Eötvös Loránd Tudományegyetem

Informatikai Kar

Információs Rendszerek

Tanszék

Vasúti vágányhálózat- és menetrendtervező alkalmazás

Témavezető:
Dr. Gombos Gergő
egyetemi docens

Szerző:
Ramel Dániel
programtervező informatikus BSc

Budapest, 2025

Tartalomjegyzék

1. fejezet

Bevezetés

1.1. Háttér

A nagy volumenű kötött pályás infrastrukturális beruházások tervezése és kivitelezése a közlekedésfejlesztés legköltségesebb és legkomplexebb feladatai közé tartozik. A modern vasúti és elővárosi közlekedés fejlesztésében paradigmaváltás figyelhető meg: a hangsúly a puszta infrastruktúra-építésről áttevődik a szolgáltatás-orientált, úgynevezett menetrend-alapú tervezésre. A dolgozatomban bemutatott alkalmazás célja, hogy ezt a tervezési folyamatot támogassa egy olyan iteratív szimulációs környezet biztosításával, amely lehetővé teszi a menetrendi koncepció és a pályageometria együttes optimalizálását.

A hagyományos megközelítésben gyakran először valósul meg az infrastruktúra fejlesztése, és a szolgáltatást (menetrendet) a már megépült fizikai korlátokhoz igazítják. Ezzel szemben a fejlesztett szoftver alapvetése, hogy a legjobb költség-haszon arányú ráfordításhoz folyamatos iteráció szükséges a tervezési fázisban. Az alkalmazás módot ad arra, hogy a felhasználó párhuzamosan módosítsa a menetrendi struktúrát és a szükséges infrastruktúra elemeit (például kitérők, jelzők elhelyezése, sebességkorlátozások), egészen addig, amíg a rendszer el nem éri a kívánt egyensúlyi állapotot.

Kiemelt tervezési szempont volt, hogy az alkalmazás specifikusan támogassa az ütemes menetrendek (ITF - Integrált Ütemes Menetrend) létrehozását és vizsgálatát. A hazai és nemzetközi tapasztalatok azt mutatják, hogy az utasok számára kiszámíthatóbb, rendszeres időközönként ismétlődő, csatlakozásokra épülő menetrendi struktúrák válnak be a legjobban, mivel ezek növelik a rendszer vonzerejét

és átláthatóságát. A szoftver segítségével pontosan modellezhető, hogy egy adott ütemes struktúra bevezetéséhez milyen minimális, de elégséges infrastrukturális beavatkozások szükségesek [1].

Ennek a megközelítésnek a létjogosultságát a gazdasági tényezők is alátámasztják. A menetrend-alapú tervezés bizonyítottan erőforrás-hatékonyabb: a módszer alkalmazásával elkerülhető a felesleges vágányok, jelzők és kereszteződések kiépítése. Beruházás csak ott történik, ahol a célzott immáron ütemes menetrendi struktúra azt valóban megkívánja. Továbbá ez a tervezési mód segíti a stabil, megbízható szolgáltatás biztosítását, optimalizálja a hálózat kihasználását, és minimalizálja a késések, valamint a torlódások kockázatát.

Hosszú távon az alkalmazás által támogatott módszertan lehetővé teszi a szisztematikus, hálózati szintű beruházásokat. A fejlesztések így nem szigetszerűen, egymástól függetlenül valósulnak meg, hanem összhangban vannak a vonalak hálózati szerepével és a szolgáltatási igények valós képével.

Ugyanakkor a szoftver fejlesztése során figyelembe kellett venni ezen tervezési filozófia korlátait is. A menetrend, a hálózat és az infrastruktúra integrált tervezése rendkívül komplex feladat, amely sok szereplő operátorok, infrastruktúra-gazdák, finanszírozók összehangolt munkáját igényli. Kritikus szempont a rendszer rugalmassága is: ha a tervezett menetrend túl feszes, a valós utas- és forgalmi igények változását nehéz lekövetni, ami túlszabályozáshoz vagy az infrastruktúra alulkihasználtságához vezethet. Ezért a dolgozat keretében bemutatott alkalmazás olyan egyensúlyt keres, amely egyesíti a menetrend-központú tervezés előnyeit a gyakorlati megvalósíthatósággal és a rendszer adaptivitásával. Összességében a fejlesztett alkalmazás nem helyettesíti az infrastruktúra-beruházást - hiszen a kapacitási vagy sebességi problémák fizikai beavatkozást igényelnek –, hanem célzottabbá és optimalizáltabbá teszi azt. A cél egy olyan eszköz biztosítása, amely támogatja a „menetrend az első” elvet, de teret enged a szükséges infrastrukturális korrekciók szimulációjának is, így biztosítva a leghatékonyabb megoldást a modern közlekedésfejlesztésben.

1.2. Célkitűzések

2. fejezet

Felhasználói kézikönyv

Jelen fejezet célja, hogy átfogó útmutatást nyújtson az alkalmazás telepítéséhez és kezeléséhez. A dokumentáció részletesen bemutatja a szoftver rendszerigényét, a telepítés lépéseit, a felhasználói felületet, valamint a hatékony tervezéshez és forgalomirányításhoz szükséges munkafolyamatokat.

2.1. Rendszerkövetelmények és Telepítés

2.1.1. Rendszerkövetelmények

A szimulációs szoftver zökkenőmentes futtatásához az alábbi hardver- és szoftverkonfiguráció szükséges. Bár az alkalmazás 2D megjelenítést használ, a komplex útvonalkeresési algoritmusok és a szimulációs logika miatt a processzor teljesítménye meghatározó.

Komponens	**Minimális követelmény**	**Ajánlott konfiguráció**
Operációs rendszer	Windows 10, macOS 10.13+, vagy modern Linux disztribúció	Windows 10/11 (64-bit)
Processzor (CPU)	Dual-core processzor (pl. Intel Core i3 / AMD Ryzen 3)	Quad-core processzor, 3.0 GHz+ (pl. Intel Core i5 / AMD Ryzen 5)
Memória (RAM)	4 GB	8 GB vagy több
Kijelző	1280x720 felbontás	1920x1080 (Full HD) felbontás

2.1.2. Az alkalmazás telepítése

A szoftver Python alapú, így futtatásához a Python környezet és a szükséges könyvtárak telepítése elengedhetetlen.

1. Python környezet telepítése: Töltse le és telepítse a Python3 legfrissebb stabil verzióját a hivatalos `python.org` weboldalról. A telepítés során ügyeljen arra, hogy a "Add Python to PATH" opciót jelölje be.

2. Függőségek telepítése: Az alkalmazás grafikus megjelenítéséhez és hálózatkezeléséhez külső könyvtárak szükségesek. Nyissa meg a parancssort (Command Prompt / Terminal), és futtassa az alábbi parancsot:

```
pip install pygame networkx PyQt6
```

A parancs automatikusan letölti és telepíti a `pygame` (megjelenítés), `networkx` (gráfkezelés) és `PyQt6` (felhasználói interfész) modulokat. A telepítés sikeressége a verziószámok megjelenésével ellenőrizhető.

2.2. Működési környezet és jogosultságok

Az alkalmazás architektúrája egyfelhasználós (single-user), lokális működésre lett optimalizálva. A rendszer nem igényel központi szerverkapcsolatot vagy felhasználói azonosítást (autentikációt).

Ez a kialakítás a következő előnyökkel jár a felhasználó számára:

- **Azonnali hozzáférés:** Nincs beléptetési procedúra vagy szerepkör alapú korlátozás; a szoftver indítása után minden tervezési és szimulációs funkció azonnal, teljeskörűen elérhető.
- **Egyszerűsített fájlkezelés:** A rendszer nem alkalmaz fájlzárolást vagy konkurrens hozzáférés-kezelést, így a projektfájlok kezelése (másolás, mozgatás) az operációs rendszer szintjén szabadon elvégezhető.

A szoftver struktúrája a gyors prototípuskészítést és az önálló mérnöki munkavégzést támogatja, felesleges adminisztrációs terhek nélkül.

2.3. Hardverkövetelmények és ergonómia

A szoftver tervezésekor elsődleges szempont volt a vasúti menetrend-tervezés és a hálózatszerkesztés komplexitása. A diszpécseri és mérnöki munkaállomások ergonómiájához igazodva a támogatott környezet a következő:

- **Platform:** Asztali számítógép vagy laptop.
- **Beviteli eszközök:** A precíz pályaszerkesztés (pl. vágánygeometria, jelzők elhelyezése) miatt az egér és billentyűzet kombinációjának használata erősen ajánlott. Érintőképernyős vezérlés nem támogatott.

A fentiek miatt a mobil eszközök (okostelefonok, táblagépek) támogatása nem került implementálásra, mivel azok kijelzőmérete és beviteli módszerei nem teszik lehetővé a szoftver funkcióinak kompromisszummentes használatát.

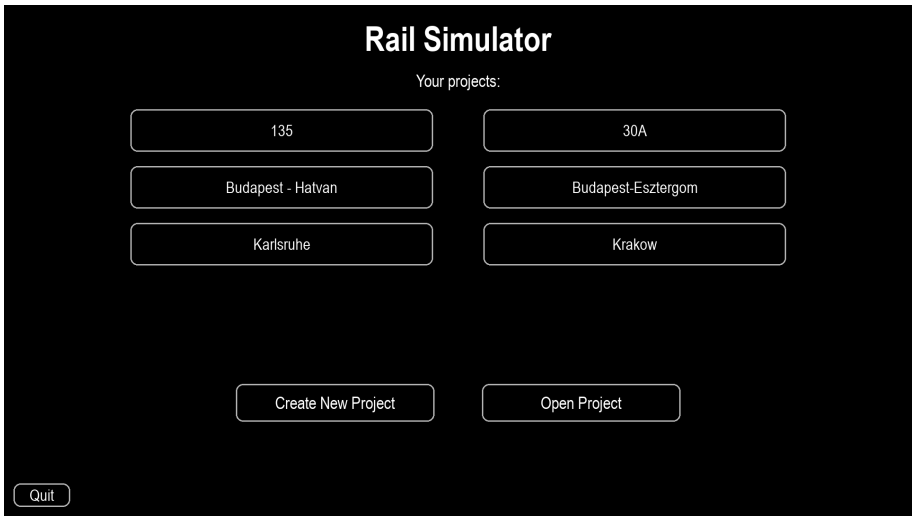
2.4. Az alkalmazás funkcióinak részletes ismertetése

Az alábbiakban részletesen ismertetem az alkalmazás menürendszerét, a pályaszerkesztés lépéseit és a szimulációs nézet funkcióit.

2.4.1. Kezdőoldal

Az alkalmazás indításakor a Kezdőoldal fogadja a felhasználót. A felület felső szekciója automatikusan listázza az alapértelmezett munkakönyvtárban detektált, korábban mentett szimulációkat.

A képernyő alsó sávjában elhelyezkedő vezérlőgombokkal új projekt hozható létre, illetve lehetőség nyílik a külső könyvtárban tárolt állományok tallózására és megnyitására. A programból való kilépés a bal alsó sarokban található 'Quit' gombbal kezdeményezhető.



2.1. ábra. A program kezdőoldala a projektválasztóval

2.4.2. Projekt nézet és navigáció

Egy szimulációs állomány betöltése után a központi Projekt nézet válik aktívvá. Innen érhető el az összes szerkesztési és szimulációs funkció.

Interakciók és hibakezelés:

A projekt szerkesztése közben a felhasználói interakciók során a következő visszajelzési mechanizmusok segítik a felhasználót:

- *Szövegbevitel:* Amennyiben egy művelet szöveg megadását igényli (pl. név, paraméter), azt a felhasználó a felugró ablakban (popup) teheti meg.

2.2. ábra. Példa szöveges adatbevitelre

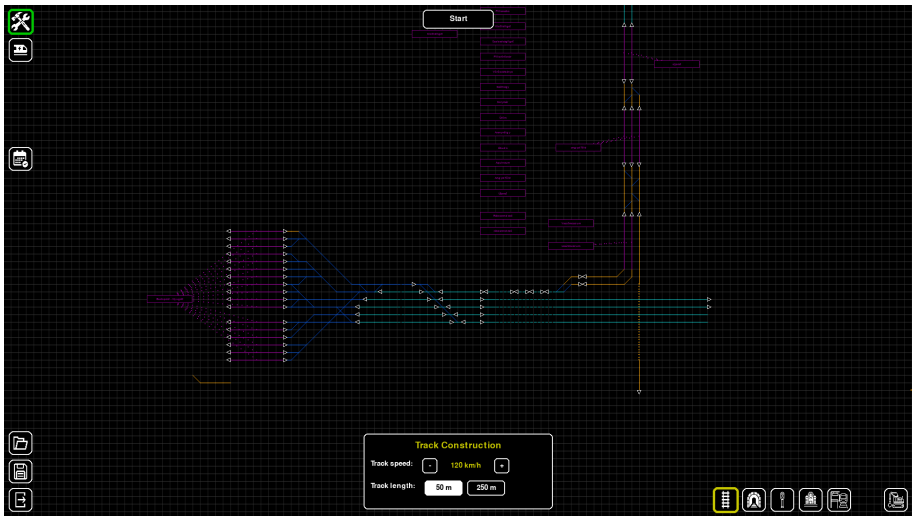
- *Vizuális visszajelzés:* Ha egy elem az adott pozícióban nem helyezhető el (pl. ütközés vagy érvénytelen geometria), a kurzor alatt a terület piros színre vált.
- *Hibaüzenetek:* Amennyiben a felhasználó érvénytelen helyre kattint, vagy szabálytalan műveletet kísérel meg, a program **felugró értesítésben (popup)** jelzi a hiba tényét és részletezi annak okát.

