín is. Majd elmozgattam a G+Y gyorsbillentyűzettel az Y tengely mentén. A sín vastagságát ugyanúgy csináltam meg, mint az elsőét. És végül a talpfákat elmozgattam, hogy mindkét sínpárhoz hozzáérjen (2.3. ábra).



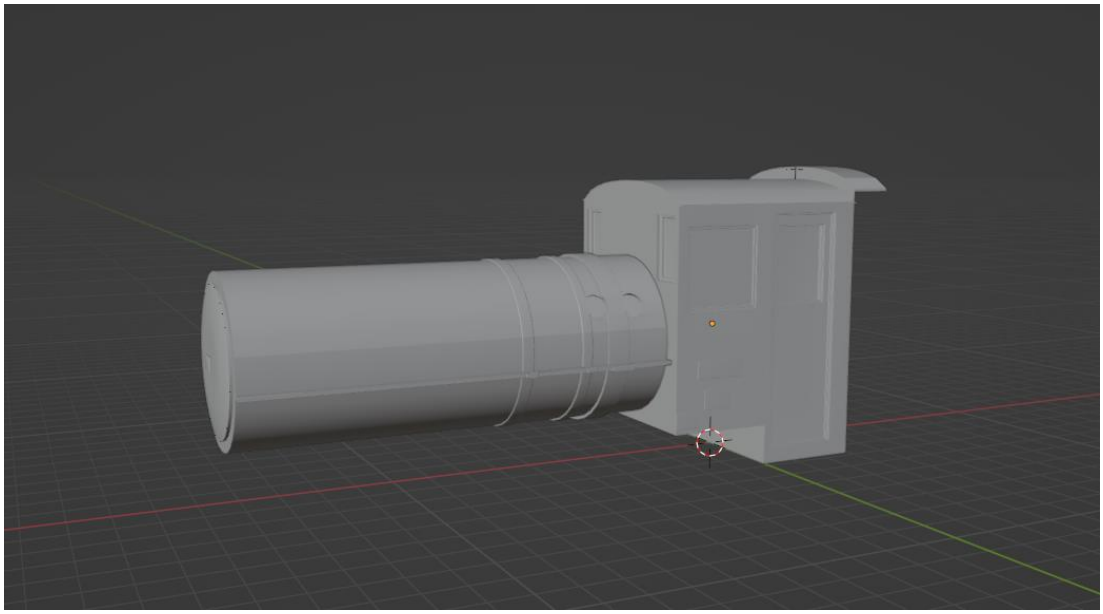
2.3. ábra: A kész terepasztal modellje

Ezután következett a textúrázás, amit UV mapping-gel csináltam. Kerestem három képet, amivel tudtam dolgozni, az egyik egy terepasztal füvéről készült kép, a másik egy faanyag, a harmadik pedig egy metálos anyagról egy kép. Blender-ben a Shading fülön, minden elemhez hozzáadtam új Material-t, majd egy Image Texture-t, ahol kiválasztottam az előbb említett képeket. Az asztalhoz és a talpfákhoz a faanyagos képet, a sínpárhoz egy metalos képet, a terepasztal tetejéhez pedig a fűről készült képet. Ezután átmentem Blender-ben az UV-Editing fülre, és itt kiválasztottam az adott elemet, amit szerkeszteni akartam és ha átváltottam Szerkesztő módra, megjelent bal oldalon a kiválasztott elem dróthálója. Majd bal oldalon, ahol a dróthálója megjelent az objektumnak, kiválasztottam a megfelelő képet. Ha mozgattam a dróthálót a képen, jobb oldalon úgy változott az objektum színe. Tehát oda mozgattam el az objektumot amilyen színt szerettem volna adni neki. Majd minden objektummal megcsináltam ezt a műveletsort (2.4. ábra).

A második modell, aminek az elkészítését bemutatom az a KV-4-es vonat. A vonat alapja egy kocka és egy henger volt. Mindkettő elemet skáláztam méretarányosan és elhelyeztem egymás mellett őket. A kockát először Szerkesztő módban Edge Loop-okra felbontottam, Ctrl+R-rel. Kiválasztottam a középső Loop-ot, majd megnyomtam az O-t, aztán a G+Z-t, ennek hatására, ha mozgattam felfelé az egeret, nem csak a középső Loop mozdult el, hanem követték a mellette levők is. Így készült el a kocsi vezető kabinjának a boltíve. A kockából Extrude és Invert segítségével létrehoztam az ablakokat, valamint a hátsó ajtót. A hengeren is végrehajtottam a Loop Cut-ot és az Extrude-ot, és ennek hatására létrehoztam a hosszanti kiemelkedéseket a vonat oldalán. A henger oldalán vannak bemélyedések azt pedig a Bool-

Tool kiegészítővel csináltam meg. Hozzáadni a kiegészítőt az Edit Preferences Add-ons Bool-Tool-nál lehet. Ezután pedig a hengerhez illesztettem egy másik kisebb hengert, majd kijelöltem a kis hengert, aztán a nagy hengert Ctrl és mínusz gomb. Ekkor már csak ki kellett törölni a kis henger hálóját és a hengerben maradtak a bemélyedések (2.5. ábra).

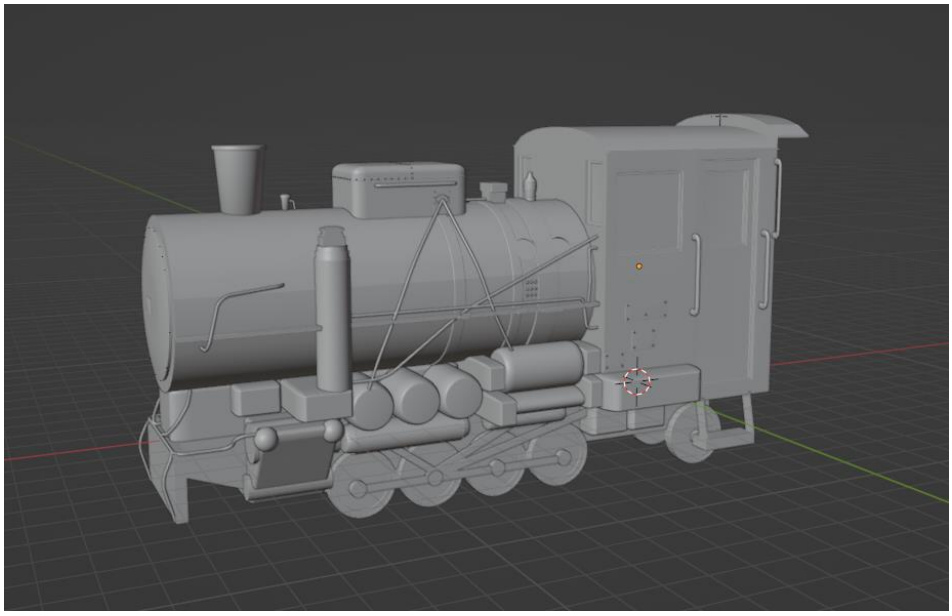
2.4. ábra: Terepasztal textúrázása



2.5. ábra: Vonat modell fő elemei

Ezután elkezdtem a kisebb részleteket elkészíteni. A kerekeket hengerekből hoztam létre, skáláztam őket az S lenyomásával. Majd Invert-tel és Extrude-dal kialakítottam a lépcsőzetes formáját a kerekeknek. A Vonat fogókáit is hengerekből alakítottam ki, először

leskáláztam kisebbre, majd Edit módban Ctrl+R-rel a közepére egy Loop Cut-ot hoztam létre. Az egyik felét kitöröltem, és ezután hozzáadtam a Mirror Modifier-t Z tengelyre. Ezután pedig elkezdtem Extrude-dal bővíteni felfelé a hengert és elkezdtem minden bővítésnél elforgatni a Rotate-tel. A kábeleket úgy készítettem el, hogy létrehoztam egy kockát, ezt Szerkesztő módban M+Center-rel egy ponttá kicsinyítettem, majd ezt a pontot elkezdtem Extrude-dal bővíteni abba az irányba, ahova szerettem volna a kábelt tenni. Ha szerettem volna a kábelt kerekebbé tenni Ctrl+B+V-vel csináltam meg. Átkonvertáltam görbévé, majd jobb oldalon a Curve fülön a Depth-et elállítva adtam neki vastagságot. A többi részletet hengerek, kockák és gömbök skálázásával, elforgatásával készítettem el a fénykép alapján. Ha volt egy objektum, ami nem volt elég sima, akkor azt általában a Ctrl+A Shade Smooth-tal oldottam meg, de ezután a jobb fülön be kellett állítani a Normals-nál a 30 fokot és így nem veszett el teljesen az objektum formája (2.6. ábra).

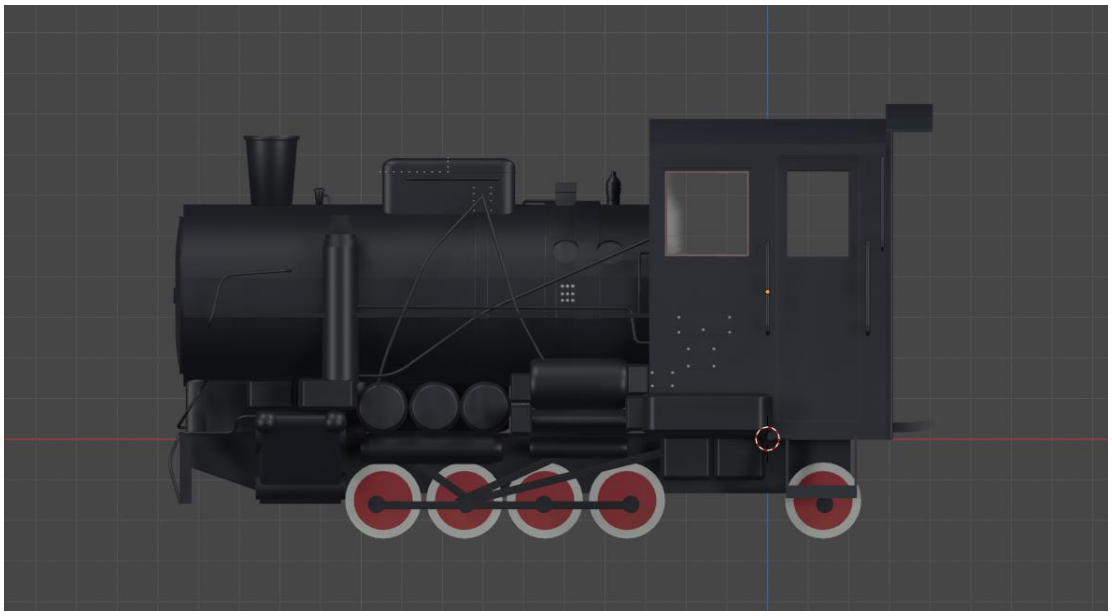


2.6. ábra: Vonat modell a részletekkel együtt

A vonat textúrázása ugyanúgy készült el ahogy a terepasztal, de itt minden objektumhoz az eredeti vonatról készült képet választottam ki az új Material-nál. (2.7. ábra).

2.3. Modellek exportálása

A Blender a modelleket a saját bináris .blend formátumában menti le. Viszont nekem a modelleket ki kellett exportálnom, hogy fel tudjam őket használni a Three.js-ben. A modelleket többféleképpen lehet exportálni, a két legnépszerűbb formátum a Glt és az Obj. Én az Obj-t választottam.