2.3. ábra. Figyelmeztető hibaüzenet

Megjelenítés és geometria: A munkaterület egy izometrikus rácshálóra illeszkedik. *Megjegyzés:* Bár a perspektivikus torzítás miatt a rács átlója vizuálisan hosszabbnak tűnik az oldalaknál, a szimulációs logika szerint a rácsátlók és a rácsélek hossza megegyezik.

Menürendszer: A nézet felső sarkában található menüsor biztosítja a váltást a különböző szerkesztési üzemmódok között. Az aktív funkciót a gomb körüli vastag keret jelzi. A különböző nézetek gyorsbillentyűkkel is elérhetők:

- **C** - Pályaszerkesztés (Construction)
- **T** - Vonatok elhelyezése (Train placement)
- **R** - Menetrendek kezelése (Timetables)



2.4. ábra. A projekt nézet és a felső menüsor

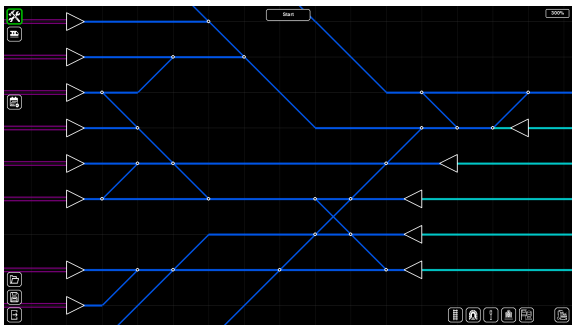
A menetrendek adminisztrációja - funkcionális elkülönülése miatt - egy dedikált ablakban történik (lásd: 2.4.3 fejezet).

Navigáció a munkaterületen

A projekt területe szabadon bejárható:

- **Mozgatás:** Bármely egérgomb lenyomva tartása mellett az egér húzásával (pan).
- **Nagyítás/Kicsinyítés (Zoom):** Az egérgörgő használatával. A nagyítás középpontja mindig az egérmutató aktuális pozíciója.

Amennyiben a nézetpozíció vagy a nagyítás mértéke eltér az alapértelmezettől, a jobb felső sarokban megjelenik egy navigációs információs ablak. Erre kattintva a nézet azonnal visszaállítható a pálya geometriai középpontjára és a minimális nagyítási szintre. Projekt megnyitásakor a program alapértelmezetten ezt a nézetet tölti be.

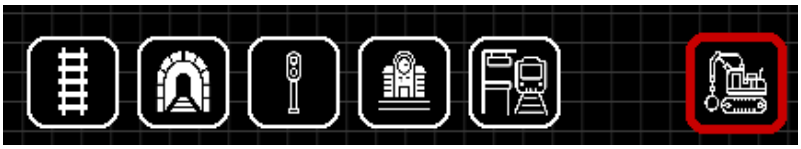


2.5. ábra. Navigációs információs ablak nagyításkor

Infrastruktúra építése (Pályaszerkesztés)

Ebben a módban végezhető el a vasúti hálózat topológiájának kialakítása. A képernyő jobb alsó sarkában található eszköztáron hat különböző elemtípus választható ki. Gyorsbillentyűk az elemek kiválasztásához:

- **1** - Vágány
- **2** - Alagút
- **3** - Jelzőberendezés
- **4** - Állomásépület
- **5** - Peron
- **0** - Törlés



2.6. ábra. Az építési mód eszköztára

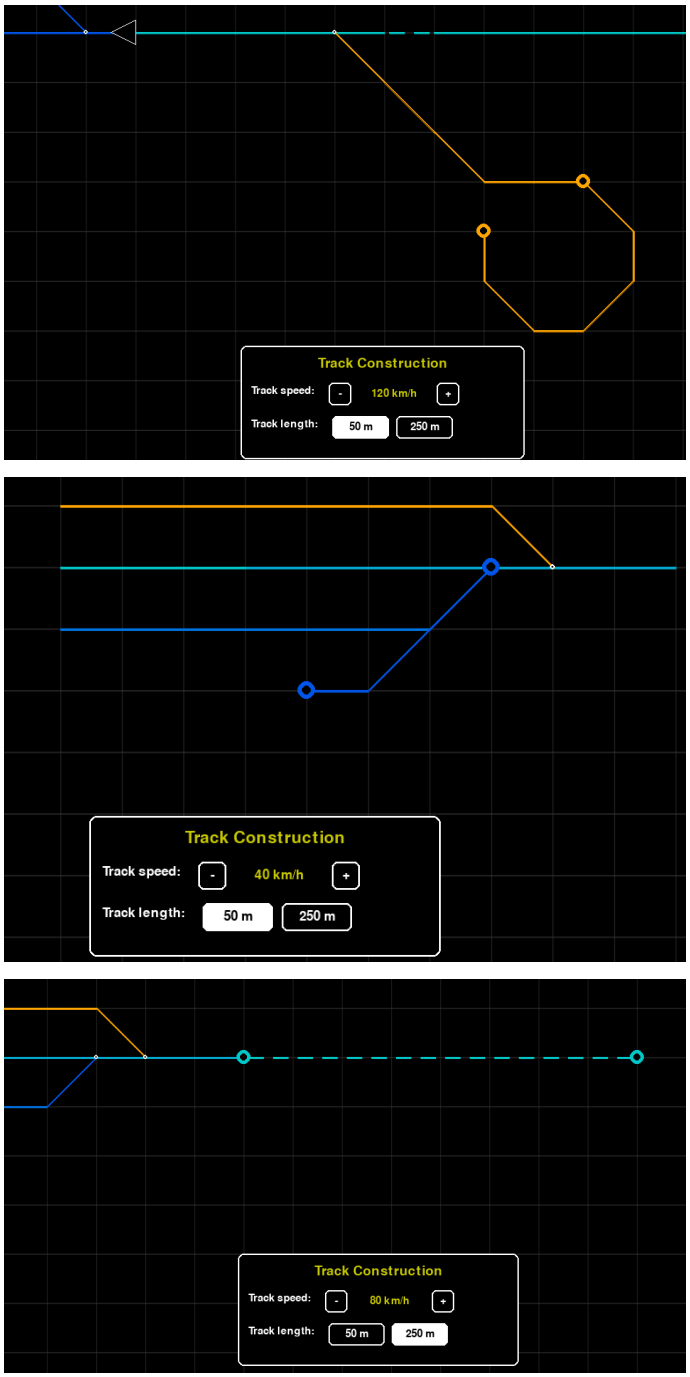
Vágány fektetése

A vágányhálózat a projekt alapja. Az építés menete:

1. Kattintással jelölje ki a szakasz kezdőpontját.
2. Húzza az egeret a kívánt végpont felé. A program automatikusan generálja a legrövidebb, fizikailag kivitelezhető nyomvonalat.
3. Újabb kattintással véglegesítse a szakaszt.

Láncolt építés: A véglegesítés után az eszköz aktív marad, és az előző szakasz végpontja válik az új szakasz kezdőpontjává, lehetővé téve a folyamatos építést. A rendszer automatikusan tiltja a járművek számára járhatatlanul szűk ívek létrehozását. A művelet megszakítása vagy az előnézet törlése a **jobb egérgombbal** lehetséges.

Paraméterek: Az alsó panelen definiálható a vágány sebességhatára (színkódolással jelölve) és megjelenítési típusa (folyamatos vagy szaggatott vonal).



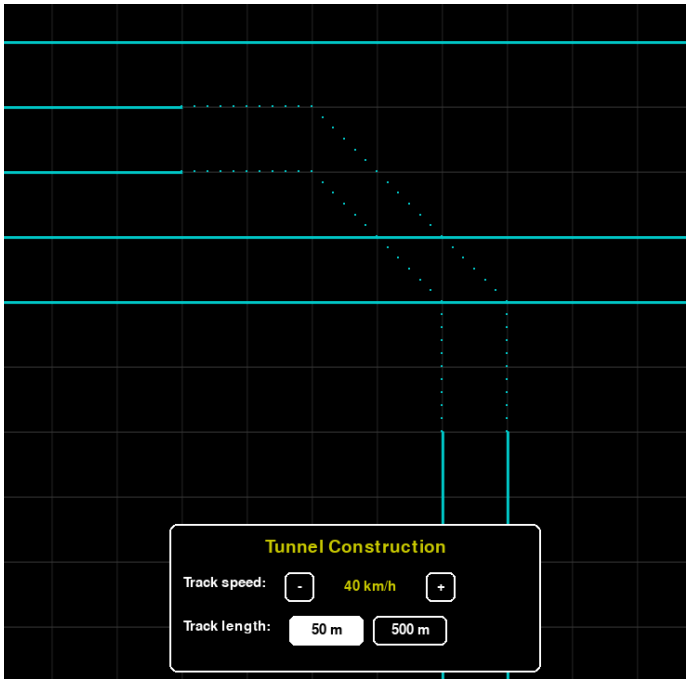
2.7. ábra. Vágányépítés folyamata és a különböző sebességek és hosszak megjelenítése

Alagút

Az alagút funkció a vágányok különszintű keresztezését teszi lehetővé. Ez nem földrajzi alagút-szimuláció, hanem logikai eszköz a vágányok különszintű átvezetésére.

Telepítési feltételek:

- Kizárólag meglévő vágányvégek közé illeszthető.
- Alagutak keresztezése nem engedélyezett.



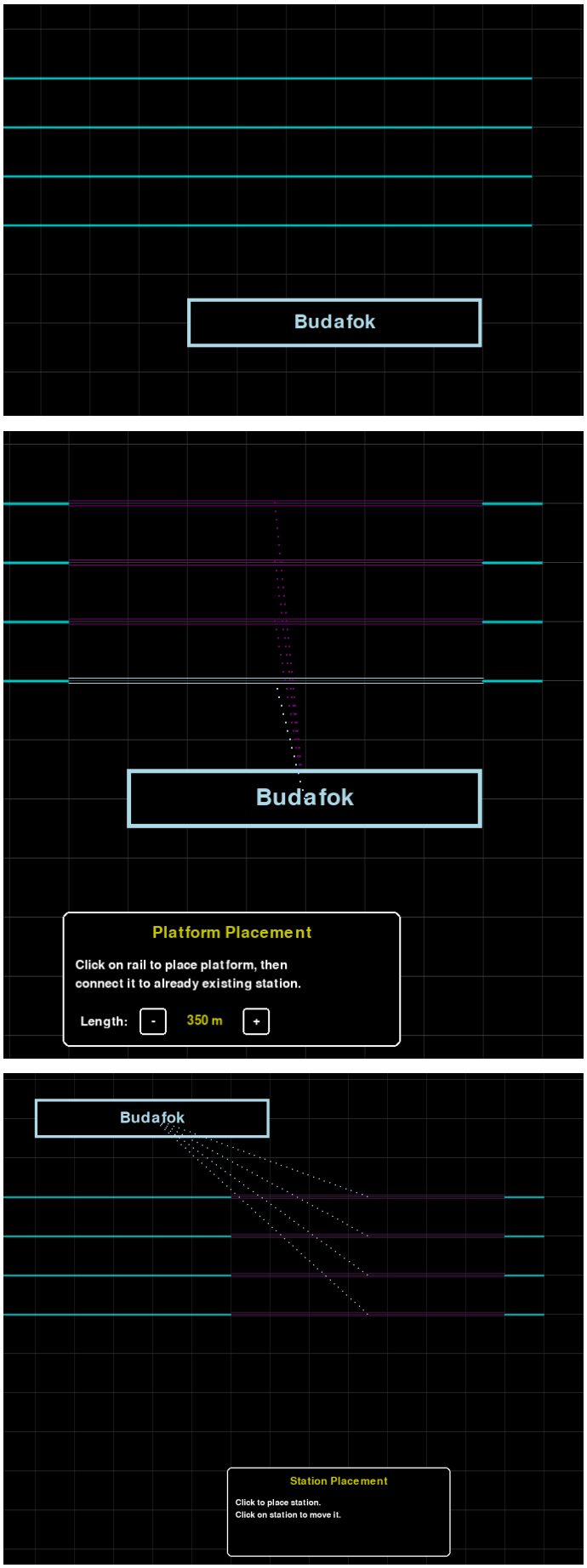
2.8. ábra. Alagút elhelyezése két vágányvég között

Állomások és Peronok

Az utasforgalmi létesítmények definiálása hierarchikus:

1. **Állomásépület:** A vágányok melletti szabad területre helyezendő.
2. **Peron:** Közvetlenül a vágányra illeszkedik, és logikailag a legközelebbi állomásépülethez kapcsolódik (ezt pontozott vonal jelzi).

Peron nem létesíthető váltókörzetben (kereszteződés) vagy íves pályaszakaszon. Az állomásépület pozíciója utólagosan módosítható.

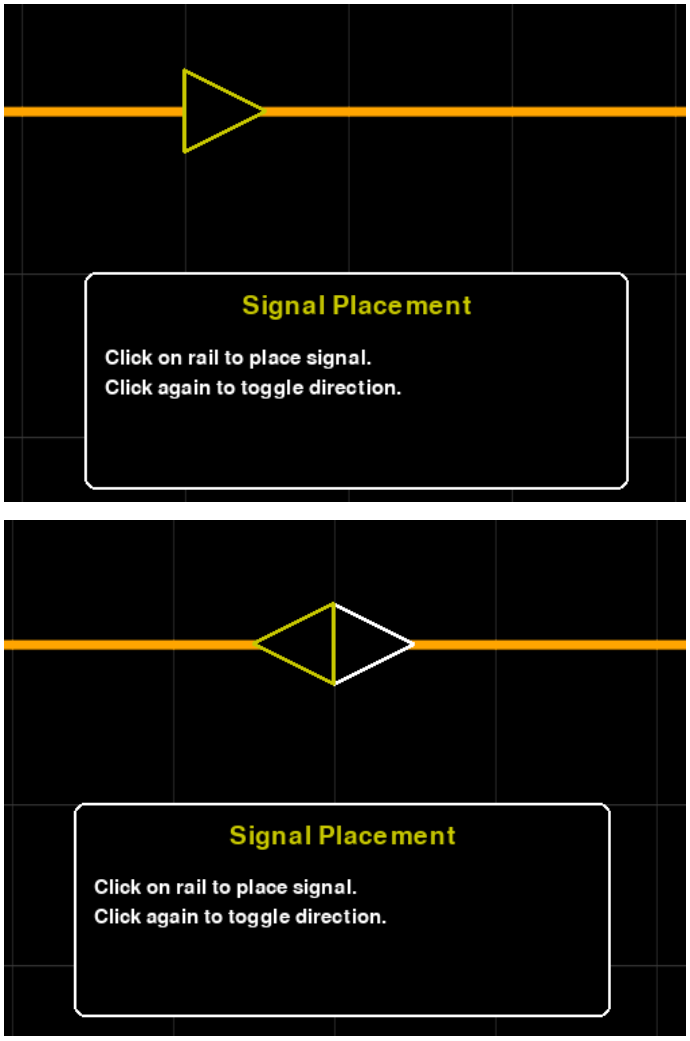


2.9. ábra. Állomások és peronok elhelyezése, illetve mozgatása

Jelzőberendezések

A forgalomszabályozó jelzők a vágányok mentén helyezkednek el, irányultságuk a vágánytengelyhez kötött (kattintással megfordítható).

Tiltott zónák: A jobb átláthatóság érdekében nem telepíthető jelző váltókra, ívekre, peronokra vagy alagutakba.



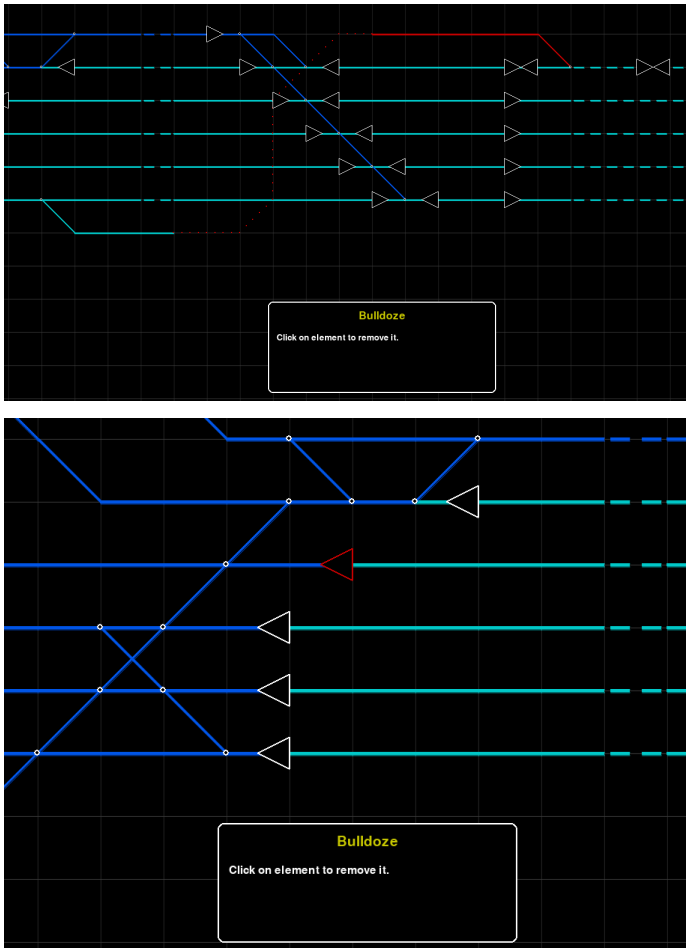
2.10. ábra. Jelző elhelyezése és irányának megfordítása

Elemek törlése

A törlés funkcióval az infrastruktúra elemei eltávolíthatók. A kurzor alatt a törlendő objektum piros kiemelést kap.

- **Vágány:** A rendszer igyekszik a logikailag egybe tartozó pályaszakaszokat egyben kijelölni.

- **Állomás:** Az épület törlése a hozzárendelt peronok automatikus eltávolítását is maga után vonja.

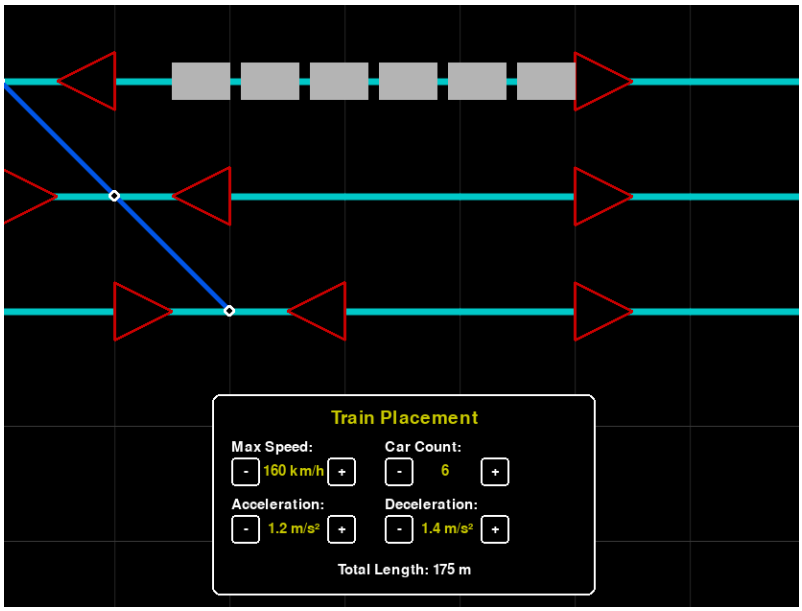


2.11. ábra. Vágány és jelző törlésének kijelölése

2.4.3. Vonatok elhelyezése

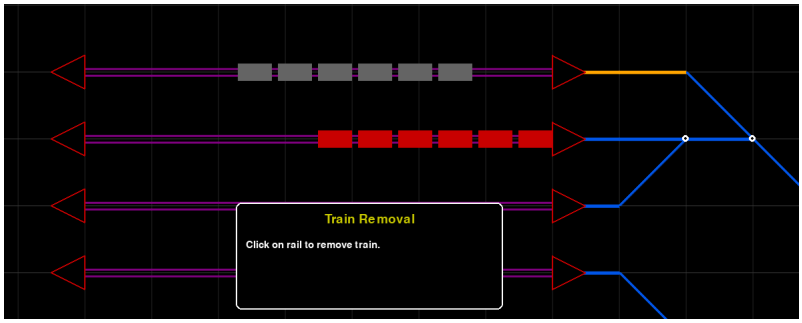
Gördülőállomány elhelyezése

A szimuláció kezdőállapotának beállításához a vonatokat a pályára kell helyezni. A *Vonat hozzáadása* eszközzel a vágányra kattintva jön létre az új szerelvény. A vonat fizikai paraméterei (kocsik száma, maximális sebesség, gyorsulás/lassulás) az alsó panelen konfigurálhatók.



2.12. ábra. Új szerelvény elhelyezése a pályán

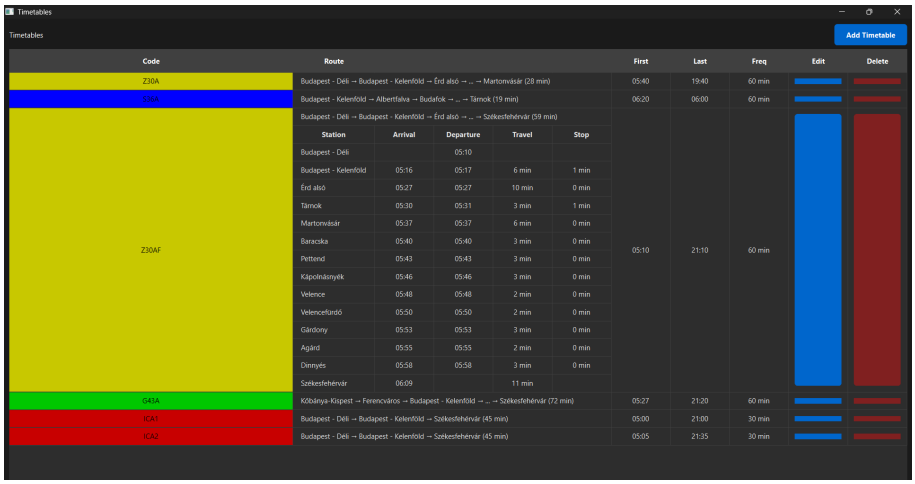
A *Vonat törlése* eszközzel a pályán elhelyezett járművek távolíthatók el.



2.13. ábra. Vonat eltávolítása

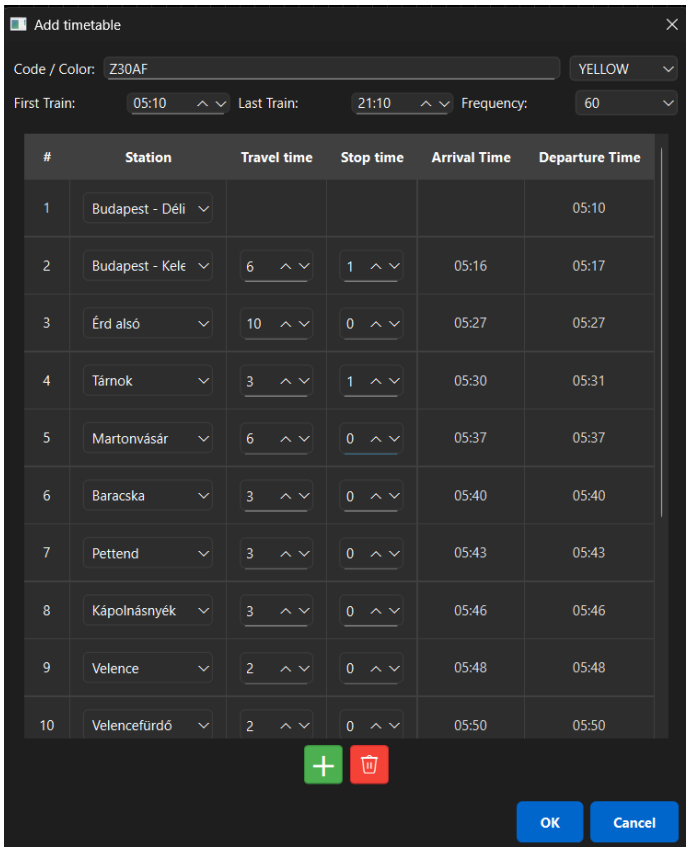
Menetrendi tervezés

A menetrendek definiálják a járművek útvonalát és időzítését. A funkció egy külön ablakot használ a jobb áttekinthetőség érdekében. A menetrendek először listanézetben jelennek meg, az egyes viszonylatok kattintással lenyithatók. A lista elemeit kibontva láthatóvá válnak a részletes adatok: állomások sorrendje, érkezési-, indulási-, várakozási- és menetidők.