2.7. ábra: A kész vonat modell textúrázva

A Blender-ben az alapértelmezett exportálások közül kiválasztom a `Wavefront (.obj)` opciót. Ezekután a dialógusablak jobb oldalára kellett figyelni, mivel be kellett állítanom, hogy mentse le a program a modellemet. Az `Include` fülön az `Object as OBJ Objets`-nek kellett bepipálba lennie. A `Transform` fülnél két dolgot is meg kellett nézni, hogy be van-e jól állítva. Az egyik a `Forward`-nak `-Z Forward`-on kellett állnia, az `Up`-nál pedig `Y Up` kellett, hogy be legyen állítva. A `Geometry` fülön kellett figyelni a legtöbb beállításra. Be kellett pipálva lennie az `Apply Modifiers`-nek, a `Write Normals`-nek, az `Include UVs`-nak, a `Write Materials`-nak, valamint a `Triangulate Faces`-nek. Egy kivételével mindegyik alapbeállítás a programban, tehát ha nem volt korábban el állítva, akkor mindegyik úgy lesz kiválasztva ahogy kell, kivéve a `Triangulate Faces`. Ennél figyelni kell, hogy ne felejtsük el bepipálni.

Ekkor két fájl jön létre, az egyik egy .obj fájl a másik pedig egy .mtl fájl. Mindkettő szöveges fájl. A .obj-ben találhatóak a geometriák és a görbék, tehát itt tárolja el a Blender, hogy milyen alakzatokat használtam, milyen formázásokat hajtottam végre rajtuk. Az .mtl-ben pedig a modell anyagát tárolja el. Például, hogy milyen textúrákat használtam fel és honnan származnak ezek.

Mivel én a Blender-ben készítettem el a modellek textúrázását `UV Mapping`-gel, figyelnem kellett arra, hogy a textúrázáshoz használt képeknek az elérése jó legyen. Tehát relatív elérésre kellett állítani a képek útvonalát az .mtl fájlban.

Eleinte úgy volt, hogy elég ennyi dologra figyelni az exportálásnál, de problémák adódtak a Three.js-be való beimportáláskor, ezért még két beállításra is figyelni kellett a későbbiekben. Az első dolog amire figyelni kellett még, az az volt, hogy a modellekbe több vonal geometria került bele véletlenül és emiatt a modellek drótvázasak lettek a Three.js-ben. Ezt kétféleképpen lehet megoldani, az egyik, hogy a Blender-ben megkeresem ezeket a geometriákat és kitörlöm. Mivel nagy és bonyolult modellekről volt szó, nem találtam meg mindet, így, ha ilyen probléma van akkor az .obj fájlban kell megkeresni az l-lel kezdődő sorokat és ezeket ki kell törölni.

A másik dolog amire figyelni kellett exportáláskor az az átlátszóság. Ez a vonatok ablakainál jelentkezett. Amikor a Blender-ben elkészült az ablak átlátszóra a `Shadow Mode Opaque`-val, nem lehetett beimportálni a Three.js-be, tehát máshogy kellett megoldani az átlátszóságot. A Blender-ben az ablakoknak más `Material`-t adtam, ami az .mtl fájlban megjelenik. Ekkor ennek a `Material`-nak a d-vel kezdődő sorát kell elállítani 1-ről, egy kisebb értékre.

3. Alkalmazás felépítése Angularral

A webalkalmazásomat több modulra osztottam fel, hogy könnyebben el legyenek különítve a részek. Az alkalmazás gyökérmodulja az `App.module`, itt indul el az egész alkalmazás. A moduljaim: `Info, Map, Nav, Not-found, Splash` és `Trains`. A modulok szerint mutatom be az alkalmazásomat, mivel minden modulban egy oldalt készítettem el. Kivéve a `Nav` modul, ami a navigációs sávot tartalmazza. Mivel az minden oldal tetején megtalálható, célszerűbbnek láttam kivenni egy külön modulba. Ezen kívül található az alkalmazásban egy `constants.ts` nevű fájl, ahonnan később adatokat olvasok be.

Általánosságban egy modulban öt darab fájl található, illetve gyakran használtam még egy hatodik fájlt kiegészítésnek. Az öt fájl között van egy darab HTML fájl, egy CSS és kettő Typescript fájl. A hatodik fájl, amit használtam, az egy `routing` nevű typescript fájl volt, ami a navigációt segítette elő.

Az alkalmazáshoz szükség volt a legújabb node.js-hez, valamint az Angular CLI-re, amiket letöltöttem. Létrehoztam egy projektet a Visual Studio Code-ban, majd ezután a terminálban az `npm install`-lal telepítettem az alkalmazáshoz szükséges függőségeket, és végül az `ng serve`-vel elindítottam. Az utóbb említett parancsot kellett később használni, ami után a webalkalmazás megtekinthetővé vált a `localhost`-on.

3.1. App.module

Az App modul ahogy fentebb is említettem a webalkalmazás gyökérmodulja, ami a webalkalmazás futását biztosítja.

Ennek a modulnak a HTML fájljában a webalkalmazás általános oldal kinézetét készítettem el. Tehát a HTML fájlba a navigációs rész szelektorja szerepelt először, ezt úgy tudtam beolvasni hogy az app.module.ts-hez hozzáadtam a Navigációs sáv modulját, majd a HTML-ben már hozzáférhettem a modul szelektorjához `<app-nav></app-nav>` formában. Így tudtam meghívni, hogy mindegyik oldalon a Navigáció jelenjen meg az oldal tetején.

A következő sor a HTML fájlban, a `router-outlet` nevű szelektor, ami az app-routing.module.ts fájlt hívja meg. A `Router-Outlet` egy Angular direktíva a router könyvtárból, amely az útvonalak által összeillesztett komponensek megjelenítését segíti a webalkalmazásban és ez támogatja a `lazy-loading`-ot.

Én is ezt használtam az app-routing.modul fájlban, mert ennek segítségével könnyebb a navigáció kezelése a webalkalmazásban. Az app-routing.module.ts fájlba beexportáltam az NgModule-t, a Routes-t és a RouterModule-t az @angular/router csomagból. Ezután elkezdtem a Routes mappába felírni az összes lehetséges útvonalat, az alábbi módon:

```
const routes: Routes = [
   {
      path: 'splash',
      loadChildren:()=>import('./pages/splash/splash.module').then(m=>m.SplashModu
      le),
   }
```

A path mögé azt a nevet írtam, ahogy szeretnék erre az útvonalra hivatkozni, tehát itt a splash. A következő sorban az importba beírtam a kiválasztott modul elérési útvonalát relatívan, és a then után pedig a modul nevét írtam be. Ezt megcsináltam az összes oldallal, majd felvettem még két másik elérési útvonalat. A következő útvonalnál azt állítottam be, amikor a path üres. Ez azt jelenti, hogy mi lesz a kezdőoldal, ennek a Splash nevű modult választottam és az egyezést full-ra állítottam be. Ez azt jelenti, hogy az útvonalnak mindig pontosan egyeznie kell. Az utolsó lehetőségnek pedig a path mögé ** került, ami azt jelenti hogyha az egyik eddigi eshetőség sem következett be akkor ezt az oldalt nyissa meg, ezt pedig a redirectTo után kellett írni a nevet rövidítve, én a not-found oldalt választottam. Figyelni kellett a sorrendre, nehogy a **-os útvonal hamarabb legyen, mint a többi, mert akkor a később található útvonalak sosem töltenek be.

Ahhoz hogy működjenek az útvonalak, az app.component.ts fájl konstruktorához hozzá kellett adni a private routes:Router sort. Az app.module.ts fájlban a routing miatt nem kellett az összes modult beimportálni, hanem csak a RoutingModule kellett, mert oda már be vannak a modulok töltve.

3.2. Nav modul

A Nav modulban a navigációs fejlécet készítettem el. Ahhoz, hogy minden jól működjön, a nav.module.ts-be be kellett importálni az Angular Material könyvtárból a MatMenuModule-t, a MatButtonModule-t, valamint a MatToolbalModule-t, illetve az AppRoutingModule-t. A Navivációs modulhoz nincsen szükség külön routing fájlra, mivel ez nem jelenik meg külön oldalon.

A HTML fájlban az `Angular Material Toolbar` szelektorját használtam fel, ami segítségével a navigációs sávot nem kellett formázni, mivel az Angular Material adott hozzá egy általános formázást, csak a CSS fájlban változtattam meg a színét. Ezután hozzáadtam egy gombot, amit elneveztem Lillafüredi kisvasútnak. Ehhez hozzáadtam a `mat-button`-t és így nem is kellett megszerkesztenem a gombot. Ezután, hogy működjön a navigáció hozzáadtam a gomb `routerLink`-jéhez azt a rövid elérési útvonalat, amit a gyökérmodul alfejezetben létrehoztam. Tehát ha a `SplashModule`-t szerettem volna meghívni akkor csak azt kellett a `routerLink`-hez írnom, hogy /`splash`.

```
<button mat-button routerLink="/splash" >Lillafüredi kisvasút</button>
```

Ezután pedig ugyanígy létrehoztam a többit gombot is, a Kezdőoldalt, Útmutatót, Térképet és a Vonatokat.

Azért, hogy a Lillafüredi kisvonat neve elkülönüljön a navigációs sáv baloldalán CSS-ben `flex: 1 1`; segítségével oldottam meg. CSS-ben formáztam még az oldalak neveit úgy hogy mindegyik név jobb oldalán legyen 16px margó, kivéve az utolsónál:

```
[mat-button]:not(:last-child){margin-right:16px;}
```

A navigációs rész mindig az oldal tetején marad a `position`: `sticky` CSS beállítás miatt, valamint, hogy ne kerüljön semmi a navigációs rész elé a `z-index`-szel állítottam be.

3.3. Info modul

Az `Info` modulban az Útmutató oldalt készítettem el. Ebben a modulban csak egy `routing` fájlt kellett létrehoznom és beimportálnom az info.module.ts fájlba ezt fájlt. Az oldalt felbontottam két részre, ezt CSS-ben `display: flex`-szel állítottam be. Illetve a HTML fájlban elkészítettem az oldal tartalmát HTML-lel és CSS-sel.

3.4. Map modul

A `Map` modulban a Térkép oldalt készítettem el. A modulban a map komponens a szülőkomponens, a gyerekkomponensei pedig az `info, list` és `terkep`. Hozzáadtam egy `routing` fájlt a modulhoz. A szülőkomponens HTML-jében csak a `terkep` szelektorja található.

A terkep.module.ts-be beimportáltam a Forms, a MatDialog, a MatButton és a MatIcon modulokat az Angular Material-ból. A terkep.component.HTML-ben csak egy sor van, az pedig egy vászon elem. A vásznat arra használtam, hogy a Three.js-ben beolvasott objektumokat megjelenítsem.

Ahhoz, hogy érzékelje a canvas-t az alkalmazás, egy értéket kellett felvenni a canvas-nek. Ezt át kellett adni a component.ts fájlban egy ViewChild dekorátonak, hogy tudja melyik canvas-hez kell hozzáadni a grafikákat.

Mivel használtam dialógusablakot, a component.ts fájl konstruktorához hozzáadtam a MatDialog-ot. A dialógusablakokra akkor volt szükségem amikor a Three.js raycast érzékeli az oldalon a kattintást. Ezt bővebben a 4. fejezet 2. alfejezetében fejtettem ki. Viszont ahhoz, hogy a dialógusablak megjelenjen Angular-t is használni kellett.

Az onMouseDown függvényben elkészítettem, hogy mit csináljon a raycast amikor az objektumokra kattintok. Megadtam, hogyha végigmegy minden elemen, akkor egy megoldas nevű érték vegye fel az adott objektum nevét, valamint a második if-ben található elemeknél üres értéket vegyen fel.

```
intersects = raycaster.intersectObjects(scene.children);
      if (intersects.length !== 0) {
            this.megoldas = intersects[0]['object'].name;
            if   (intersects[0]['object'].name   ===   "   Cube.023_Cube.025"   ||
            intersects[0]['object'].name === "NurbsPath.003") {
                  this.megoldas = "";
            return;
            }
      return;
      }
}
```

A HTML fájlban a vásznon belül hozzáadtam egy kattintás eseményt, ami meghívja a component.ts fájlban található openDialog függvényt. Az openDialog-ban pedig meghívtam az InfoComponent-et, aminek átadtam a chosenMap értékét a megoldas-t, ami azt tárolta el, hogy melyik a kiválasztott objektum. Viszont ahhoz, hogy át tudjam adni az értéket, létre kellett hoznom a fájl legelején egy interface-t, ami eltárolja a megállók neveit. Ha a név és az adatbázis egyik eleme megegyezett akkor tudott csak továbbmenni és meghívni a dialógusablakot.

```
export interface DialogData {chosenMap: 'garadna'| 'kozepgaradna'}

openDialog() {
	if (this.megoldas.length != 0) {
		this.dialogRef.open(InfoComponent, {
			data: {
				chosenMap: this.megoldas
			}
		});
	}
}
```

A dialógusablakban megjelent információkat pedig az `InfoComponent` HTML fájlja töltötte be, attól függően, hogy milyen adatot továbbítottunk az eredeti fájlból.

```
<span *ngIf="data.chosenMap === 'garadna'">Garadna</span>
<span *ngIf="data.chosenMap === 'kozepgaradna'">Pisztrángtelep</span>
```

3.5. Not-Found modul

A `Not-Found` modult azért hoztam létre, hogy valamilyen jelzés megjelenjen akkor, amikor a felhasználó maga szeretne rákeresni az oldalakra, de félreüti a nevet, vagy pedig egy nem létező oldalra szeretne rákeresni.

Ezt a fájlt az első alfejezetben említett módon hívja meg a `routing` fájl, tehát ha egy egyezést sem talál az elérési útvonalak között, akkor ezt fogja betölteni. Emiatt szükség volt itt is egy `routing` fájl létrehozására is.

A HTML fájljában egy sor található, ami kiírja a képernyő közepére, hogy `Page not found`.

3.6. Splash modul

Ez a modul a kezdőoldalt tartalmazza. Ez az oldal nyílik meg amikor a felhasználó megnyitja az alkalmazást, valamint a Navigációs sávból két helyen is el lehet érni, a Lillafüredi kisvasútra és a Kezdőoldalra kattintva.

Az Angular Material-ból a MatCard-ot, és a MatRipple-t használtam, valamint a SplashRoutingModule-t amiket beimportáltam a splash.module.ts-be.

A splash.routing.module.ts fájlt hasonlóan kellett létrehozni mint az app-routing fájlt. Itt viszont nem új path-okat kellett létrehoznom, hanem a Routes tömb children-jei közé kellett felvennem az olyan modulokat, amiket szerettem volna használni. Ugyanúgy kellett hozzáadni a modulokat ahogy az app-routing-ban, tehát a path után a rövid nevét kellett adni az útvonalnak, ahogy szeretnénk hivatkozni rá, valamint a következő sorban pedig a modul útvonalát kellett betölteni relatívan, valamint hozzá kellett adni a modul nevét.

A Splash.component.HTML fájlban matCard-okat hoztam létre, amiben elneveztem a kártyákat. A mat-card már adott egy általános formázást, viszont hozzáadtam még a matRipple kiegészítőt a szebb vizualizáció miatt, és a matRippleColor-nál megadtam, hogy milyen legyen a szín amikor kattintok a kártyán. Az útvonalat pedig úgy jelenítettem meg ahogy a NavModul fejezetében is, tehát a rövidített nevet írtam be, ebben a formában:

```
<mat-card matRipple [routerLink]="'../info'" matRippleColor="#823b17" >
      <p>Útmutató</p>
</mat-card>
```

3.7. Trains modul

A Trains modulban a Vonatok oldalt készítettem el. A trains.module.ts fájlban az Angular Material-ból beimportáltam a MatCard, Forms, MatButton, MatIcon, MatDialog, MatTooltip modulokat. Valamint beimportáltam még a Routing modult, illetve a később bemutatott dialógusablakok moduljait.

A trains.component.HTML fájlban több gyerekkomponens szelektorját is használtam. Az app-list-et, az app-image-t, valamint az app-viewer-t. Az app-list-ben a lenyíló lista megjelenítését és az opciók feltöltését csináltam meg, az app-image a kiválasztott vonat képét, az app-viewer pedig a kiválasztott vonatról az információkat jelenítette meg. Input output direktívákkal oldottam meg az adatátvitelt a gyerek és a szülő komponensek között.

A Vonatok oldalon megjelenített adatokat a constants.ts fájljában gyűjtöttem össze, itt a TrainsObject nevet adtam a tömbnek. Ahhoz, hogy ezt használni tudjam a trains komponensben, be kellett importálnom ide, valamint felvettem egy trainObject nevű tömböt, ami a TrainsObject adatait kapja meg. Felvettem még egy chosenTrain nevű változót is, ami a későbbiekben azt jelöli, hogy melyik vonat van kiválasztva. A trains

modulban átadjuk az app-list-nek a trainObject-et, és ezt az adatot a gyerek komponensben mostmár a trainsObjectInput tömb tárolja el. Ez egy Input direktíva, ami azt jelenti, hogy a szülő fájlból kapjuk meg az adatot.

Az List modulban az Angular Material MatSelect modulját használtam, amit be is importáltam. A list komponensben a select option-nél az adatok betöltését strukturális direktívával oldottam meg. Ez végig iterál az opciók között és megjeleníti mindet. Tehát az app-list.HTML-ben a *ngFor=”let item of trainsObjectInput” és az item.nev -vel megjelentek a select-ben az vonatok nevei opcióként.

Szükségem volt egy value-ra ami azonosítja a betöltött adatokat, így az Angular-os ngValue-t használtam fel, aminek az előbb említett item értékét adom át. Ahhoz hogy vissza tudjuk juttatni a szülőnek hogy melyik az éppen kiválasztott elem, szükség volt egy kétirányú adatösszekötésre, amit a select-ben az alábbi módon oldottam meg: [(NgModel)]=”chosenTrain”. Az ngModel-hez be kellett importálni a Forms modul-t, hogy működjön az NgModule.

A list komponensben először az ngOnInit függvény indul el, amiben a chosenTrain-nek átadjuk a trainsObjectInput 0. elemét, ami a vonat nevei közül az első lesz. Ez lesz a kezdő opció a lenyíló listában. Ezután meghívódik a reload függvény, ami, ami azt csinálja, hogy a szülő komponensnek elküldi a chosenTrain értékét. Azért, hogy érzékelje a select a változást, felvettem egy change eseményt és hozzárendeltem a reload függvényt. Ekkor, ha más vonatot választok, frissül a chosenTrain és azt fogja a szülő komponens megkapni.

```
<div>
   <select mat-select (change)="reload()" [(ngModel)]="chosenTrain" >
      <option        *ngFor="let        obj        of        trainsObjectInput"
      [ngValue]="obj">{{obj.nev}}</option>
   </select>
</div>
```

Ahhoz, hogy át tudjuk adni az adatot a szülőnek, még szükségem volt egy Output direktívára, egy EventEmitter-re, amit az angular/@core-ból kellett beimportálni. Ezt a list komponensben vettem fel, trainObjectEmitter néven. Ennek segítségével küldtem el az adatot az előbb említett reload függvény segítségével a trains komponensbe:

```
this.trainObjectEmitter.emit(this.chosenTrain);
```

Amikor történik valami változás a gyerekben, tehát meghívódik a `trainsObjectEmitter`, akkor a szülőben meghívódik a `loadTrain` függvény, ami megkap egy bejövő eseményt. Ezt az eseményt pedig átadja a `chosenTrain`-nek, vagyis a szülő is megkapja mi lesz a kiválasztott vonat.

```
<app-list [trainsObjectInput]="trainObject" (trainObjectEmitter) = "loadTrain($event)" > </app-list>
```

Az adatok megjelenítéséért az `image` és a `view` gyerek komponensek feleltek. Az `image` komponens jelenítette meg az kiválasztott vonatról készült képet. Ehhez át kellett adni azt, hogy melyik vonatról van szó. Tehát a `chosenTrain` értékét kellett átadni az image komponensben található `trainInput`-nak. Ez egy `Input` változó, ami azt jelenti, hogy a gyerek a szülőtől kapta az adatot. Az image komponensben az `*ngIf`-nek átadtam, hogyha megkapja a `trainInput`-ot, akkor jelenítse meg a képet, aminek az elérési útvonalát így jelenítettem meg: `[src]=”trainInput.kep”`. Az src-t azért kellett így jelölni, mert kívülről érkezik az adat.

A `view` komponensnél is ugyanúgy kellett a `chosenTrain` adatait átadni, mint az `image`-ben, viszont itt nem az `src`-nek adtam meg a `trainInput`-ot, hanem a `trainInput` különböző értékeit jelenítettem meg.

Ahhoz, hogy a modelleket megjelenítsem külön, egy gomb segítségével jelenítettem meg dialógusablakot az aktuálisan kiválasztott vonathoz. Tehát a gomb kattintás eseményének az `openDialog` függvényt adtam át. Az `opanDialog`-ban pedig lekezeltem `if`-fel, hogy a `chosenTrain` függvényében melyik vonathoz tartozó dialógusablakot hívja meg.

4. Three.js

Ebben a fejezetben azt fejtem ki, hogy használtam fel a Three.js-t a webalkalmazásomhoz, kitérek arra, hogy töltöttem be a Blender modelleket, valamint bemutatom a `Raycast` felhasználását.

4.1. Blender modellek betöltése

Mivel a modelleket a Blender-ben hoztam létre, a Three.js-be kellett betölteni a fájlokat. Ha egy Blender modellnek van textúrája is, amit a Blender-ben hoztam létre, akkor exportáláskor két darab fájl fog létrejönni. Az egyik egy .obj fájl, ami a geometriákat tárolja el, a másik pedig egy .mtl fájl, ami az anyagokat. Mivel két darab fájlom volt minden modellnél, ezért figyelni kellett, hogy mind a kettő fájlt beolvastam-e. A modell beolvasáshoz szükségem volt kettő betöltő függvényre.

Az .obj fájl betöltéséhez egy `OBJLoader` nevű kiegészítőt kellett importálnom a fájlba, az .mtl fájl betöltéséhez pedig egy `MTLLoader`-t kellett importálnom, hogy a modellek textúráját is fel tudjam használni. Az `MTLLoader`-t mindig az `OBJLoader`-rel kellett együtt importálni. Külön csak az `OBJLoader`-t lehet használ akkor, amikor nincs textúrázva a modell Blender-ben.

Legelőször az anyagot töltöttem be az `MTLLoader` .load függvényével, amihez szükségem volt a betöltendő modellem .mtl fájl elérési útvonalára. Majd ezután a geometriákat töltöttem be az `OBJLoader`-rel. Ehhez pedig a modell .obj fájl elérésére volt szükségem. Majd a beolvasott `THREE.Group Mesh` gyerekeinek objektumait bejárással értem el, ha szerettem volna a beolvasás után módosítani őket, például a nevét, vagy a méreteit.

Nem töltöttem be minden megállót, mivel mindegyik ugyanolyan, így csak egyet töltöttem be. A házakat exportálás előtt össze kellett vonni, hogy a `raycast` egy tárgyként érzékelje. Azt az egy betöltött házat a `clone()` függvénnyel megsokszoroztam annyiszor ahány megállóra szükségem volt. Ezután elkellett neveznem az összes leklónozott megálló gyerekét, ezt a `traverse()` bejárásával tudtam megcsinálni, amin belül adtam nekik egy nevet. Erre azért is volt szükség, hogy a `raycast` érzékelje, hogy éppen melyik megállóra kattintok, a másik indok pedig, hogy mindegyiket másik helyre kellett tenni a térképen. Ezután mindegyik leklónozott objektumot hozzáadtam a színtérhez.