2.14. ábra. Menetrendek listanézete

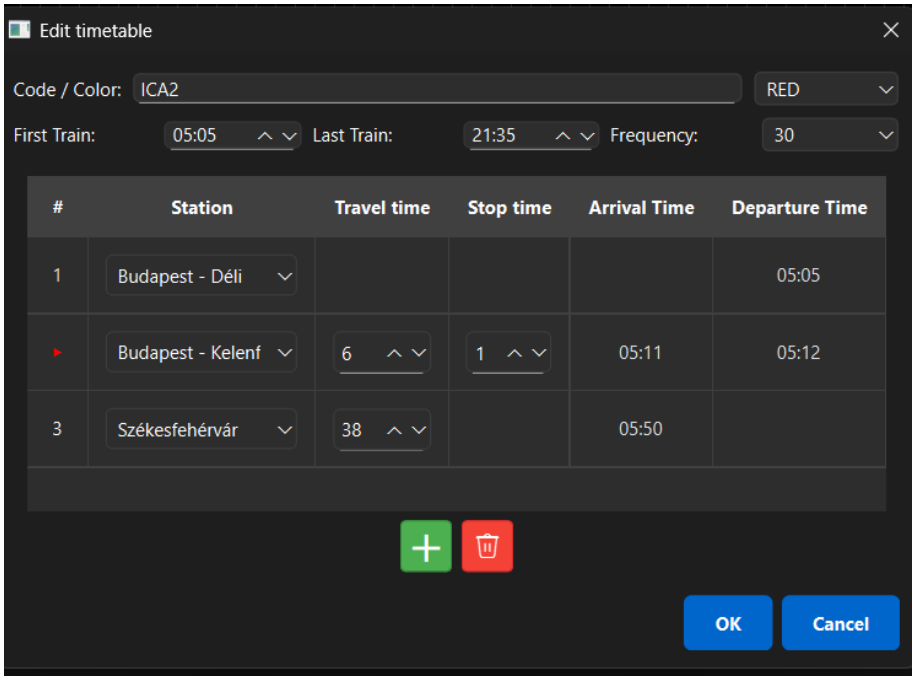
Szerkesztési műveletek: Az 'Add Timetable' gomb új viszonylatot hoz létre. A felugró ablakban a viszonylat neve, színe, gyakorisága és a megállók időzítése állítható be.



2.15. ábra. Új menetrend létrehozása

A menetrend szerkesztő felületén a következő műveletek végezhetők el:

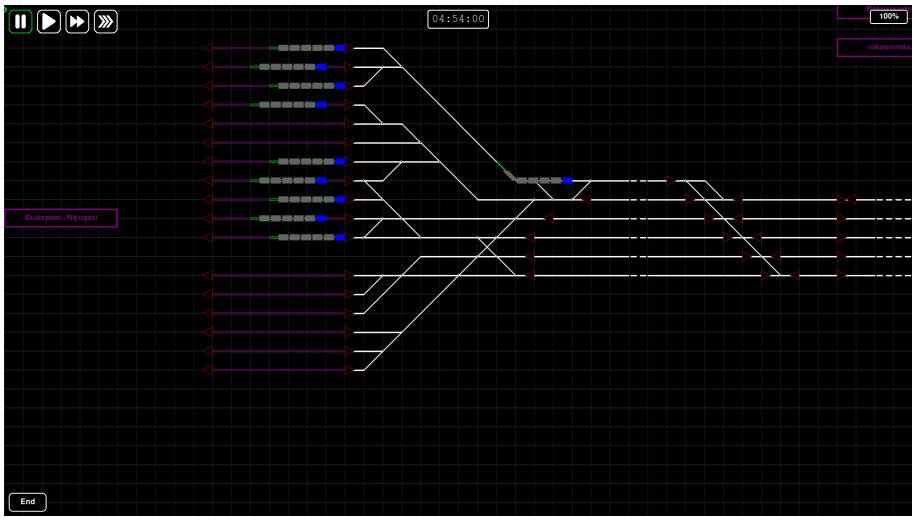
- **Állomás beszúrása:** A '+' gombbal. (Kijelölés hiányában a lista végére kerül).
- **Időzítés számítása:** A felhasználó által megadott várakozási és utazási idők alapján a szoftver automatikusan kalkulálja az abszolút indulási és érkezési időpontokat.
- **Törlés:** A 'kuka' ikonnal.



2.16. ábra. Állomás kiválasztása a menetrend szerkesztőben

2.4.4. Szimulációs üzemmód

A projekt futtatása a 'Start Simulation' gombbal és a kezdő időpont megadásával indítható.



2.17. ábra. Aktív szimulációs nézet

Idővezérlés

A szimuláció sebessége a vezérlőpulton vagy billentyűzettel szabályozható:

- **SPACE** - Szünet / Folytatás
- **0** - Szünet
- **1** - Normál sebesség (1x)
- **2** - Gyorsított (5x)
- **3** - Maximális sebesség (25x)

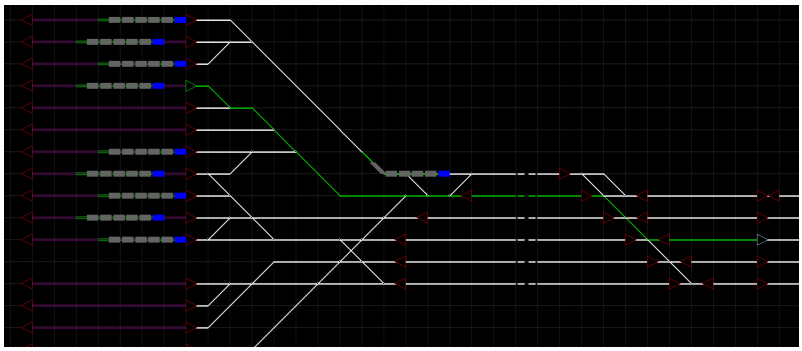
Forgalomirányítás (Vágányutak kezelése)

A vonatok biztonságos közlekedése vágányutak beállításával történik.

Vágányút kijelölése:

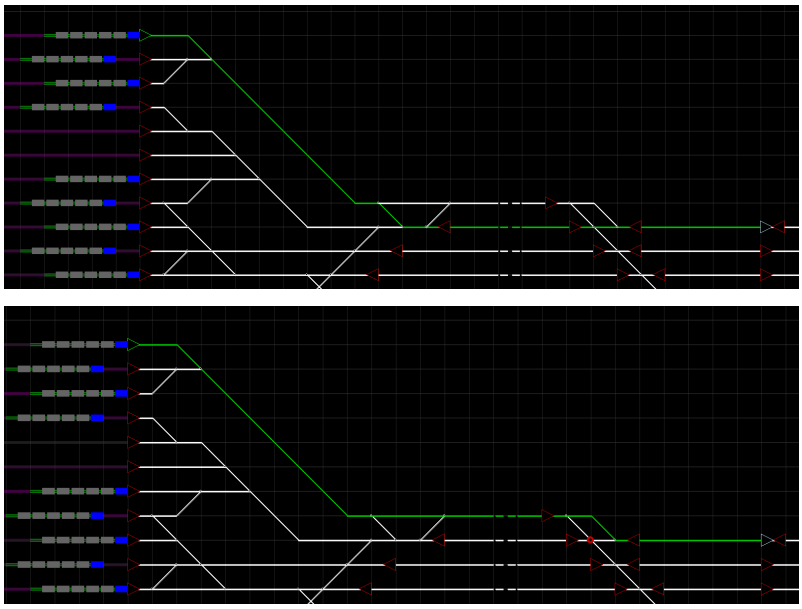
1. Kattintson a kezdő jelzőre (Start).
2. Vigye a kurzort a céljelző (Cél) fölé az útvonal-előnézet megjelenítéséhez.
3. Kattintson a céljelzőre a beállításhoz.

Sikeres beállítás esetén a start és a köztes jelzők 'Szabad' állásba kerülnek. Visszavonáshoz kattintson jobb gombbal a jelzőre.



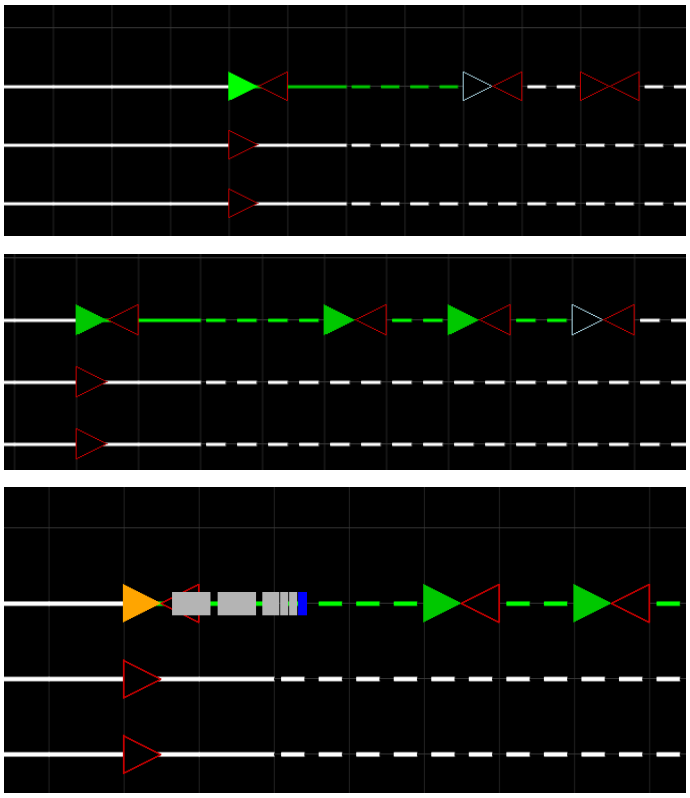
2.18. ábra. Vágányút előnézete (zöld vonal)

Útvonal-kényszerítés (Blokkolás): Alternatív útvonalak esetén a felhasználó kizárhat csomópontokat a keresésből. A Start jelző kijelölése után kattintson a térképen a kerülendő pontra (piros kör jelzi), majd jelölje ki a Cél jelzőt.



2.19. ábra. Útvonal tervezése tiltott pont (blocker) alkalmazásával. Fent: alapútvonal, lent: blokkolt pont után.

Automata térközbiztosítás: Hosszú, elágazásmentes szakaszokon lehetőség van láncolt kijelölésre. A céljelző kijelölésekor tartsa nyomva a **SHIFT** billentyűt. Ekkor a rendszer az összes köztes jelzőt automata térközjelzővé alakítja (töltött háromszög szimbólum), melyek a vonat áthaladása után automatikusan kezelik a foglaltságot.

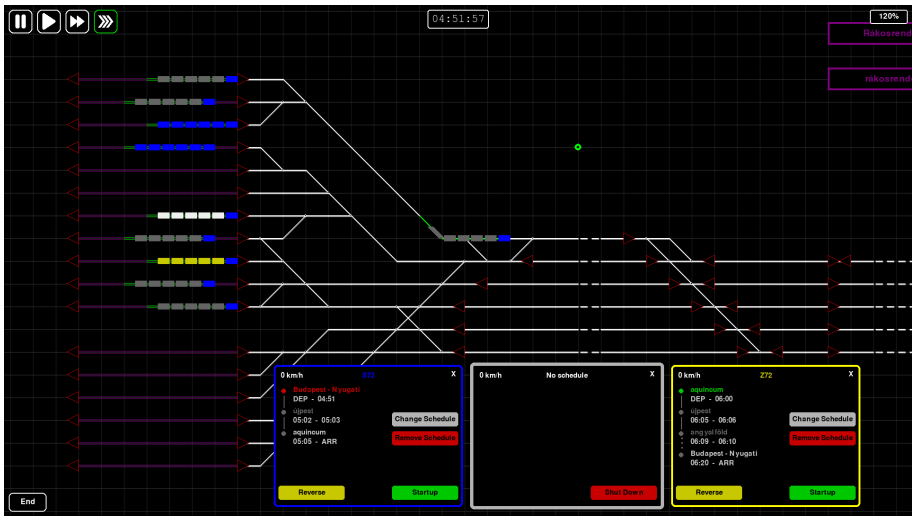


2.20. ábra. Automata térközjelzők működése és foglaltságjelzése

Járműfelügyelet

A vonatokra kattintva megjelenik a vezérlőpanel (max. 4 panel egyidejűleg).

- **Operatív beavatkozás:** Menetrend hozzárendelése, kézi indítás/megállítás, irányváltás.
- **Státuszmonitor:** Sebesség, menetrendi pontosság (késés kijelzése) és következő célállomás nyomon követése.