4.2. Raycast

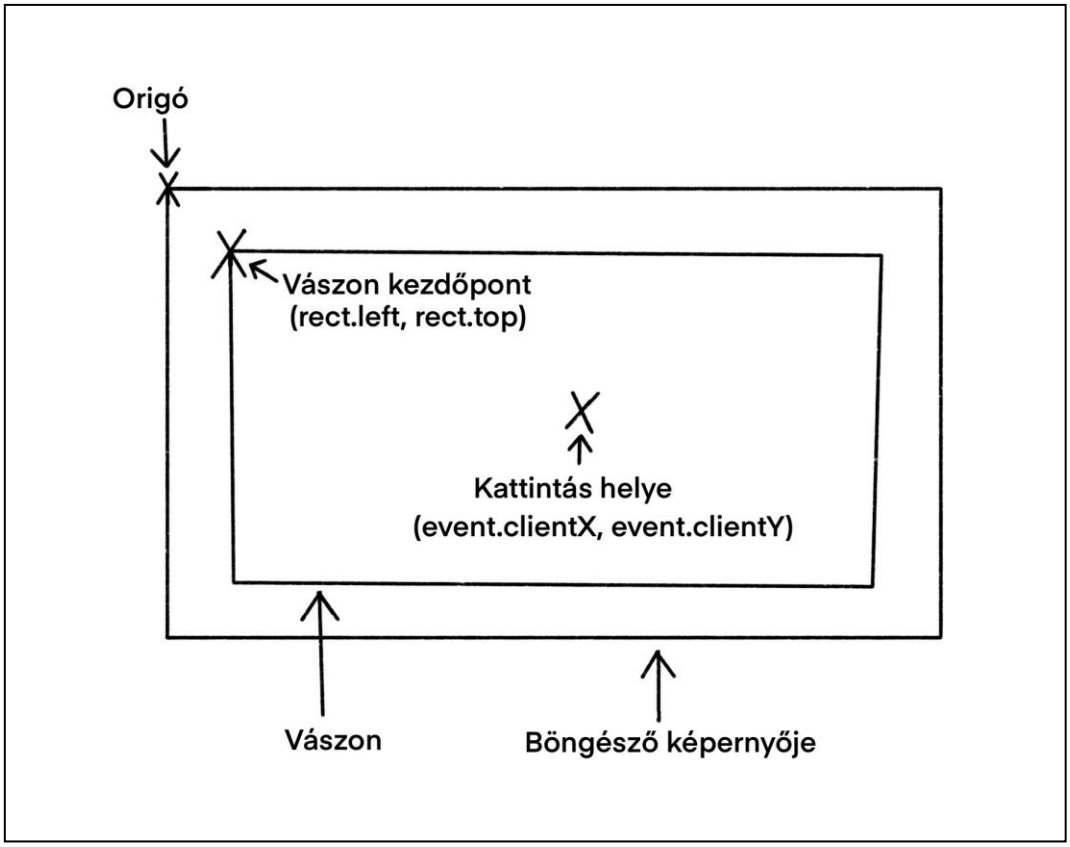
A Three.js egyik hasznos osztálya a `Raycast`, amire szükségem volt a weboldal elkészítéséhez, többek között ezért is választottam a Three.js-t. Arra jó, hogy érzékeli, hogy milyen objektum felett van az egér. Legtöbbször arra szokták használni, hogy megváltoztassák az objektumok tulajdonságait.

A `raycast`-ot a Térkép oldalon használtam. Minden megállónál más információt akartam megjeleníteni és érzékelni kellett, hogy melyik megállóra kattintok. Az sem volt mindegy, hogy megállóra vagy pedig máshova kattintok, mert ha máshova kattintok akkor nem akartam, hogy bármi megjelenjen. Illetve azt csinálta, hogyha más megállóra kattintok, mindig más információ jelenjen meg. A legjobb megoldás a `raycast` volt ehhez.

```
var intersects;
raycaster = new THREE.Raycaster();

const rect = canvas.getBoundingClientRect()
const x = event.clientX - rect.left
const y = event.clientY - rect.top

mouse.x = (x / window.innerWidth) * 2 - 1;
mouse.y = - (y / window.innerHeight) * 2 + 1;

raycaster.setFromCamera(mouse, camera);

intersects = raycaster.intersectObjects(scene.children);

this.megoldas = "";

if (intersects.length !== 0) {
    return;
}
```

A `raycast` az alábbiak szerint működik. A raycast bemenetként megkapja az event objektumot. A kezdőpont az origó. A vászon kezdőpontja a rect.left és a rect.top értékekkel kerül definiálásra, ezt kérjük le a `getBoundingClientRect`-tel a böngészőtől. A kattintás eseményt az event.clientX és event.clientY jelenti. Mivel a kattintás az origótól számít, ahhoz, hogy a vásznon belüli (x, y) koordinátákat megkapjuk, az event.clientX és event.clientY

koordinátákból ki kell vonni a vászon kezdőkoordinátáit, a rect.left és rect.top értékeket. (4.1. ábra). Ez a `const x` és a `const y` lesz. Valamint a mouse.x és mouse.y soraiban található műveletekkel kiszámoltam a koordináták pontos helyzetét a képernyőn.



4.1. ábra: Új kezdőpont kiszámítása

A következő sorban beállítottam a sugár pozícióját. A `setFromCamera` segítségével nem kellett megadnom a sugár kezdőpontjait és az irányt, amelybe a sugár vetül. Az első parameter az egér, amely a kurzor koordinátáit tartalmazza. A második pedig a `THREE.Camera` a jeleneten belül. Ezután hozzáférhettem a jeleneten belüli objektumokhoz, amiket metszett a sugar, ezt az `intersectObjects` segítségével tehettem meg.

A következő sorokra a dialógusablak megjelenítéséhez volt szükségem, ezt a 3.4. fejezetben fejtem ki. Valamint a `raycast`-hoz szükség van ahhoz hogy a beolvasott modelleknek egyedi legyen a neve, ezt úgy oldottam meg hogy az összes modell gyerekét elneveztem a `traverse` függvényben, ezt 4.1-es alfejezetben fejtettem ki bővebben.

5. Felhasználói útmutató

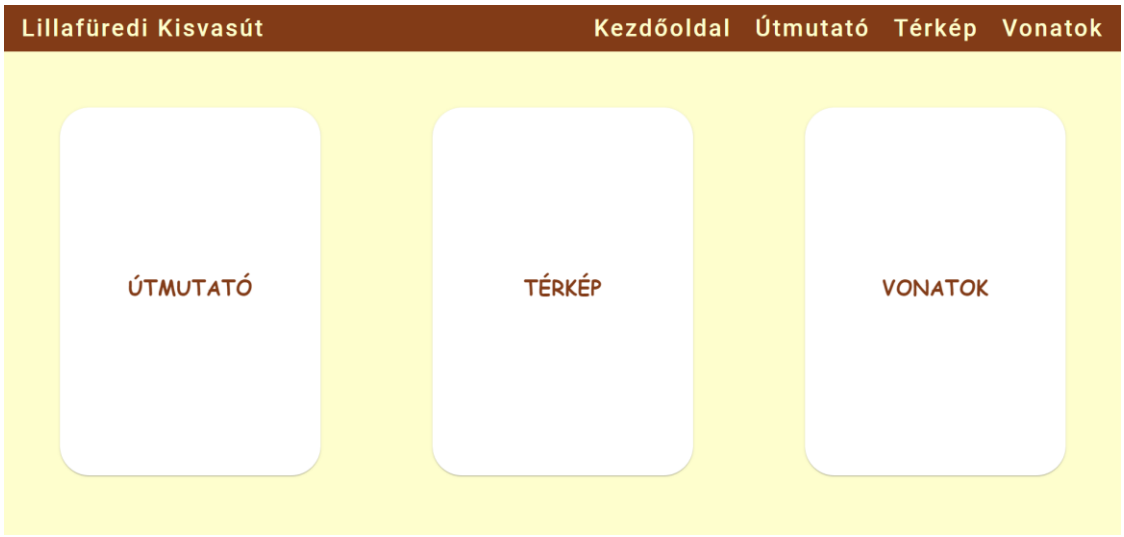
Ebben a fejezetben bemutatom, milyen lett a kész webalkalmazás, valamint azt hogyan kell a webalkalmazást használni egy olyan embernek, aki most találkozik a webalkalmazással először.

A weboldalt végül négy külön oldalra bontottam a jobb átláthatóság kedvéért. Az első a kezdőoldal, ami a navigációt segíti, aztán egy Útmutató oldal, ahol információt lehet kapni arról, hogy kell használni az alkalmazást. A következő oldal a Térkép, ahol a Lillafüredi kisvasút vonathálózatát és a nevezetességeket lehet megtekinteni és a negyedik oldal pedig a Vonatok, ahol a vonatokról lehet információt szerezni.

Minden oldal tetején ugyanaz a navigációs sáv található. Ez elősegíti a könnyebb navigációt a weboldalon. A nevekre való kattintáskor elirányítja a felhasználót a megfelelő oldalra. A baloldalon található cím, illetve a jobb oldalt található Kezdőoldal a kezdőoldalra navigál, a többi pedig a nevéből adódóan egyértelmű melyik oldalt nyitja meg.

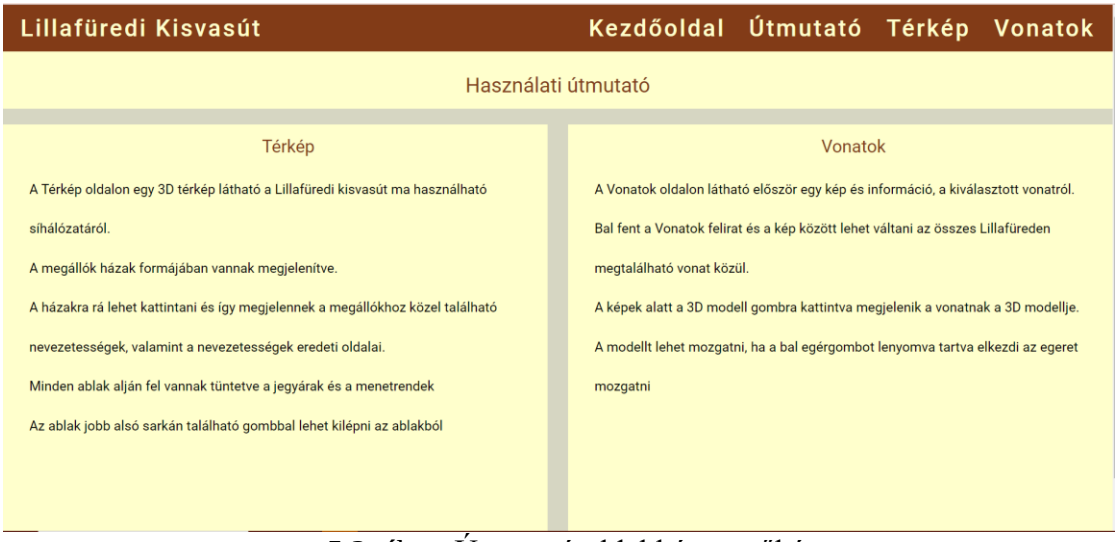
5.1. Kezdőoldal és Útmutató

Amikor betölt a weboldal legelőször, a kezdőoldal jelenik meg. Ennek az oldalnak a funkciója az, hogy megmutassa egyben, hogy milyen oldalak vannak a weboldalon, valamint segítsen a navigációban. Ha a felhasználó el szeretne jutni a kezdőoldal felhasználásával az egyik oldalra, akkor rákattint az egyik négyzetre, aminek hatására megjelenik a kívánt oldal (5.1. ábra).



5.1. ábra: Kezdőoldal képernyőképe

Az Útmutató oldal ahogy a neve is tartalmazza azért van, hogy a másik két főoldal használatát segítse. Azért hoztam létre ezt az oldalt, hogy a nem mindenki számára egyértelmű dolgok be legyenek mutatva és ezáltal mindenkinek könnyebb legyen az oldal használata. Leírja például, hogy mire kell kattintani hogyha a vonatok között szeretnénk váltani vagy ha szeretnénk információt szerezni arról, hogy milyen árak vannak a kisvasút használatakor, akkor azt hol keressük (5.2. ábra).



5.2. ábra: Útmutató oldal képernyőképe

5.2. Térkép

A Térkép nevű oldalon található a Lillafüredi kisvasút sínhálózatáról készült terepasztal. Itt meg lehet tekinteni, hogy melyik szárnyvonalon lehet használni a kisvasutat, valamint információt lehet szerezni a nevezetességekről. Minden ház egy megállót jelent a kisvasút mentén. (5.3. ábra)

Ha információt szeretne kapni a felhasználó arról, hogy melyik megálló közelében milyen nevezetességek találhatóak, akkor rá kell kattintani az egyik házra a térképen, és ekkor felugrik egy ablak, ahol megjelenik az összes információ a megállókról (5.4. ábra). Majdnem minden megállóhoz az odajutást is megtalálja itt, valamint azt, hogy mit tud megnézni a környéken. A legtöbb nevezetességnek a neve alá van húzva, ami azt jelenti hogyha rákattint a névre, elnavigálja a nevezetesség eredeti oldalára, ahol több információt találhat róla. Az északi szárnyvonalat alkalmanként lehet használni, tehát ennél az útvonalnál a szárnyvonalnak a történetéről lehet olvasni. Valamint megtalálható egy link az oldal alján, ahol információt lehet kapni arról, ha meghirdetnek egy járatot ezen a szárnyvonalon.

5.3. ábra: Térkép oldal képernyőképe

Ha információt szeretne a jegyárakról a felhasználó, akkor azt minden megálló információi után találja csoportosítva aszerint, hogy milyen jegyet szeretne venni, teljes árut, kedvezményeset, esetleg egész naposat, vagy csoportosat, és meg lehet tekinteni, hogy mennyibe kerül az egy útra szóló és az oda vissza szóló jegy. A menetrend elérhetősége a megállóknál felugró ablak legalján található, a jegyárak alatt. Ha rákattint az aláhúzott szövegre, elnavigálja a látogatót a Lillafüredi kisvasút hivatalos weboldalára, ahol megtalálható az évszaknak megfelelő menetrend.

5.4. ábra: Megálló dialógusablaka a Térkép oldalon

5.3. Vonatok

Ezen az oldalon információt lehet szerezni az összes vonatról, amit valaha használtak a Lillafüredi kisvasútnál (5.5. ábra). A képernyő két részre lett bontva a jobb átláthatóság miatt. A képernyő bal oldalán egy kép jelenik meg a kiválasztott vonatról. A jobb oldalon pedig a felhasználó információkat talál az éppen kiválasztott a vonatról. A képernyő bal oldalán a fénykép és a Vonatok felirat között található egy fül. Erre kattintva lenyílik egy lista, ami tartalmazza az összes vonat nevét. Ha rákattint egy másik vonat nevére akkor ez lesz az aktuálisan kiválasztott vonat, amiről megjelenik az információ.

5.5. ábra: Vonatok oldal képernyőképe

Ha a kép alatti 3D modell nevű gomb fölé viszi az egeret, akkor információ jelenik meg arról hogyan lehet mozgatni a modelleket a megjelenő ablakba. Ha rákattint a gombra, megjelenik egy 3 dimenziós modell a vonatról, ami magától forog körbe (5.6. ábra). Ha lenyomja a bal egérgombot, lenyomva tartja és ekkor elkezdi mozgatni az egeret akkor el tudja kezdeni mozgatni a vonatról készült modellt. Ekkor olyan irányból lesz megtekinthető a vonat ahogy a felhasználó szeretné. Ha az ablak mellé kattint a felhasználó, akkor ki tud lépni innen.

5.6. ábra: Vonat modell megjelenítése dialógusablakon

Irodalomjegyzék

[1] Hivatalos Three.js dokumentáció: https://threejs.org/docs/index.HTML

[2] Számítógépes grafika jegyzet: https://www.inf.u-szeged.hu/~tanacs/threejs/index.HTML

[3] Hivatalos Angular dokumentáció: https://angular.io/docs

[4] Hivatalos Blender dokumentáció: https://docs.blender.org/manual/en/latest/index.HTML

[5] Spencer Grey: Mind-Melding Unity and Blender for 3D Game Development: Unleash the power of Unity and Blender to create amazing games

Nyilatkozat

Alulírott Tóth Fanni gazdaságinformatikus BSc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Képfeldolgozás és Számítógépes Grafika Tanszékén készítettem, gazdaságinformatikus BSc diploma megszerzése érdekében.

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

Tudomásul veszem, hogy szakdolgozatomat a Szegedi Tudományegyetem Diplomamunka Repozitóriumában tárolja.

Szeged, 2023. május.12.

Tóth Fanni

Köszönetnyilvánítás

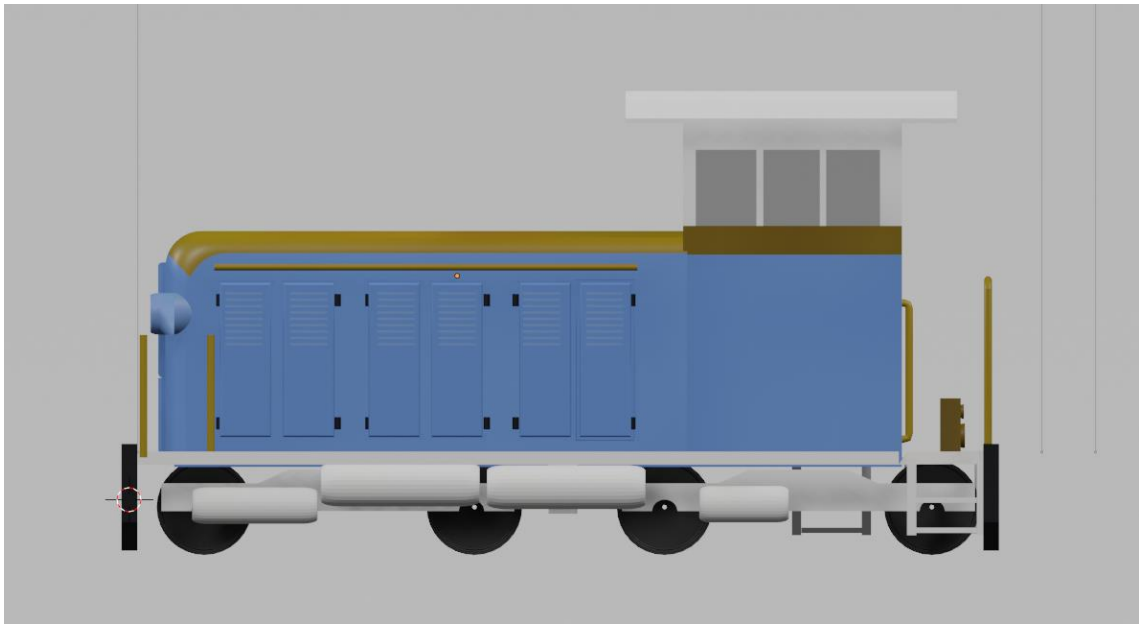
Ezúton szeretnék köszönetet mondani a témavezetőmnek, Dr. Tanács Attila egyetemi adjunktusnak a szakdolgozatom elkészülése folyamán való segítségéért, szakmai tanácsiért és türelméért.

Mellékletek

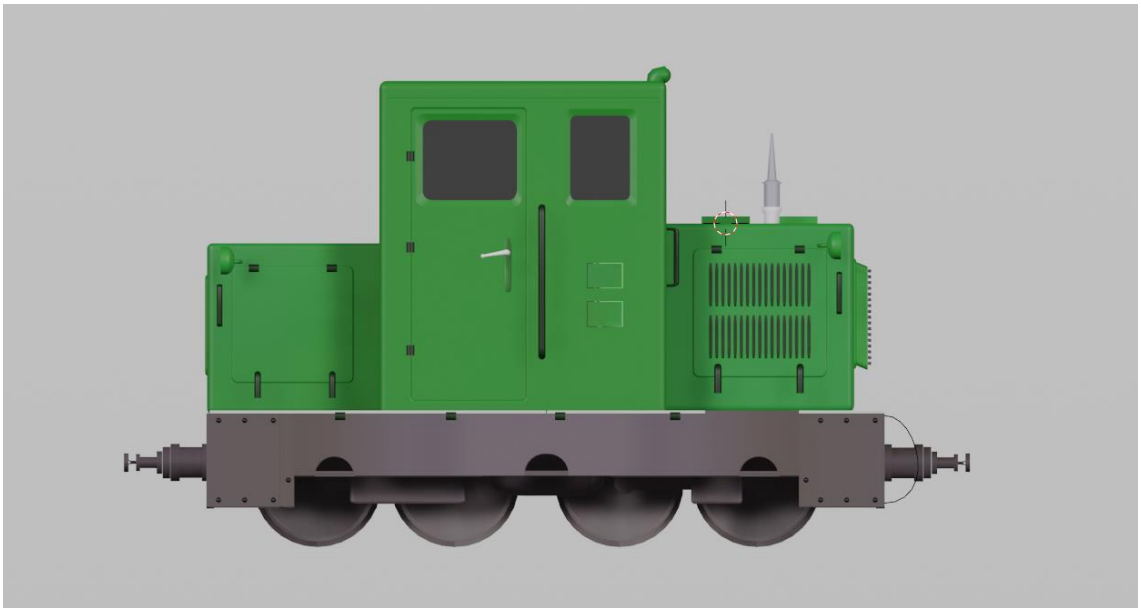
A mellékletbe az összes elkészült Blender modellemet teszem nagy méretben. Ezeket kisebb méretben már megjelenítettem a 2. fejezet 1. alfejezetében. Az összes modellt textúrázva helyezem el, a végleges állapotában.