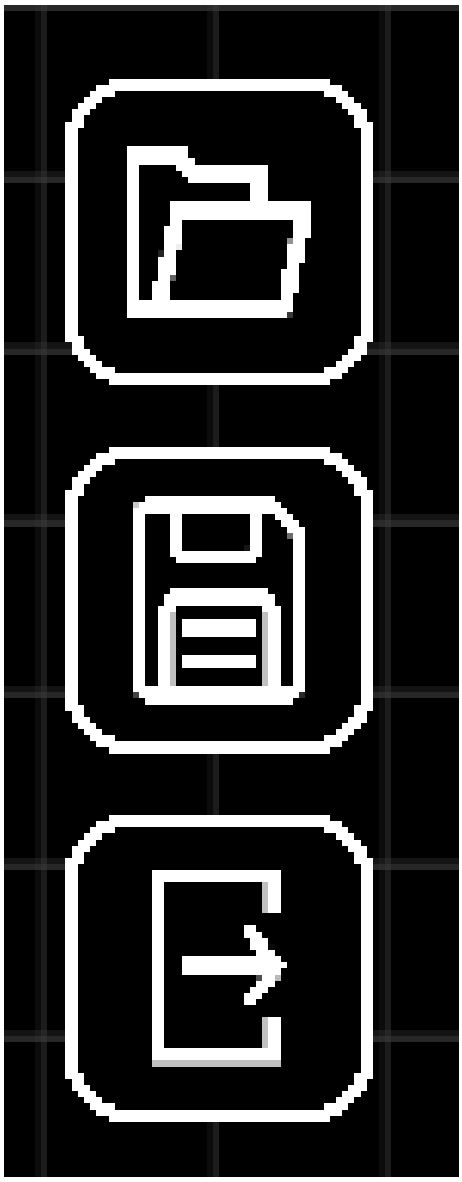
2.21. ábra. Vonatok információs panelei (késés és pontos haladás jelzése)

Szimuláció leállítása

A 'Stop Simulation' gomb megnyomásával a rendszer visszaáll szerkesztő módba, és a szimulációs állapot törlődik.

2.4.5. Projektkezelés (Mentés és betöltés)

A projektfájlok kezelése az alkalmazás kulcsfontosságú funkciója, amely biztosítja a munkafolyamatok folytonosságát. A projektek tárolása szabványos `.json` formátumban történik, amely biztosítja az adatok hordozhatóságát. A fájlkezelő műveletek a képernyő bal alsó sarkában található funkciógombokkal, illetve gyorsbillentyűkkel érhetők el.



2.22. ábra. Mentés, Betöltés és Kilépés gombok

Mentés (Save)

A mentés állapota vizuálisan követhető a mentés ikon változásain keresztül:

- **Szinkronban:** A "floppy" ikon alapállapotban jelzi, hogy a projekt mentve van.
- **Módosítva:** Ha a projektben változtatás történt az utolsó mentés óta, az ikon mellett egy 'x' jelzés, vagy módosult állapotjelző tűnik fel.

2.23. ábra. Módosított állapotjelző

- **Visszajelzés:** Sikeres mentés esetén a program rövid ideig egy pipa megjelenítésével nyugtázza a műveletet.

Mentési opciók:

- **Gyorsmentés:** A **CTRL + S** kombinációval a módosítások azonnal felülírják az aktuális fájlt.
- **Mentés másként:** A mentés gombra kattintva felugró ablakban lehetőség van a projekt új néven történő elmentésére.

Biztonsági funkció: Amennyiben a felhasználó mentés nélkül próbál kilépni vagy új projektet nyitni, a rendszer figyelmeztető ablakban ajánlja fel a változtatások rögzítését.

Betöltés (Load)

Korábbi munkamenetek visszaállítása a **CTRL + O** billentyűkombinációval, vagy a mappát ábrázoló 'Betöltés' ikonnal kezdeményezhető. A megjelenő fájlkezelő ablakban tallózható be a kívánt `.json` állomány. A rendszer beépített validációval rendelkezik: kizárólag érvényes struktúrájú projektfájlok betöltését engedélyezi. Sérült vagy inkompatibilis fájl esetén hibaüzenet tájékoztatja a felhasználót.

Visszalépés a főmenübe

A szerkesztési felületről való kilépéshez és a Kezdőoldalra való visszatéréshez a bal alsó sarokban található 'Kilépés' (Exit) gomb szolgál. Ezzel a művelettel a jelenlegi projekt bezárul (mentési figyelmeztetéssel), és a felhasználó visszakerül a projektválasztó képernyőre.

3. fejezet

Fejlesztői dokumentáció

3.1. Technológiai döntések és alternatívák elemzése

Az alábbiakban bemutatom a mérlegelt alternatívákat és a választott megoldás melletti érveket.

3.1.1. Programozási nyelv: Python vs. C++ / C#

A hasonló 2d szimuláció területén a leggyakoribb alternatívák a C++ vagy a C# (Unity környezetben).

- **C++ / C# alternatívák:** Előnyük a magas futási teljesítmény és a szigorú típusosság, amely nagy számításigényű 3D grafikánál elengedhetetlen. Hátrányuk azonban a hosszabb fejlesztési idő (boilerplate kódok).
- **A választott Python előnyei:** Jelen projekt esetében - a 2D-s megjelenítés révén a kritikus tényező nem a nyers grafikus teljesítmény, hanem a *fejlesztési sebesség* és az *adatstruktúrák rugalmas kezelése* volt. A Python dinamikus típusossága és tömör szintaxisa lehetővé tette a komplex logika (vonatok mozgása, jelzők állapota) gyors implementálását.

3.1.2. Szimulációs környezet: Pygame vs. Game Engines (Unity/Unreal)

A grafikus megjelenítés és a szimulációs ciklus (game loop) megvalósítására két fő irányvonal létezik: egy kész játékmotor (pl. Unity) használata vagy egy keretrendszer (framework) alkalmazása.

- **Játékmotorok (Unity, Godot):** Ezek az eszközök "dobozos" megoldásokat kínálnak fizikára és renderelésre. Bár látványos eredményt adnak, a projekt szempontjából túlzott komplexitást (overhead) jelentenek. Egy Unity projekt esetén a fejlesztő kénytelen a motor logikájához igazodni, ami megnehezíti a saját, egyedi gráf-alapú mozgási logika implementálását.
- **A választott Pygame előnyei:** A Pygame nem egy motor, hanem egy könyvtár, amely közvetlen hozzáférést ad a rajzolási felülethez (canvas) és az eseményhurokhoz. Ez a "code-first" megközelítés ideális a szakdolgozathoz, mivel:

1. Teljes kontrollt biztosít a szimulációs ciklus felett (nincs rejtett motorlogika).
2. A megjelenítés szorosan integrálható a Python adatmodelljeivel (nem kell adatokat átkonvertálni C# objektumokká).
3. Alacsony erőforrásigényű és könnyen hordozható.

Fontos megjegyezni, hogy a Python stack – és specifikusan a Pygame – legnagyobb hátránya a frontend teljesítménye. A Pygame szoftveres renderelése nem versenyképes a GPU-gyorsított motorokkal. A választás mégis erre esett, mivel ennél a szimulációnál nem a pixelpontos megjelenítés, a nagy felbontás vagy a magas FPS (képkockaszám) a cél, hanem a logikai komplexitás kezelése. A rendszer lényege a jelzők állítása, a dinamikus útvonalkeresés, valamint a vonatok és a pálya közötti kommunikáció szimulálása, így ezen funkciók mellett a nyers grafikus performance másodlagos tényező.

3.1.3. Gráfkezelés és algoritmusok: NetworkX

A vasúti hálózat és a pályaelemek modellezéséhez a **NetworkX** könyvtár mellett döntöttem. A szimuláció során a programnak rengetegszer kell különféle gráfalgoritmusokat meghívnia a pálya állapotának kiértékeléséhez. A NetworkX használata mellett szólt, hogy a könyvtárból elérhető rengeteg extra feature és optimalizált függvény (pl. `EdgeView`, `neighbors`, `degree`, stb.) jelentősen megkönnyíti a fejlesztést, kiváltva a saját, potenciálisan lassabb gráfkezelő algoritmusok írását.

3.1.4. A felhasználói felület kihívásai: A hibrid architektúra indoklása

A fejlesztés egyik kritikus pontja a menetrend-szerkesztő modul kialakítása volt. Itt ütközött ki a Pygame, mint grafikus könyvtár korlátja, és vált szükségessé egy hibrid megoldás bevezetése.

A probléma: Pygame korlátai UI téren

A Pygame egy úgynevezett *framebuffer-alapú* megjelenítő. Minden képkockát pixelről pixelre rajzol újra, és nem rendelkezik beépített, magas szintű UI komponensekkel.

- Egy menetrend táblázatos megjelenítése (sorok, oszlopok, görgetés, szerkeszthető cellák) Pygame-ben rendkívül erőforrás-igényes feladat.
- A menetrendek megjelenítése a térképpel, a jelzőkkel és vonatokkal ellentétben standard UI elemeket igényel, mint például táblázatok, legördülő menük és szövegbeviteli mezők.
- Ezekre a komponensekre nincsenek natív megoldások beépítve a Pygame-be.
- Ezen komponensek nulláról történő lefejlesztése aránytalanul sok időt vett volna el a vasúti szimuláció fejlesztésétől, és az eredmény ergonómiailag és esztétikailag is elmaradt volna a szabványos megoldásoktól.

A megoldás: PyQt6 integrációja

Ezen hiányosságok áthidalására választottam a **PyQt6** keretrendszert a menetrend-kezelő modulhoz. A PyQt6 (a Qt keretrendszer Python portja) ipari szabványos, professzionális nézetet biztosít a menetrendek számára.

- **Táblázatkezelés:** A `QTableWidget` komponens azonnal biztosítja a cellaszerkesztést, görgetést és legördülő menüket, amit Pygame-ben hetekig tartott volna implementálni.
- **Ergonómia:** A felhasználó a megszokott operációs rendszer szintű vezérlőkkel találkozik, ami növeli a szoftver használhatóságát.
- **Szétválasztás:** Ez a döntés tisztább architektúrát eredményezett. A *vizualizáció* (térkép, vonatok) a Pygame felelőssége, míg az *adminisztráció* (adatbevitel, konfiguráció) a PyQt-é. Ez a "megfelelő eszközt a megfelelő feladatra" elv gyakorlati alkalmazása.

Összegzésként elmondható, hogy a Python, Pygame, NetworkX és PyQt6 kombinációja egy olyan optimális technológiai elegyet (stack) alkot, amely egyesíti a gyors fejlesztést, a komplex algoritmusok kezelését, a teljes kontrollt a szimuláció felett, valamint a professzionális adatkezelő felületet.

3.2. Alkalmazott technológiák és fejlesztői környezet

A projekt implementációja **Python** programozási nyelven készült. A Python magas szintű absztrakciós képessége, valamint az elérhető könyvtárak és eszközök széles választéka lehetővé tette a gyors prototípus-fejlesztést és a komplex funkcionalitás hatékony megvalósítását.

3.2.1. Felhasznált könyvtárak

A fejlesztés során számos külső könyvtár került integrálásra, amelyek specifikus feladatokat látnak el a rendszer különböző rétegeiben:

Pygame: A szimulációs tér vizualizációjáért és a grafikus felhasználói felület megjelenítéséért felelős könyvtár. Ez a modul az alkalmazás grafikus felületének mintegy 80%-át adja. A Pygame multimédiás alkalmazások és játékok fejlesztésére tervezett keretrendszer, amely felelős a 2D grafikus elemek (sprite-ok) renderelésééért, a képfrissítési ciklus kezeléséért, valamint a felhasználói interakciók (billentyűzet- és egéresemények) valós idejű feldolgozásáért.

NetworkX: A vasúti hálózat topológiájának reprezentálására és kezelésére szolgáló könyvtár. A NetworkX komplex hálózatok és gráfok elemzésére specializálódott eszköz, amely gráf alapú adatszerkezetet (csomópontok és élek) biztosít. Alkalmazása jelentősen egyszerűsítette a hálózati topológia implementálását és a kapcsolati viszonyok kezelését.

PyQt6: Míg a szimuláció megjelenítése Pygame alapú, a menetrendek szerkesztése PyQt6 keretrendszer segítségével valósult meg. A Qt könyvtár Python implementációja professzionális, eseményvezérelt GUI komponenseket (gombok, táblázatok, legördülő menük) nyújt, amelyek ergonómikusabb felhasználói élményt biztosítanak az adminisztratív műveletek során.

Tkinter: A Python standard könyvtárának grafikus eszközkészlete. Az alkalmazásban kizárólag modális párbeszédablakok (pl. hibaüzenetek, figyelmeztetések, megerősítő dialógusok) megjelenítésére kerül felhasználásra (`messagebox` modul). Használatát az indokolta, hogy natív rendszerintegrációt biztosít további külső függőségek nélkül.

3.2.2. Adattárolás és perzisztencia

A projektek állapotának mentése és betöltése **JSON** (JavaScript Object Notation) formátumban történik. A JSON formátum választását több tényező motiválta: emberek által is könnyen olvasható, szöveges reprezentációja biztosítja a verziókezelés egyszerűségét, ugyanakkor strukturált adatmodellt nyújt a komplex objektumok szerializálásához.

A mentett JSON struktúra a következő főbb komponenseket tartalmazza:

graph: A vasúti pályahálózat topológiai reprezentációja NetworkX gráf formátumban, beleértve a vágányok kapcsolatait és térbeli pozícióit.

station_repository: Az állomások és az azokhoz tartozó peronok adatainak gyűjteménye.

signal_repository: A jelzők térbeli elhelyezkedése és irányítási tulajdonságai.

timetable_repository: A menetrendi útvonalak és időzítések definíciója.

train_repository: A járművek kezdeti pozíciója és paraméterezett tulajdonságai.

3.3. Szoftverarchitektúra és rendszerfelépítés

Az alkalmazás fejlesztése során moduláris, rétegelt architektúrát alakítottam ki, amely elősegíti a kód átláthatóságát és a későbbi bővíthetőséget. A projekt forráskódja a `src` könyvtárban található, amely a felelősségi körök (Separation of Concerns) alapján az alábbi fő rétegekre és komponensekre tagolódik.

3.3.1. Belépési pontok és vezérlés

Az alkalmazás indításáért és az életciklus-kezelésért felelős fájlok:

- `main.py`: A program belépési pontja. Feladata az alkalmazás indítása és a `MenuManager` inicializálása.
- `menu_manager.py`: Az alkalmazás központi állapotkezelője. Feladata az alkalmazás inicializálása, valamint a beérkező események (billentyűzet, egér) továbbítása a megfelelő vezérlő felé.
- `home_page_screen.py`: A kezdőképernyő megjelenítéséért és interakcióiért felelős osztály. Lehetővé teszi új projektek létrehozását, meglévők betöltését, valamint a legutóbbi projektek gyors elérését.

3.3.2. Core (Mag) réteg

A `core` réteg képezi az alkalmazás gerincét; ez a modul kapszulázza az üzleti logikát, a domain modelleket, a konfigurációs állományokat és az alapvető infrastrukturális szolgáltatásokat. A réteg feladata biztosítani a rendszer belső konzisztenciáját, függetlenül a felhasználói felület (UI) aktuális megvalósításától.

Config modul

A konfigurációs modul felelős az alkalmazás globális paramétereinek, állandóinak és erőforrás-hivatkozásainak központosított tárolásáért.

`color.py`: A színpaletta definícióit tartalmazza, biztosítva a konzisztens vizuális megjelenést az egész alkalmazásban.

`keyboard_shortcuts.py`: A beviteli vezérlés absztrakciója; itt kerülnek definiálásra a módváltásokhoz és funkciókhoz rendelt gyorsbillentyűk.

`paths.py`: Az erőforrás-kezelés segédosztálya, amely dinamikusan kezeli az ikonok és adatfájlok relatív és abszolút elérési útvonalait.

`config.py`: Az alkalmazás központi konfigurációs osztálya. Tartalmazza a szimulációs és megjelenítési paramétereket (pl. rácsméret, sebességhatárok, fizikai állandók). A fájl tartalma a 3.1 kódrészleten látható.

```python
class Config:
    MAPS_FOLDER = "maps"
    GRID_SIZE = 40           # Pixel
    BUTTON_SIZE = 50
    STATION_RECT_WIDTH = 6
    STATION_RECT_HEIGHT = 1
    FPS = 20

    MIN_TRACK_SPEED = 10
    MAX_TRACK_SPEED = 200
    TRACK_SPEED_INCREMENT = 10

    SHORT_SECTION_LENGTH = 50
    LONG_SECTION_LENGTH = 250

    TRAIN_CAR_LENGTH = 25
    TRAIN_CAR_GAP = 5
    MAX_TRAIN_CAR_COUNT = 16
    MIN_TRAIN_STOP_TIME = 30  # mp
    TRAIN_SAFETY_BUFFER = 0   # m
```

3.1. forráskód. A Config osztály részlete, amely a szimuláció változtatható paramétereit definiálja

Graphics modul

- `icon_loader.py`: Az erőforrások (sprite-ok, textúrák) betöltéséért és gyorsítótárazásáért felelős.
- `camera.py`: A felhasználói nézet transzformációit (eltolás, nagyítás/kicsinyítés) végző osztály. Lehetővé teszi a virtuális tér és a képernyő koordinátarendszere közötti konverziót.

- `graphics_context.py`: Egy *Context Object*, amely összefogja a rajzoláshoz szükséges függőségeket (pl. a Pygame felületét és a kamera objektumot), és továbbítja azokat a kirajzolást végző entitásoknak.

Models csomag

A `models` csomag tartalmazza a rendszer összes domain entitását, logikailag elkülönített alcsomagokba szervezve.

Geometry (Geometria és Térinformatika) A térbeli reprezentációért és a gráf-alapú koordinátákért felelős osztályok.

- `node.py`: A diszkrét rácspontokat reprezentáló osztály. A gráf csomópontjaként funkcionál, tárolja a koordinátákat és a magassági szintet (pl. alagút kezeléséhez).
- `position.py`: A folytonos (Euclideszi) tér egy pontját reprezentálja lebegőpontos koordinátákkal. Míg a `Node` a gráf csomópontjait jelöli, a `Position` a képpontok és a kurzor finommozgását írja le.
- `edge.py`: A gráf éleinek absztrakciója, amely két `Node` közötti kapcsolatot definiál.
- `direction.py`: A vektorirányok kezelését végző segédosztály. Kulcsszerepe van a gráfbejárás során a lehetséges haladási irányok meghatározásában.
- `pose.py`: A *Position* és *Direction* kombinációja (pozíció és orientáció). Nélkülözhetetlen a jelzők és vonatok állapotának leírásához, ahol a térbeli elhelyezkedés mellett az irány is meghatározó.

```
1    def get_connecting_poses(self, other_level: bool = False) ->
        list['Pose']:
2        neighbors = []
3        for dir in self.direction.get_valid_turns():
4            nx = self.node.x + dir.x
5            ny = self.node.y + dir.y
6            new_state = Pose(Node(nx, ny, self.node.level), dir)
7
8            neighbors.append(new_state)
9            if other_level:
```

```
10                 neighbors.append(new_state.toggle_level())
11         return neighbors
```

3.2. forráskód. A Pose osztály `get_connecting_poses` metódusa, amely azokat a pozíciókat adja vissza, amelyek a vonatok által bevehető pályageometriájú szakaszokat reprezentálják

Railway (Vasúti Logika) A különböző vasúti entitások és algoritmusait összekötő réteg.

- `railway_system.py`: Egy **Facade (Homlokzat)** tervezési mintát megvalósító osztály. Ez az egyetlen belépési pont a külvilág számára a vasúti alrendszer felé; összefogja és koordinálja az állomások, vonatok, jelzők és a biztosítóberendezés működését.
- `graph_adapter.py`: Egy **Adapter** mintát követő osztály, amely elszigeteli a *NetworkX* könyvtárat a rendszer többi részétől. Ez biztosítja, hogy a gráfimplementáció cseréje esetén csak ezt az osztályt kelljen módosítani.
- `graph_service.py`: Magas szintű gráfműveleteket és a gráfon végzett keresési algoritmusokat (pl. szekciókeresés, peron-validáció) biztosító szolgáltatás.
- `signalling_service.py`: A biztosítóberendezés logikáját implementálja. Feladata a jelzők aspektusának (szabad/tilos) dinamikus frissítése és a vágányutak foglaltságának ellenőrzése.
- `path_finder.py`: A nem gráfon végzett útvonalkeresést megvalósító osztály. A `A*` algoritmust alkalmazza két pont közötti legrövidebb vágány építésének meghatározására.

Repositories (Adattárolók) A **Repository** tervezési minta alkalmazása az entitások életciklusának kezelésére. Ezek az osztályok felelősek az objektumok (vonatok, állomások, menetrendek) memóriában történő tárolásáért, lekérdezéséért és módosításáért.

- `station_repository.py`, `signal_repository.py`, `train_repository.py`, `timetable_repository.py`

Domain Entities (Entitások) A rendszer alapvető építőkövei:

- `rail.py`: A fizikai vágány modellje, amely kiterjeszti az `Edge` osztályt olyan tulajdonságokkal, mint a sebességkorlát és a pályahossz.
- `signal.py`: A vasúti jelzőberendezés modellje. Tárolja a pozíciót (`Pose`), a következő jelző referenciáját és a fedezett vágányutat.
- `train.py`: A vonat modellje. Az osztály kezeli a jármű fizikai paramétereit (sebesség, gyorsulás, fékezés), követi a menetrendet és interakcióba lép a pályával.
- `timetable.py` és `schedule.py`: A `Timetable` a statikus útvonaltervet (megállók sorrendje, és időkülönbsége), míg a `Schedule` egy konkrét, időponthoz kötött járatot reprezentál, amely alapján a szimuláció elindítja a vonatot.
- `app_state.py`: Az alkalmazás globális állapotát (pl. Szerkesztő mód vs. Szimulációs mód) tároló modell.

3.3.3. Shared (Megosztott) réteg

A `shared` réteg alapvető célja a kód újrafelhasználhatóságának biztosítása a különböző modulok között. Ez a réteg tartalmazza az alapvető UI osztályokat, a vezérlő logikákat (controllers), valamint a rajzolást és egyéb segédfunkciókat megvalósító könyvtárakat.

UI Modellek és Alaposztályok

Az `ui/models` könyvtár definiálja az összes felhasználói felület (UI) komponenst és azok absztrakt ősosztályait.

`ui_component.py` Az összes UI komponens absztrakt ősosztálya (*Abstract Base Class*). Ez az osztály definiálja a komponensek alapvető viselkedését és interfészét.

```
1 from core.models.geometry.position import Position
2 from core.models.event import Event
3 from abc import ABC, abstractmethod
4
5 class UIComponent(ABC):
```

```
6       def dispatch_event(self, event: Event) -> bool:
7           if hasattr(self, 'handled_events') and event.type not
                in self.handled_events:
8               return False
9
10          return self.handle_event(event)
11
12      def handle_event(self, event: Event) -> bool:
13          """Process an event that has already been filtered by
                type. Return True if consumed."""
14          return False
15
16      @abstractmethod
17      def render(self, screen_pos: Position) -> None:
18          """Render the UI component."""
19          pass
20
21      @abstractmethod
22      def contains(self, screen_pos: Position) -> bool:
23          """Check if a position is within the component's area.
                """
24          return False
25
26      def tick(self) -> None:
27          """Advance the component's state by one tick."""
28          pass
```

3.3. forráskód. Az UIComponent absztrakt osztály

`rectangle_ui_component.py` Téglalap alakú UI elemeket valósít meg. Az inicializáció során megadott befoglaló téglalapot elmenti az osztály állapotába, és implementálja a `contains` függvényt a geometriai vizsgálathoz.

`full_screen_ui_component.py` A teljes képernyőt elfoglaló UI komponens. A `contains` metódusa minden esetben `True` értéket ad vissza, biztosítva, hogy minden eseményt érzékeljen a képernyőn.

`shortcut_ui_component.py` Billentyűparancsok kezelésére szolgáló komponens. A konstruktorban megadott billentyűkombináció figyelését végzi, és lenyomás esetén meghívja a hozzárendelt visszahívó (callback) függvényt.

`clickable_ui_component.py` Kattintható UI komponensek osztálya. Kiszűri a "húzott egér" (*drag*) eseményeket, és valid kattintás esetén meghívja az `on_click` metódust. A származtatott osztályoknak így csupán az `on_click` logikát kell implementálniuk.

`button_component.py` Összetett komponens, amely egyesíti a `shortcut` és a `clickable` osztályok funkcionalitását.

`panel.py` A `rectangle` és `clickable` osztályok leszármazottja. Alapértelmezetten definiál egy panelt a képernyő alsó középső részén. A konstruktor paraméterei lehetővé teszik a pozíció és méret testreszabását, valamint segédfunkciókat biztosít a betűtípusok kezeléséhez. A `render` metódus felelős a háttér és a keret kirajzolásáért, így a származtatott osztályok kizárólag a tartalom megjelenítésére koncentrálhatnak.

`ui_controller.py` Az UI komponensek központi gyűjtője és kezelője.

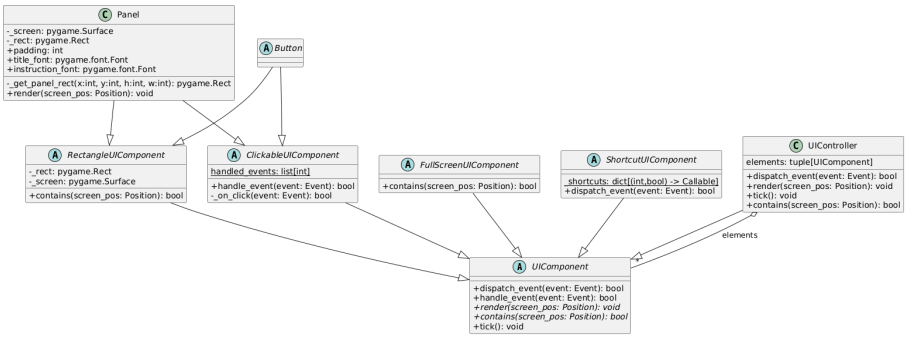
- **Eseménykezelés:** Az eseményeket a komponensek definiált sorrendjében továbbítja. Amennyiben egy komponens lekezeli az eseményt, a továbbítás megáll (chain of responsibility minta).
- **Kurzorkezelés:** Érzékeli, melyik komponens felett tartózkodik a kurzor. A takart komponensek felé nem továbbítja a kurzor pozícióját, így azok nem rajzolnak feleslegesen előnézetet (*preview*).
- **Kirajzolás:** Felelős az összes regisztrált komponens `render` és `tick` metódusainak hívásáért.

```
1  from shared.ui.models.ui_component import UIComponent
2  from core.models.event import Event
3
4  class UIController(UIComponent):
5      elements: tuple[UIComponent]
6
7      def dispatch_event(self, event: Event):
8          for element in self.elements:
9              if element.dispatch_event(event):
10                 return True
11
12         return False
```

```
13
14      def render(self, screen_pos):
15          elements_above_cursor = []
16          if screen_pos is not None:
17              for element in self.elements:
18                  elements_above_cursor.append(element)
19                  if element.contains(screen_pos):
20                      break
21
22          for element in reversed(self.elements):
23              if element in elements_above_cursor:
24                  element.render(screen_pos)
25              else:
26                  element.render(None)
27
28      def tick(self):
29          for element in self.elements:
30              element.tick()
31
32      def contains(self, screen_pos):
33          return any(element.contains(screen_pos) for element in
               self.elements)
```

3.4. forráskód. Az UIController osztály

A UI modellek közötti öröklődési és kapcsolati viszonyokat az 3.1. ábra szemlélteti.



3.1. ábra. Az UI komponensek UML osztálydiagramja

Modellek (Models)

app_state.py Az alkalmazás állapotát reprezentáló osztály. Tárolja az aktuális projektet, a mentési állapotot, valamint az alkalmazás fázisát (pl. *Setup*,

Simulation).

Vezérlők (Controllers)

A `controllers` könyvtár tartalmazza az alkalmazás logikai vezérlőit.

`app_controller.py` **(szülő:** `ui_controller`**)** A projekt nézet belépési pontja. Feladatai közé tartozik:

- A projekt inicializálása vagy betöltése.
- A beérkező nyers események átalakítása az `event.py`-ban definiált belső eseménytípusokká.
- Hibák vizuális kezelése.
- Események továbbítása az UI komponensek felé.

`app_phase_strategy.py` **(szülő:** `full_screen_ui_component`**)** Stratégia osztály, amely a projekt különböző fázisainak (pl. *Setup*, *Simulation*) állapotát kezeli és továbbítja a megfelelő modulok felé.

`camera_controller.py` A nézet transzformációjáért felelős osztály. Kezeli a kamera pozícióját (pan) és nagyítási szintjét (zoom), valamint feldolgozza a vezérléshez kapcsolódó felhasználói bemeneteket.

Megosztott UI Komponensek

A `components` könyvtárban találhatóak azok az általános elemek, amelyeket több projekt modul is felhasznál:

- **alert_component.py**: Általános figyelmeztető üzenetek és modális ablakok megjelenítésére szolgáló komponens.
- **input_component.py**: Általános szövegbeviteli mező.
- **zoom_button.py**: A nézet alaphelyzetbe állításáért és a jelenlegi nagyítás megjelenítéséért felelős gomb.

Segédfunkciók és Szolgáltatások

A rendszer működését támogató egyéb osztályok és függvények.

Utils (Rajzolási segédfunkciók) A `utils` könyvtár tartalmazza a grafikus megjelenítéshez szükséges alacsony szintű függvényeket:

- **grid.py**: A háttérrács kirajzolása.
- **lines.py**: Vonalak rajzolása, beleértve a szaggatott (`draw_dashed_line`) és pontozott (`draw_dotted_line`) stílusokat.
- **nodes.py**: Csomópontok (`draw_node`) és elágazások (`draw_junction_node`) megjelenítése.
- **signal.py**: Vasúti jelzők rajzolása (`draw_signal`).
- **draw_station.py**: Állomások vizuális megjelenítése(`draw_station`).
- **tracks.py**: Vágányok kirajzolása (`draw_track`).

Services és Enums

`services/color_from_speed.py` Segédfunkció, amely a vágány sebessége alapján meghatározza annak megjelenítési színét, vizuális visszajelzést adva a sebességkorlátozásokról.

`enums/edge_action.py` Enumeráció, amely a vágányokkal kapcsolatos kirajzolási és szerkesztési módokat definiálja.

3.3.4. Setup Modul

Az alkalmazás alapértelmezett állapota; ez a modul töltődik be új projekt létrehozásakor vagy egy meglévő betöltésekor. A Setup modul két almodult foglal magában:

- **Construction modul:** Pályaszerkesztési funkciók.
- **Train Placement modul:** Vonatok elhelyezése a pályán.

Architektúrális szempontból ezek az almodulok szándékosan nem a globális `shared` rétegben kaptak helyet, mivel funkcionalitásuk kizárólag a Setup fázisra korlátozódik. Ezzel a megoldással csökkenthető a globális névterek szennyezése és a felesleges függőségek kialakulása.

Vezérlők (Controllers) A modul működéséért felelős vezérlő osztályok:

`setup_mode.py` A Setup modul fő vezérlője. Felelős a két szerkesztési mód (építés és elhelyezés) közös funkcióinak vezérléséért, valamint a szimuláció indítása előtti inicializálási lépésekért (pl. vonatok alaphelyzetbe állítása).

`setup_state.py` Állapotkezelő osztály, amely nyilvántartja az éppen aktív szerkesztési módot.

`setup_mode_strategy.py` A Strategy tervezési mintát megvalósító osztály, amely a Construction és Train Placement módok közötti dinamikus váltást és az azokhoz tartozó specifikus logikák cseréjét kezeli.

UI Komponensek A `ui` könyvtár tartalmazza mindkét Setup almodul által közösen használt felületi elemeket:

- **Módválasztó és Vezérlés:**
  - `setup_mode_selector_buttons.py`: A két szerkesztési mód közötti váltást lehetővé tevő gombsor.
  - `start_simulation_button.py`: A szerkesztés lezárását és a szimuláció indítását kezdeményező gomb.
  - `timetable_button.py`: A menetrend-szerkesztő felület megnyitására szolgáló gomb.
- **Fájl- és Rendszerműveletek:**
  - `save_button.py`: A projekt aktuális állapotának mentése.
  - `open_button.py`: Meglévő projektfájl betöltése.
  - `exit_button.py`: Kilépés a Setup módból vissza a főmenübe (*Home Page*).

3.3.5. Construction (Pályaszerkesztő) Modul

A Construction modul a Setup fázison belül a pályaszerkesztési és infrastruktúraépítési funkciókat valósítja meg. A modul architektúrája szigorúan szétválasztja a megjelenítést, az állapotkezelést és a vezérlést.

`construction_mode.py` A modul központi vezérlője. Feladata a szerkesztési nézet inicializálása, az eszközök példányosítása és a fő eseményhurok kezelése a szerkesztés alatt.

Modellek és Absztrakciók (`models`) A `models` könyvtár tartalmazza a szerkesztőeszközök működéséhez szükséges ősosztályokat és a közös állapotkezelőt:

- **construction_state.py**: A szerkesztési mód állapotkezelője. Ez az osztály tárolja az aktuálisan kiválasztott eszközt, valamint a szerkesztési előnézetek (*previews*) állapotát (pl. egy platform elhelyezésének vizualizációját).
- **construction_tool_panel.py**: Az egyes eszközökhöz (*Tools*) tartozó információs és beállító panelek absztrakt ősosztálya.
- **construction_tool_controller.py**: Az egyes szerkesztőeszközök vezérlőinek közös ősosztálya.
- **construction_tool_view.py**: Az egyes eszközök nézeti logikájának absztrakt ősosztálya.

Felhasználói Felület (`ui`) A felhasználói felület elemei a közös keretrendszerre és a specifikus eszköz-implementációkra tagolódnak.

`construction_buttons.py` A különböző szerkesztőeszközök közötti váltást biztosító gombsor.

`construction_common_view.py` A szerkesztőnézet alaprétege. Felelős a statikus elemek (koordináta-rendszer, meglévő vágányok, jelzők, állomások, peronok) kirajzolásáért. Az aktív eszközök erre a rétegre rajzolják rá a saját dinamikus előnézeteiket, emellett bizonyos előnézeti logikákat is ez az osztály kezel.

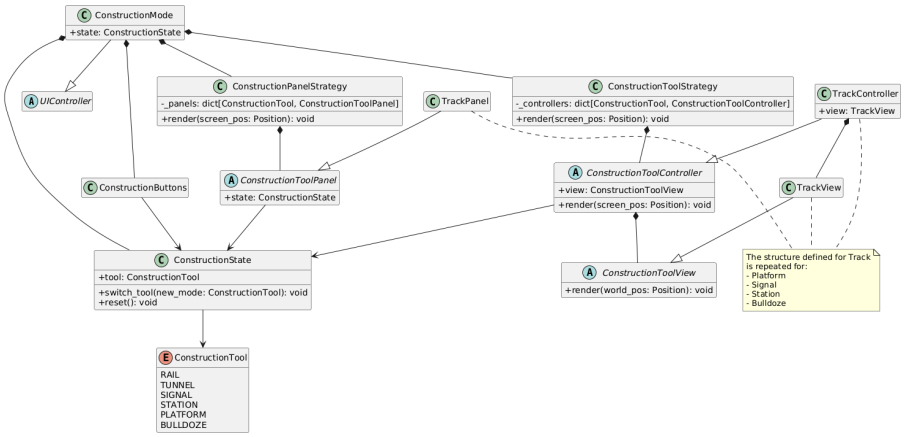
`construction_panel_strategy.py` A *Strategy* tervezési minta alapján kezeli a dinamikus panelváltást, biztosítva, hogy mindig a kiválasztott eszközhöz (pl. váltóépítés) tartozó beállítások jelenjenek meg.

`construction_tool_strategy.py` A *Strategy* tervezési minta alapján kezeli az eszközök közötti váltást, biztosítva, hogy mindig a kiválasztott eszköz vezérlője és nézete legyen aktív.

Szerkesztőeszközök (*Tools*) felépítése: A rendszerben minden szerkesztőeszköz (pl. vágányépítő, jelzőlerakó) egységes, négy komponensből álló architektúrát követ:

1. **Controller:** Kezeli a felhasználói interakciókat (kattintás, billentyűzet) és frissíti az eszköz belső állapotát.
2. **View:** Megvalósítja a vizuális logikát, kirajzolja az eszköz-specifikus előnézeteket, és manipulálja a `construction_state` előnézeti állapotait.
3. **Panel:** Az eszközhöz tartozó specifikus beállításokat és a felhasználói instrukciókat megjelenítő felületi elem.
4. **Target:** A nézet és a vezérlő közös geometriai logikáját tömörítő segédosztály. Feladata, hogy a kurzor pozíciója alapján azonosítsa a pályán végezhető lehetséges akciókat (pl. vágányvéghez való csatlakozás detektálása).

A Construction modul és a benne található eszközök strukturális felépítését az 3.2. ábra szemlélteti.



3.2. ábra. A Construction modul és az eszközök UML osztálydiagramja

3.3.6. Train Placement (Vonatelhelyező) Modul

A Train Placement modul a Setup fázison belül a vonatelhelyezési funkciókat valósítja meg. A modul architektúrája hasonló a construction modul-éhoz. A megjelenítés, az állapotkezelés és a vezérlés itt is szétválasztásra került.

`train_placement_mode.py` A modul központi vezérlője. Feladata a vonatelhelyezési nézet inicializálása, az eszközök példányosítása és a fő eseményhurok kezelése a vonatelhelyezés alatt.

subparagraphModellek és Absztrakciók (`models`) A `models` könyvtár tartalmazza a vonatelhelyező eszközök működéséhez szükséges ősosztályokat és a közös állapotkezelőt:

- **train_placement_state.py**: A vonatelhelyezési mód állapotkezelője. Ez az osztály tárolja az aktuálisan kiválasztott eszközt, valamint a vonatelhelyezési előnézetek (*previews*) állapotát (pl. egy törlendő vonat azonosítóját).
- **train_placement_tool_controller.py**: Az egyes vonatelhelyező eszközök vezérlőinek közös ősosztálya.
- **train_placement_tool_view.py**: Az egyes eszközök nézeti logikájának absztrakt ősosztálya.

Felhasználói Felület (`ui`) A felhasználói felület elemei a közös keretrendszerre és a specifikus eszköz-implementációkra tagolódnak.

`train_placement_buttons.py` A különböző vonatelhelyező eszközök közötti váltást biztosító gombsor.

`train_placement_common_view.py` A vonatelhelyezési nézet alaprétege. Felelős a statikus elemek (koordináta-rendszer, meglévő vágányok, jelzők, állomások, peronok) kirajzolásáért. Az aktív eszközök erre a rétegre rajzolják rá a saját dinamikus előnézeteiket, emellett bizonyos előnézeti logikákat is ez az osztály kezel.

`train_placement_panel_strategy.py` A *Strategy* tervezési minta alapján kezeli a dinamikus panelváltást, biztosítva, hogy mindig a kiválasztott eszközhöz (pl. vonat elhelyezése) tartozó beállítások jelenjenek meg.

`train_placement_tool_strategy.py` A *Strategy* tervezési minta alapján kezeli az eszközök közötti váltást, biztosítva, hogy mindig a kiválasztott eszköz vezérlője és nézete legyen aktív.

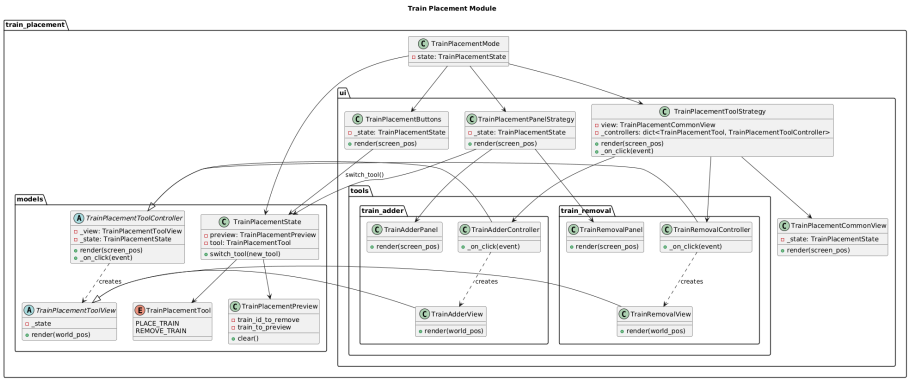
Vonatelhelyező eszközök (*Tools*) felépítése: A rendszerben minden vonatelhelyező eszköz (pl. vonat elhelyezése, vonat törlése) egységes, három komponensből álló architektúrát követ:

1. **Controller:** Kezeli a felhasználói interakciókat (kattintás, billentyűzet) és frissíti az eszköz belső állapotát.

2. **View:** Megvalósítja a vizuális logikát, kirajzolja az eszköz-specifikus előnézeteket, és manipulálja a `train_placement_state` előnézeti állapotait.

3. **Panel:** Az eszközhöz tartozó specifikus beállításokat és a felhasználói instrukciókat megjelenítő felületi elem.

A Train Placement modul és a benne található eszközök strukturális felépítését az 3.3. ábra szemlélteti.



3.3. ábra. A Train Placement modul és az eszközök UML osztálydiagramja

3.3.7. Simulation (Szimulációs) Modul

A Simulation modul a rendszer végrehajtó egysége, amely a vonatok valós idejű irányítását, a biztosítóberendezési logika (jelzők, vágányutak) érvényesítését és a fizikai szimuláció futtatását végzi. A modul architektúrája következetesen alkalmazza a modell-nézet-vezérlő (MVC) mintát, szigorúan szétválasztva az állapotkezelést, a megjelenítést és a vezérlési logikát.

`simulation_mode.py` A szimulációs környezet belépési pontja és fő vezérlője (*Main Controller*). Feladata a futtatókörnyezet inicializálásáért, beleértve a biztosítóberendezés (interlocking) logikájának példányosítását, valamint a központi eseményhurok felügyeletét.

Modellek (`models`) Az üzleti logikát és az adatszerkezeteket tároló réteg:

`simulation_state.py` A szimuláció központi állapotkezelő entitása. Feladata a szimulációs idő szinkronizációja, az aktívan kijelölt járművek nyilvántartása, valamint a vágányút-beállítások vizuális előnézetének (*preview*) menedzselése.

Felhasználói Felület (`ui`)

A `ui` könyvtár tartalmazza a szimuláció vezérlését és a vizuális visszajelzést biztosító grafikus komponenseket.

Általános UI elemek

`simulation_controller.py` A modul interakciós logikáját megvalósító osztály. Közvetít a felhasználói bemenetek (egér, billentyűzet) és a szimulációs modell között, biztosítva az állapotváltozások propagálását.

`simulation_view.py` A grafikus megjelenítésért felelős osztály. Feladata a dinamikus objektumok (vonatok), a statikus infrastruktúra (jelzők, vágányok) és az aktív vágányutak valós idejű renderelése a képernyőre.

`time_control_buttons.py` A szimulációs idő manipulálását lehetővé tevő vezérlőfelület, amely funkciókat biztosít a szimuláció gyorsítására, lassítására vagy szüneteltetésére.

`time_display.py` A szimulált rendszeridő digitális kijelzője.

`end_simulation_button.py` A szimulációs folyamat terminálására és a szerkesztő (Setup) módba való visszatérésre szolgáló vezérlőelem.

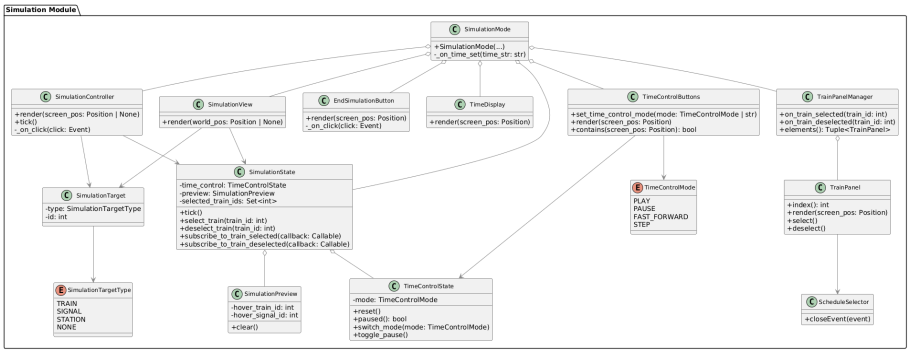
Vonatvezérlő Panelek (`ui/panel`) A járműirányítás funkcionalitásának komplexitása indokolta, hogy a vonatvezérléssel kapcsolatos komponensek egy dedikált `panel` csomagba kerüljenek.

`train_panel_manager.py` A járműpanelek életciklus-kezelője. Dinamikusan menedzseli a felhasználó által kijelölt vonatokhoz tartozó vezérlőfelületek példányosítását, megjelenítését és frissítését.

`train_panel.py` Járműspecifikus irányítópult. Interfészt biztosít a mozdonyvezérléshez (indítás, irányváltás), továbbá valós időben megjeleníti a vonat telemetriai adatait (pl. aktuális sebesség) és a hozzárendelt menetrend státuszát.

`schedule_selector.py` Menetrend-kiválasztó komponens (Qt widget integrációval). Lehetővé teszi, hogy a felhasználó az adatbázisban rendelkezésre álló menetrendek közül egyet hozzárendeljen az adott szerelvényhez.

A Simulation modul felépítését és az osztályok közötti kapcsolatokat a 3.4. ábra szemlélteti.



3.4. ábra. A Simulation modul és komponenseinek UML osztálydiagramja

3.3.8. Timetable (Menetrend) Modul

A `timetable` modul biztosítja a vonatmenetrendek teljes körű kezelését, beleértve azok grafikus megjelenítését és. Technológiai szempontból ez a modul elkülönül a projekt többi részétől: a grafikus felhasználói felület (GUI) a `PyQt6` keretrendszer natív widgetjeire épül, kihasználva annak fejlett ablakkezelési és eseményvezérlési képességeit a komplex adatbeviteli feladatokhoz.

Ablakok és Dialógusok

`timetable_window.py` Egy `QDialog` (PyQt6) alapú osztály, amely a menetrendkezelés belépési pontjaként szolgál. Feladata a rendszerben tárolt menetrendek listázása, rendszerezése és a kezelési műveletek (létrehozás, törlés, módosítás) koordinálása.

`timetable_editor_dialog.py` Egy `QMainWindow` (PyQt6) alapú összetett szerkesztőfelület. Ez az osztály valósítja meg a menetrendek részletes szerkesztéséhez szükséges logikát, lehetővé téve az állomások, megállási idők és indulási feltételek precíz konfigurálását.

Stílusdefiníciók (`stylesheets`)

A felületek vizuális stílusát az alábbi fájlok definiálják:

- `timetable_editor_stylesheet.py`: A szerkesztőablak komponenseinek vizuális szabálykészlete.

- `timetable_window_stylesheet.py`: A menetrend-választó ablak stílusleírója.

3.4. Tesztelés

4. fejezet

Összegzés

Lorem ipsum dolor sit amet, consectetur adipiscing elit. In eu egestas mauris. Quisque nisl elit, varius in erat eu, dictum commodo lorem. Sed commodo libero et sem laoreet consectetur. Fusce ligula arcu, vestibulum et sodales vel, venenatis at velit. Aliquam erat volutpat. Proin condimentum accumsan velit id hendrerit. Cras egestas arcu quis felis placerat, ut sodales velit malesuada. Maecenas et turpis eu turpis placerat euismod. Maecenas a urna viverra, scelerisque nibh ut, malesuada ex.

Aliquam suscipit dignissim tempor. Praesent tortor libero, feugiat et tellus porttitor, malesuada eleifend felis. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Nullam eleifend imperdiet lorem, sit amet imperdiet metus pellentesque vitae. Donec nec ligula urna. Aliquam bibendum tempor diam, sed lacinia eros dapibus id. Donec sed vehicula turpis. Aliquam hendrerit sed nulla vitae convallis. Etiam libero quam, pharetra ac est nec, sodales placerat augue. Praesent eu consequat purus.

Köszönetnyilvánítás

Amennyiben a szakdolgozati / diplomamunka projekted pénzügyi támogatást kapott egy projektből vagy az egyetemtől, jellemzően kötelező feltüntetni a dolgozatban is. A dolgozat elkészítéséhez segítséget nyújtó oktatók, hallgatótársak, kollégák felé is nyilvánítható külön köszönet.

A. függelék

Szimulációs eredmények

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque facilisis in nibh auctor molestie. Donec porta tortor mauris. Cras in lacus in purus ultricies blandit. Proin dolor erat, pulvinar posuere orci ac, eleifend ultrices libero. Donec elementum et elit a ullamcorper. Nunc tincidunt, lorem et consectetur tincidunt, ante sapien scelerisque neque, eu bibendum felis augue non est. Maecenas nibh arcu, ultrices et libero id, egestas tempus mauris. Etiam iaculis dui nec augue venenatis, fermentum posuere justo congue. Nullam sit amet porttitor sem, at porttitor augue. Proin bibendum justo at ornare efficitur. Donec tempor turpis ligula, vitae viverra felis finibus eu. Curabitur sed libero ac urna condimentum gravida. Donec tincidunt neque sit amet neque luctus auctor vel eget tortor. Integer dignissim, urna ut lobortis volutpat, justo nunc convallis diam, sit amet vulputate erat eros eu velit. Mauris porttitor dictum ante, commodo facilisis ex suscipit sed.

Sed egestas dapibus nisl, vitae fringilla justo. Donec eget condimentum lectus, molestie mattis nunc. Nulla ac faucibus dui. Nullam a congue erat. Ut accumsan sed sapien quis porttitor. Ut pellentesque, est ac posuere pulvinar, tortor mauris fermentum nulla, sit amet fringilla sapien sapien quis velit. Integer accumsan placerat lorem, eu aliquam urna consectetur eget. In ligula orci, dignissim sed consequat ac, porta at metus. Phasellus ipsum tellus, molestie ut lacus tempus, rutrum convallis elit. Suspendisse arcu orci, luctus vitae ultricies quis, bibendum sed elit. Vivamus at sem maximus leo placerat gravida semper vel mi. Etiam hendrerit sed massa ut lacinia. Morbi varius libero odio, sit amet auctor nunc interdum sit amet.

Aenean non mauris accumsan, rutrum nisi non, porttitor enim. Maecenas vel tortor ex. Proin vulputate tellus luctus egestas fermentum. In nec lobortis risus,

sit amet tincidunt purus. Nam id turpis venenatis, vehicula nisl sed, ultricies nibh. Suspendisse in libero nec nisi tempor vestibulum. Integer eu dui congue enim venenatis lobortis. Donec sed elementum nunc. Nulla facilisi. Maecenas cursus id lorem et finibus. Sed fermentum molestie erat, nec tempor lorem facilisis cursus. In vel nulla id orci fringilla facilisis. Cras non bibendum odio, ac vestibulum ex. Donec turpis urna, tincidunt ut mi eu, finibus facilisis lorem. Praesent posuere nisl nec dui accumsan, sed interdum odio malesuada.

Irodalomjegyzék

[1] Károly Kühne. „Integrált Ütemes Menetrend I.” (2008). Elérés dátuma: 2025-11-28.

Ábrák jegyzéke

Táblázatok jegyzéke

Algoritmusjegyzék

Forráskódjegyzék