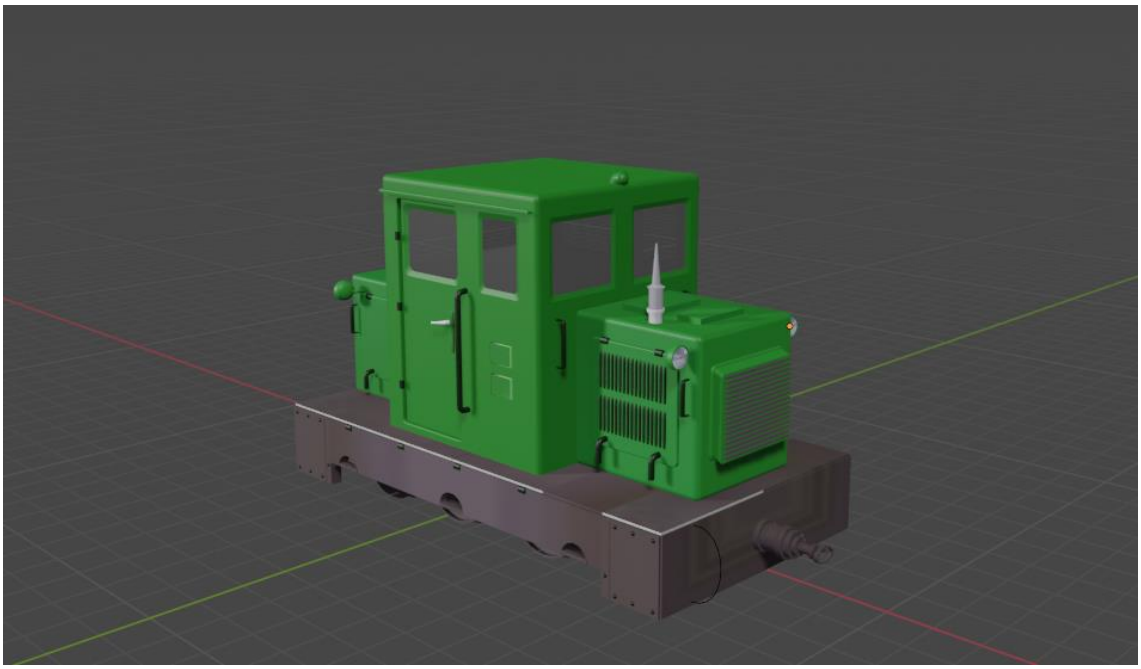
Mk48 vonat:

Mk48 hibrid vonat:



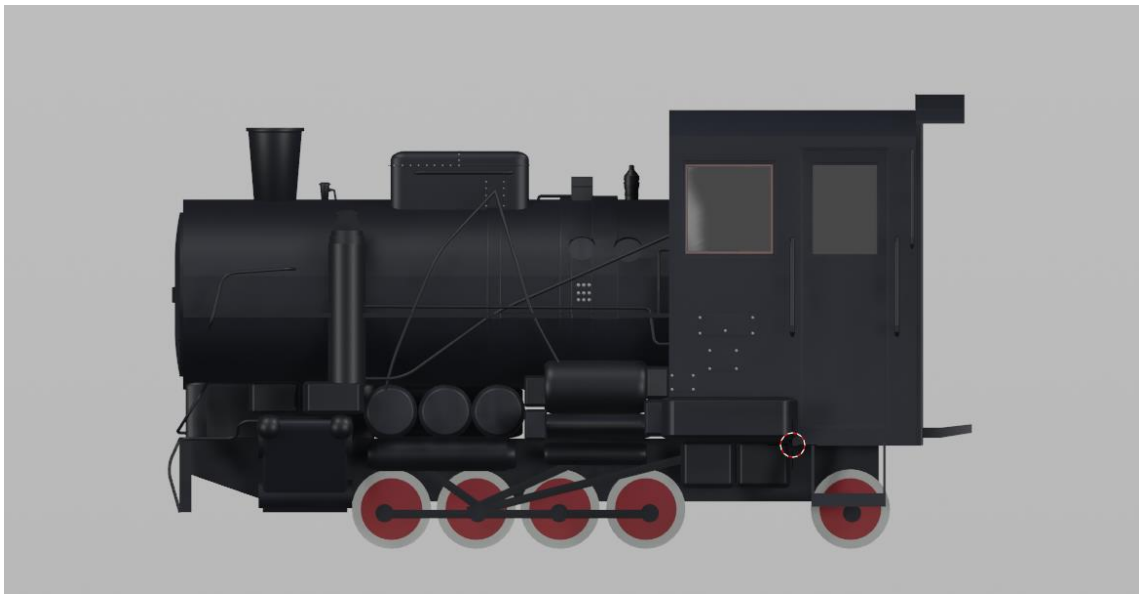
C-50 vonat:

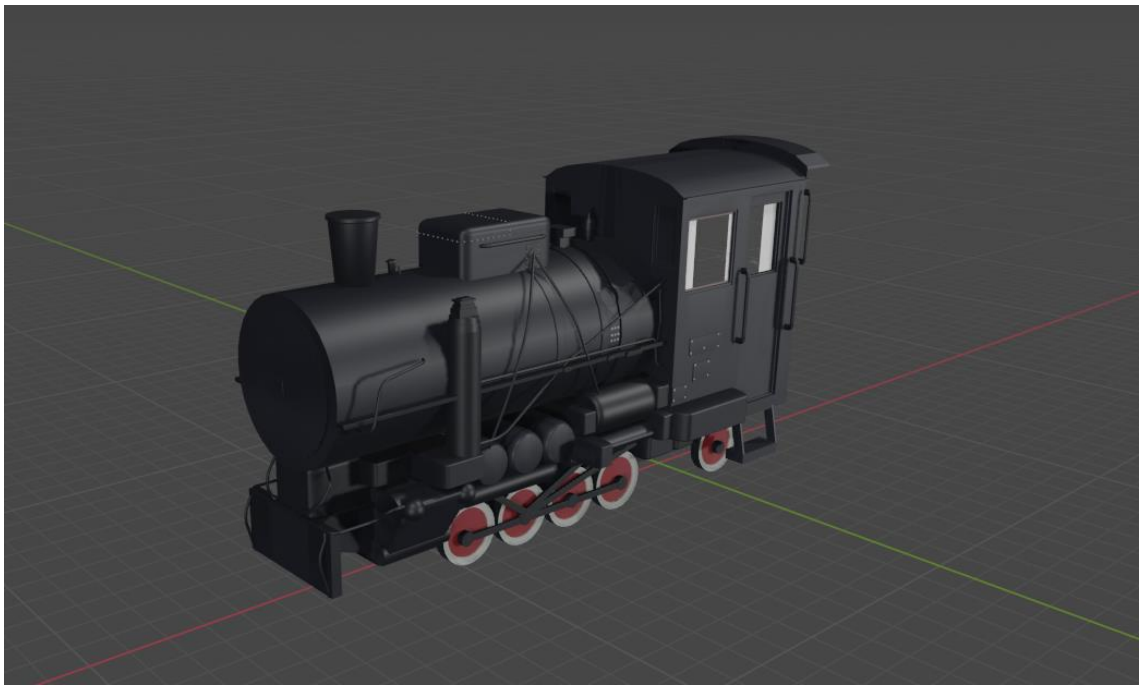




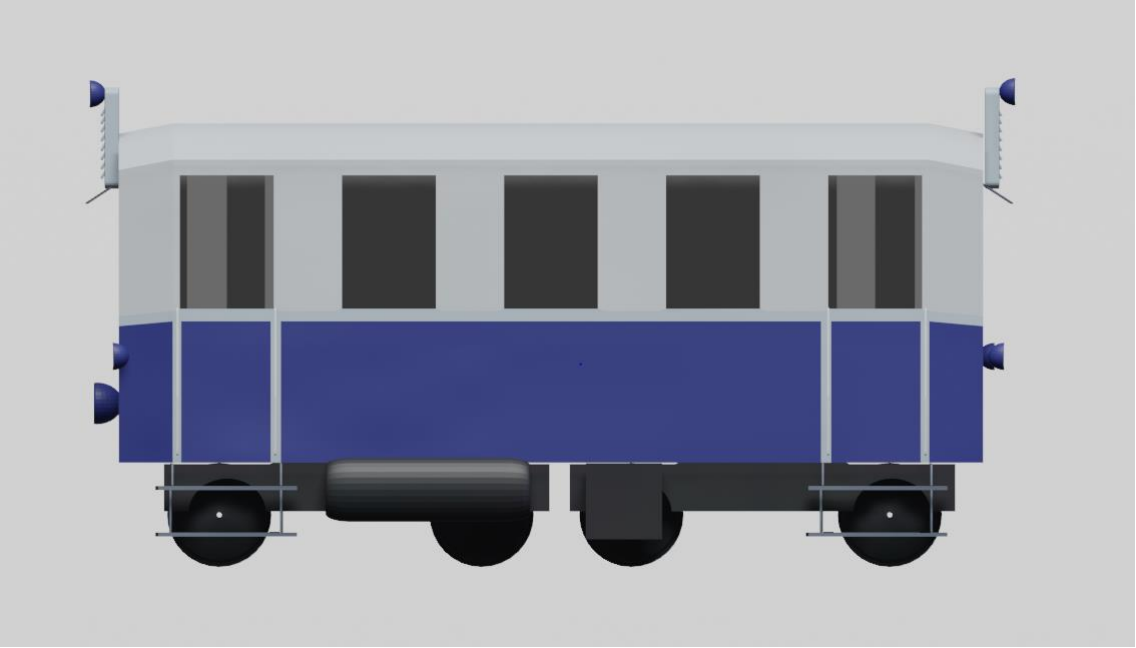
B-26 vonat:

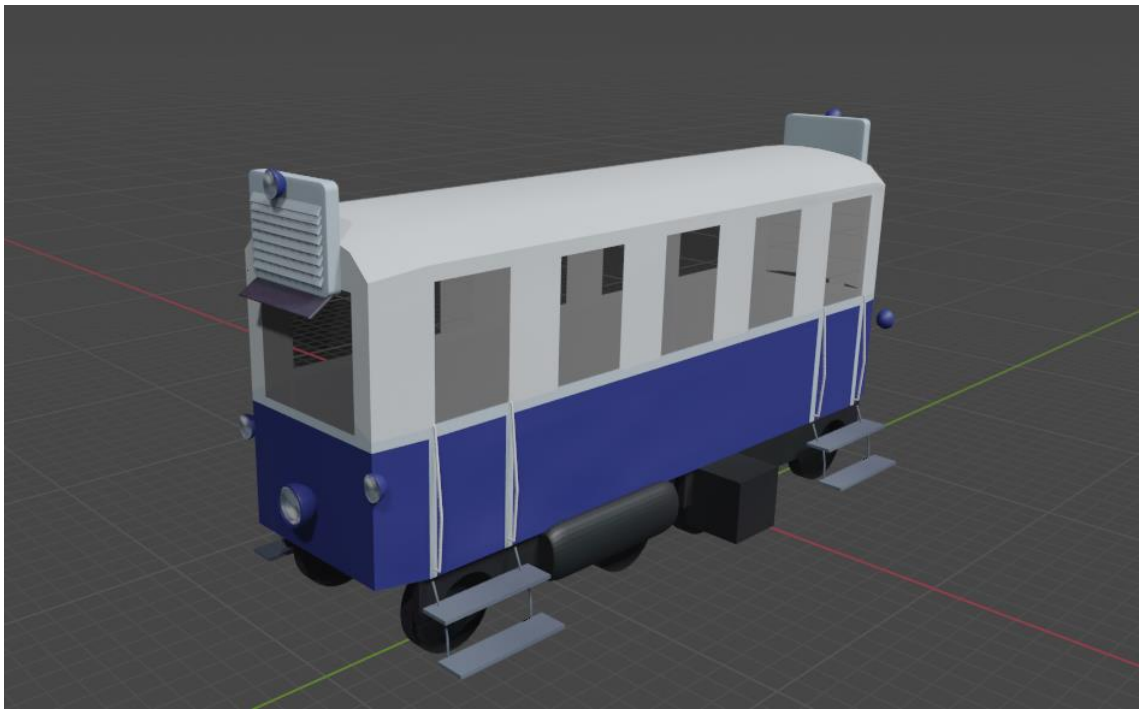
Kv-4 vonat:



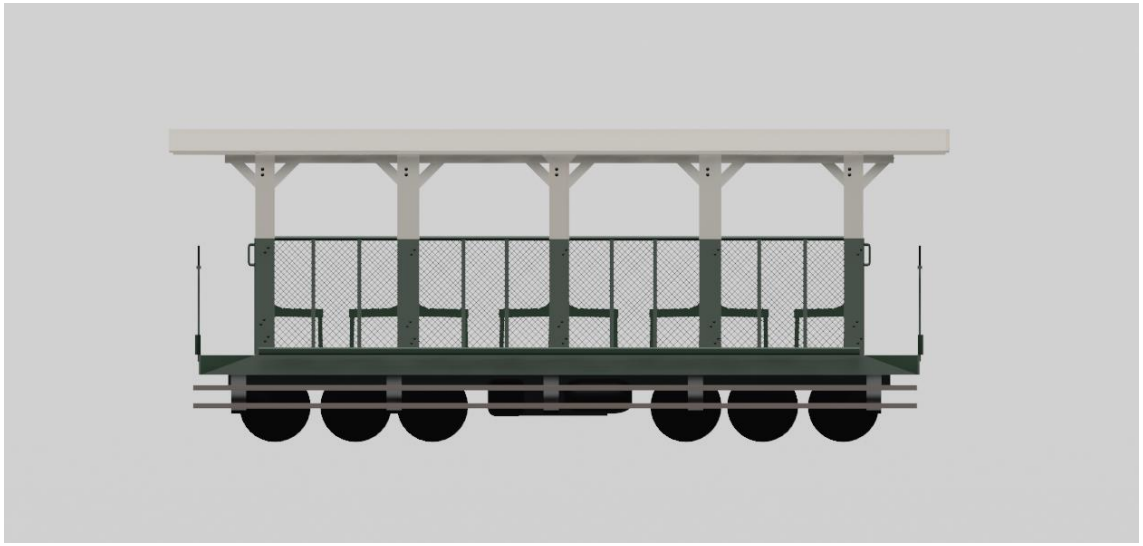


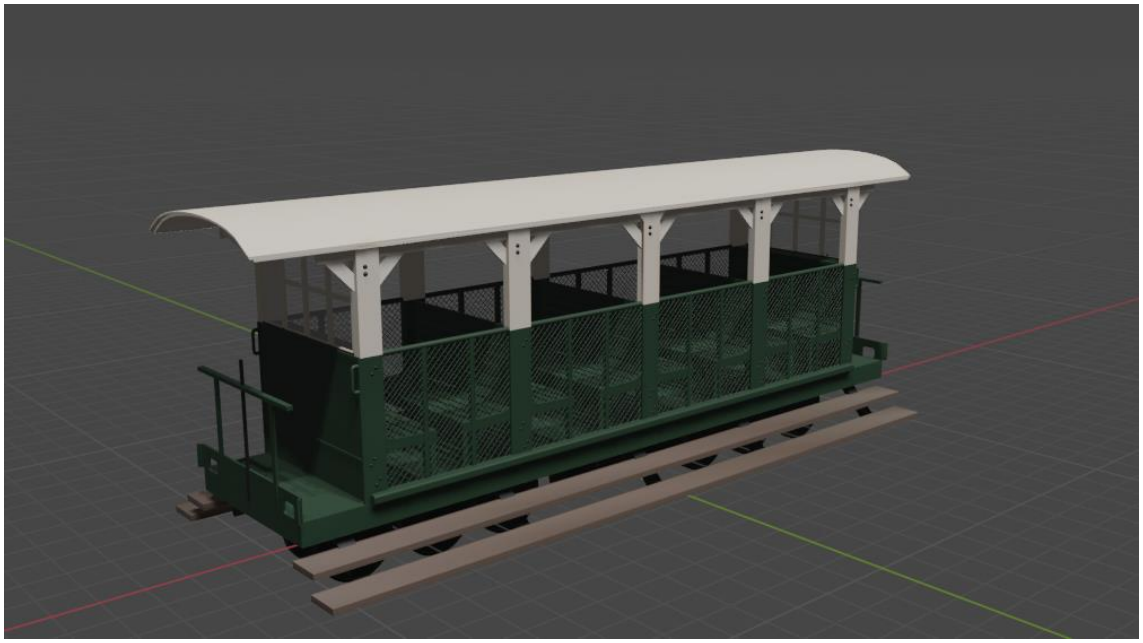
Abamot motorvonat:



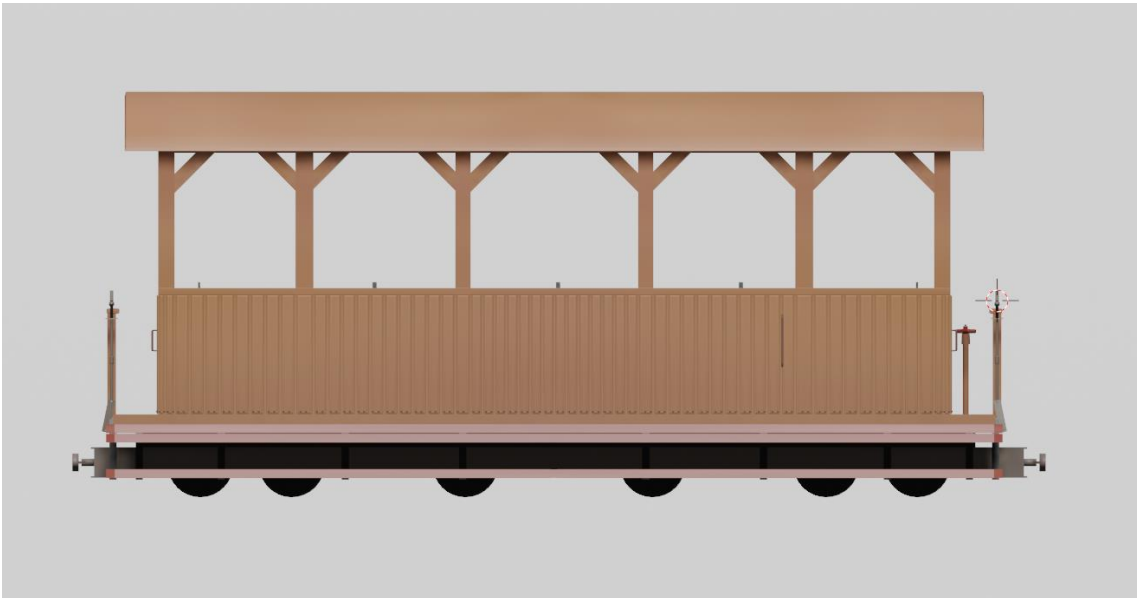


Diamond kocsi:

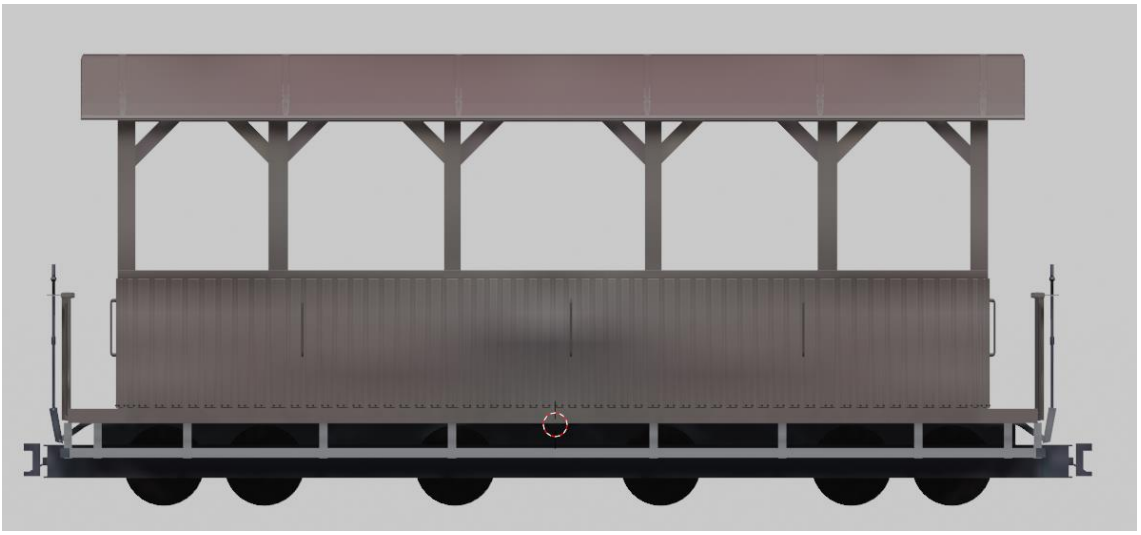




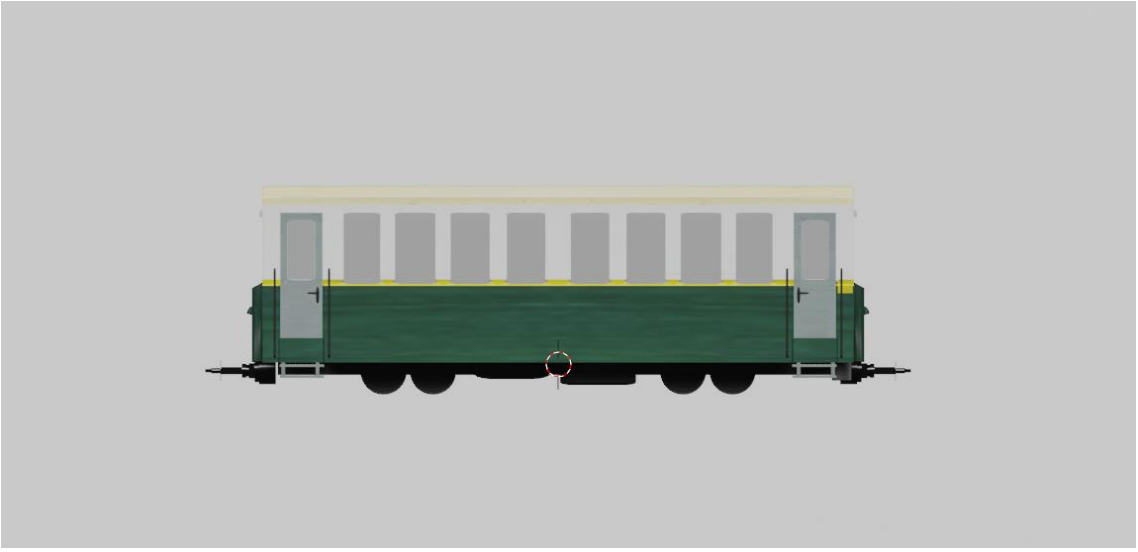
Kis Máv kocsi:



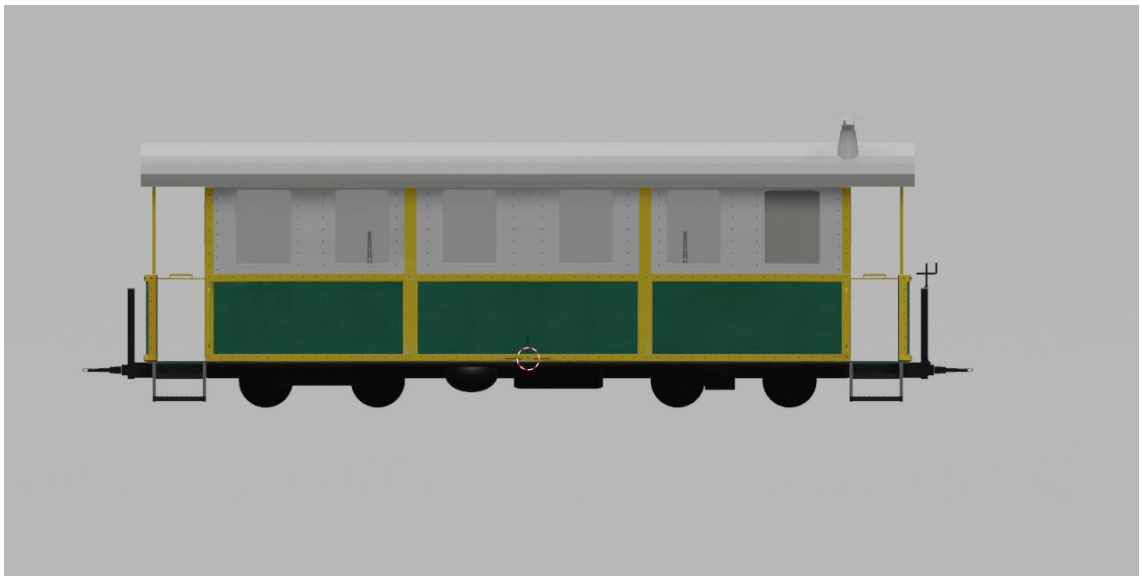
Nagy Máv kocsi:

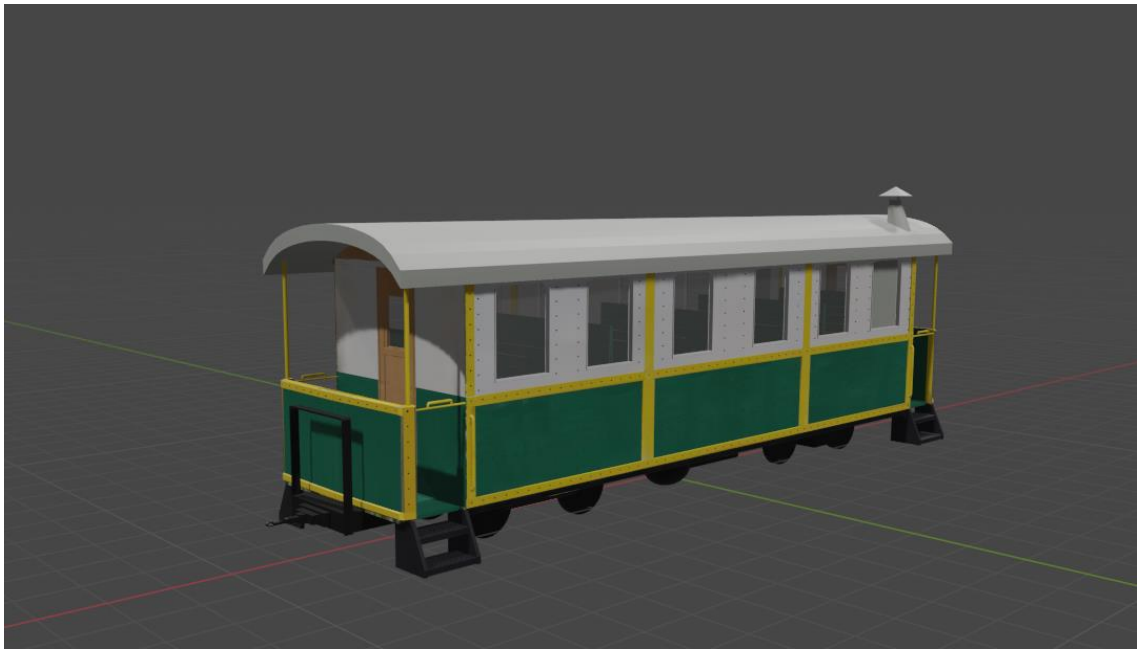


Bax kocsi:

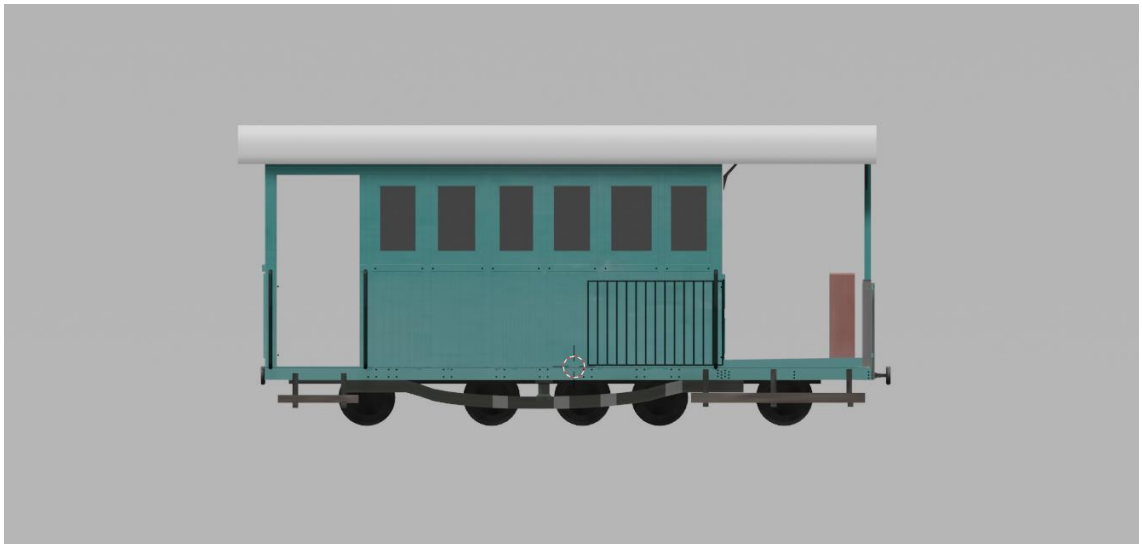


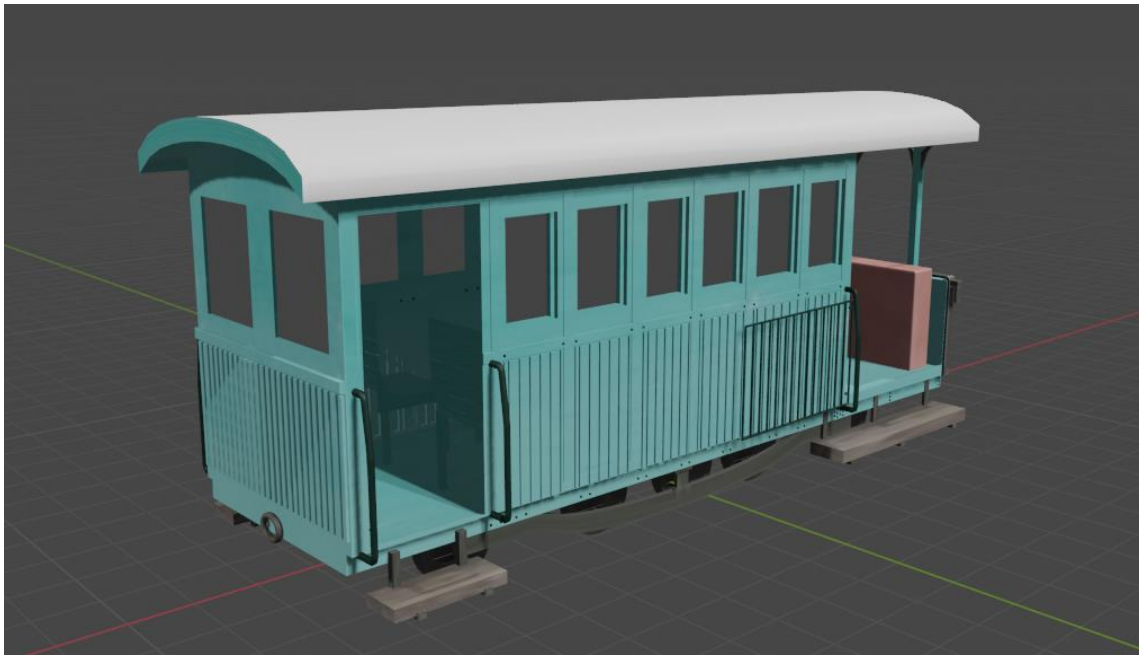
31-es kocsi:



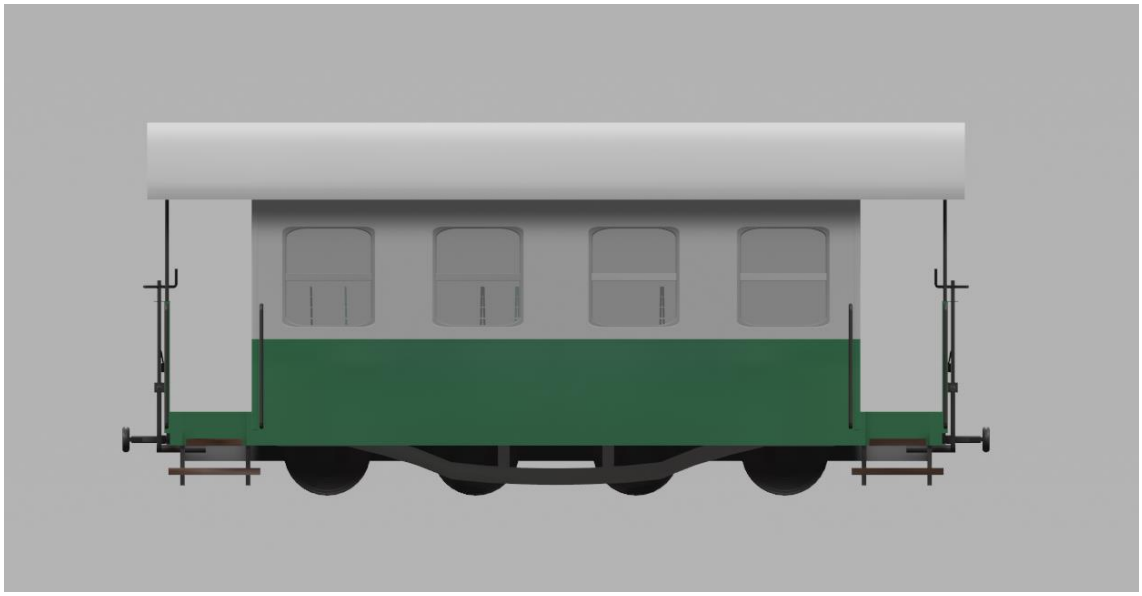


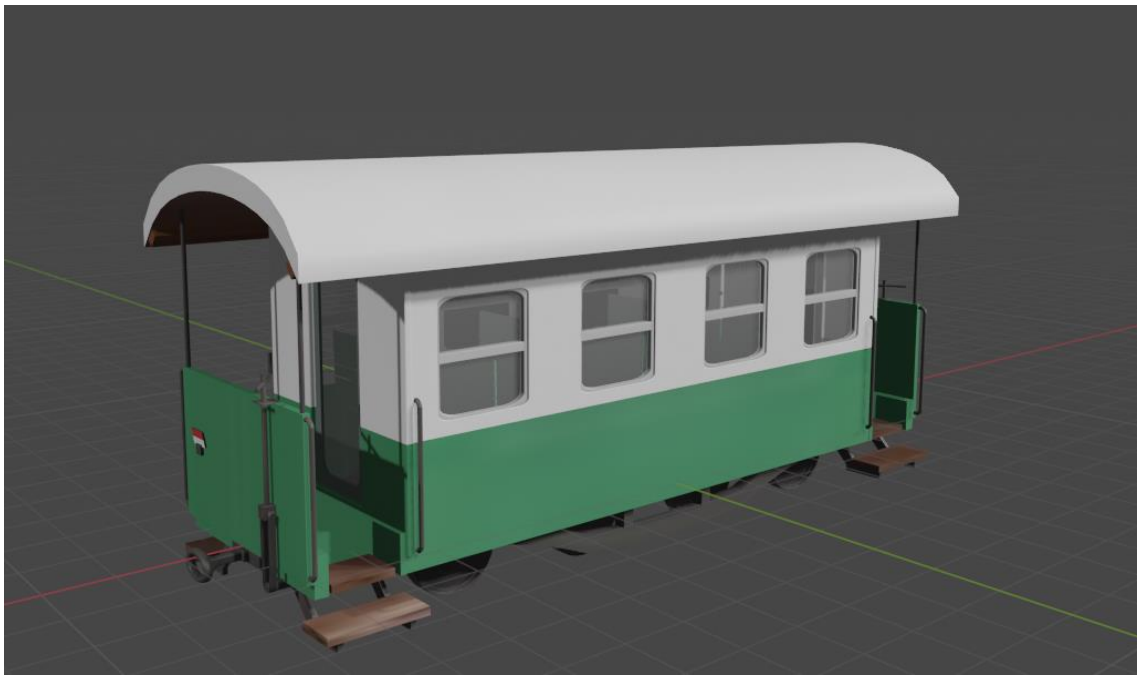
32-es kocsi:





33-as kocsi:





Megálló modellje:

Terepasztal modellje: