Szakdolgozat

Miskolci Egyetem

Grafikus formák, útvonalak integrálása és optimalizálása térképeken

Készítette:

Buha Milán

Programtervező informatikus

Témavezető:

Szabó Martin

Miskolc, 2024

Miskolci Egyetem
Gépészmérnöki és Informatikai Kar
Alkalmazott Matematikai Intézeti Tanszék **Szám:**

Szakdolgozat Feladat

Buha Milán (IY5AM2) programtervező informatikus jelölt részére.

A szakdolgozat tárgyköre: Alkalmazásfejlesztés

A szakdolgozat címe: Grafikus formák, útvonalak integrálása és optimalizálása térképeken

A feladat részletezése:

1. A szakdolgozat célja egy speciális felületen történő rajzolási funkció kidolgozása, melynek segítségével az elkészült ábrák automatikusan integrálhatók egy Google Maps térkép úthálózatához.

2. Szakirodalmi áttekintés során elemezze a térképi integrációs és grafikus illesztési technológiákat, különös tekintettel azok hatékonyságára és felismerési képességére!

3. Tervezzen meg egy alkalmazást, amely lehetővé teszi a felhasználók számára, hogy grafikus ábrákat készítsenek, majd ezeket az ábrákat az úthálózatnak megfelelően illeszkedjenek a térképre!

4. Alakítsa ki az alkalmazás adatkezelési és funkcionális modelljét! Hozzon létre egy valós idejű illesztési eljárást az elkészített rajzok és a térképi úthálózat közötti kohézió optimalizálása érdekében!

5. Implementáljon illesztési stratégiát és optimalizálási technikát a rajzok hatékony és pontos térképre való transzformálásához!

6. Készítsen egy felhasználói felületet az alkalmazás számára, mely bemutatja az eredményeket!

7. A létrehozott illesztési algoritmusok segítségével, futási eredmények alapján értékelje a technikák hatékonyságát és pontosságát, különös tekintettel az eredeti rajz felismerhetőségére!

Témavezető: Szabó Martin (tanársegéd)

A feladat kiadásának ideje: 2023.08.24.

. . . . . .
szakfelelős

Eredetiségi Nyilatkozat

Alulírott **Buha Milán**; Neptun-kód: `IY5AM2` a Miskolci Egyetem Gépészmérnöki és Informatikai Karának végzős Programtervező informatikus szakos hallgatója ezennel büntetőjogi és fegyelmi felelősségem tudatában nyilatkozom és aláírásommal igazolom, hogy *Grafikus formák, útvonalak integrálása és optimalizálása térképeken* című szakdolgozatom saját, önálló munkám; az abban hivatkozott szakirodalom felhasználása a forráskezelés szabályai szerint történt.

Tudomásul veszem, hogy szakdolgozat esetén plágiumnak számít:

- szószerinti idézet közlése idézőjel és hivatkozás megjelölése nélkül;
- tartalmi idézet hivatkozás megjelölése nélkül;
- más publikált gondolatainak saját gondolatként való feltüntetése.

Alulírott kijelentem, hogy a plágium fogalmát megismertem, és tudomásul veszem, hogy plágium esetén szakdolgozatom visszautasításra kerül.

Miskolc, ... év ..... hó ..... nap

.....
Hallgató

1. A szakdolgozat feladat módosítása

szükséges (módosítás külön lapon)
nem szükséges

....
dátum témavezető(k)

2. A feladat kidolgozását ellenőriztem:

témavezető (dátum, aláírás):	konzulens (dátum, aláírás):
.....	.....
.....	.....
.....	.....

3. A szakdolgozat beadható:

....
dátum témavezető(k)

4. A szakdolgozat ..... szövegoldalt
..... program protokollt (listát, felhasználói leírást)
..... elektronikus adathordozót (részletezve)
.....
..... egyéb mellékletet (részletezve)
.....
tartalmaz.

....
dátum témavezető(k)

5. A szakdolgozat bírálatra

bocsátható
nem bocsátható

A bíráló neve: .....

....
dátum szakfelelős

6. A szakdolgozat osztályzata

a témavezető javaslata: .....
a bíráló javaslata: .....
a szakdolgozat végleges eredménye: .....

Miskolc, ...
a Záróvizsga Bizottság Elnöke

Tartalomjegyzék

1. Bevezetés **1**

2. Alapötletek és előkészületek **2**
2.1. Probléma bemutatása 2
2.2. Illesztési stratégiák 2
2.3. Gráfelmélet alkalmazhatóságának felismerése 5
2.4. Gráf egyszerűsítés Douglas-Peucker algoritmussal 6
2.5. Útkereső algoritmusok 8
2.6. Projektépítés Gradle segítségével 10
2.7. XML használata a felhasználói felülethez 11

3. Az alkalmazás megtervezése **12**
3.1. Alkalmazott technológiák 13

4. Alkalmazás fejlesztése **17**
4.1. Mobilalkalmazás fejlesztés alapjai 17
4.1.1. Gráfelmélet és alkalmazása 19
4.2. Rajzolási folyamatok és aszinkron programozás 21
4.3. Grafikai interakciók és adatkezelés optimalizálása 23
4.3.1. Douglas-Peucker algoritmus használata 24
4.4. A* útkereső és DFS gráfbejáró algoritmusokon alapuló módszer 25
4.5. Alakzat felismerésen alapuló módszer 27
4.6. Histogramok és Chi-négyzet alkalmazása 28
4.7. Poisson-disk mintavétel generálása 30
4.8. Az illesztési folyamat és felhasználói visszajelzés 32
4.9. Az illesztések személyre szabása 34

5. Ellenőrzés és validálás **36**

6. Fejlesztési lehetőségek **39**

7. Összefoglalás **40**

8. Köszönetnyilvánítás **41**

Források **42**

1. fejezet

Bevezetés

A dolgozat a korszerű alkalmazásfejlesztés egy rendkívül aktuális, szórakoztató és kihívást jelentő területére fókuszál: a grafikus formák és útvonalak térképeken történő integrálására és optimalizálására. A téma nem csupán a szoftverfejlesztés technikai aspektusait öleli fel, hanem szorosan kapcsolódik az adatkezelési és feldolgozási stratégiákhoz, a gráfelmélet terüteihez, valamint a gépi látás eredményeihez is. A projekt célja egy olyan kísérleti mobilalkalmazás kifejlesztése, amely lehetővé teszi a felhasználók számára, hogy saját grafikus ábráikat könnyedén integrálják a Google Maps, vagy hasonló térkép alkalmazások úthálózatához.

A dolgozat első része szakirodalmi áttekintést nyújt a térkép-integrációs technológiákról és grafikus illesztési módszerekről, kiemelve azok hatékonyságát és felismerési képességüket. Ez után részletezem az alkalmazás tervezési folyamatát, az adatkezelési és funkcionális modell megalkotását, valamint egy valós idejű illesztési eljárást, amely optimalizálja a rajzok és a térképi úthálózat közötti kohéziót.

Az implementációs szakasz során a dolgozat bemutat néhány specifikus illesztési stratégiát és optimalizálási technikát, amelyek célja a rajzok hatékony és pontos térképre való transzformálása. Ezenkívül kitér a felhasználói felület kialakítására is, mely bemutatja az illesztési folyamat eredményeit. A dolgozat záró részében a létrehozott algoritmusok teljesítményét és pontosságát értékelem, különös tekintettel a rajzok eredeti formájának megőrzésére és felismerhetőségére.

A projekt aktualitását és relevanciáját növeli, hogy a dolgozat nem csupán elméleti kutatásokra és fejlesztésekre támaszkodik, hanem egy valós, piacra kész alkalmazás létrehozására is vállalkozik. Ezáltal az alkalmazás hozzájárul a felhasználók szórakoztatásához és az egészséges életmód előmozdításához interaktív és szórakoztató testmozgási lehetőségekkel, miközben új módszereket kínál a digitális térképekkel való interakcióra. Bár az alkalmazás összekapcsolódik a térinformatikai elemekkel a térképek és adatok használata révén, elsődleges célja a felhasználói élmény gazdagítása és a mindennapi tevékenységekbe való játékos integráció.

2. fejezet

Alapötletek és előkészületek

A fejezetben bemutatom azokat a kulcsfontosságú technológiák, algoritmusokat és elméleti alapokat, amelyekre az alkalmazás épül, különös tekintettel arra, hogyan illeszkednek ezek az innovációk a szélesebb informatikai és szoftverfejlesztési kontextusba.

2.1. Probléma bemutatása

Egy korábbi, hasonló törekvés a Washingtoni Egyetem munkatársai által fejlesztett Trace alkalmazás volt, amely már nem elérhető. [6] Bár nyilvánosan elérhető programkódok hiányában nehezen húzhatók párhuzamok, feltételezhető, hogy az alkalmazás hasonló megoldásokat alkalmazott a térképekhez való grafikus illesztés terén. Ez a példa rávilágít arra a kihívásra és szükségletre, amit a jelen dolgozat is igyekszik kielégíteni, amely a felhasználó által létrehozott grafikai elemek térképekhez való illesztésének és optimalizálásának automatizálására.

2.2. Illesztési stratégiák

Az alkalmazás két fő illesztési stratégiát különböztet meg. Ennek az oka, hogy kezdetben egy alapvető illesztési technikát alkalmaztam, ahol a felhasználó által létrehozott pontokat közvetlenül illesztem a térkép úthálózatához, majd az A [5] algoritmust alkalmazom az összeköttetések létrehozásához. Ez a megközelítés hatékony a főként primitív alakzatokat tartalmazó illesztési feladatok esetén, ahol a pontosság kevésbé kritikus és a felhasználónak van egy konkrét illesztési ötlete.

A fejlesztési folyamat során, amint egyre mélyebbre hatoltam a problémában és szembesültem a valóságos illesztési kihívásokkal, kezdtem kísérletezni egy összetettebb megközelítéssel, amely túllép a kiindulási módszer korlátain. Ez egy bonyolultabb, de sokkal precízebb illesztési technikát alkalmaz, a gráfokat egy előre definiált pontkészlet alapján eltolja és forgatja. Ezt követően az A* algoritmus automatikusan összeköti az illesztett pontokat, és a Shape Context alakzat felismerő algoritmus segítségével összehasonlítja az eredeti és az illesztett gráfokat, hogy megtalálja a lehető legjobb illesztést.

A cél, hogy a módszer képes legyen azonosítani a legideálisabb illesztési pozíciót, új dimenziót nyitott a kutatásomban. Ehhez olyan kiegészítő technikákat vezettem be, mint például a transzformációk (nagyítás, forgatás, eltolás) adaptív alkalmazását, amelyek segítenek az ideális illesztési pozíció megtalálásában. Ezen túlmenően, speciális gráf optimalizációs technikákat is alkalmazok, amelyekkel finomíthatom és tökéletesíthetem az eredményül kapott gráf minőségét. Ezek a fejlesztések összességében egy rendkívül hatékony és pontos illesztési modellt eredményeznek, amely jelentősen meghaladja a kezdeti egyszerű módszer által nyújtott lehetőségeket.

A Shape Context alkalmazása

Az alakzatokat logaritmikus polárkoordináta-rendszerben ábrázolva, a módszer kihasználja a pontok közötti relatív távolságokat és szögeket. A geometriai transzformációk, mint a forgatás és skálázás, ebben a rendszerben egyszerű eltolásokként kezelhetők, így a különböző méretű vagy irányú alakzatok összehasonlítása egyszerűsödik. A pontok közötti kapcsolatokat histogramokban rögzítve a módszer a kisebb deformációkat és zajokat figyelmen kívül hagyva biztosít egy stabil reprezentációt, és képes az alakzatok közötti legjobb egyezést azonosítani még akkor is, ha azok kissé el vannak tolva vagy deformálódva.

Shape Context algoritmus

Az algoritmus egy hatékony alakfelismerési módszer, amely jelentős előrelépést jelent a képfeldolgozás és alakillesztés területén. A módszer alapjait Belongie, Malik és Puzicha fektették le [1] cikkükben, amelyben bemutatták, hogy az alakzatokat jellemző pontok közötti relatív távolságok és szögek használatával hogyan lehet eredményesen azonosítani különböző geometriai alakzatokat. A módszer lényege, hogy minden alakot jellemző pont köré egy logaritmikus-polár koordináta rendszert épít. Ebben a koordináta rendszerben a pontok közötti relatív helyzet jellemzésére használják a távolságokat és szögeket, amelyeket histogramokban tárolnak. A módszer kihasználja azt a tényt, hogy a logaritmikus-polár koordináta rendszerben a geometriai transzformációk - mint például a forgatás vagy a nagyítás/kicsinyítés - egyszerű eltolásokként modellezhetők, ezáltal a különböző méretű vagy irányú alakok könnyedén összehasonlíthatóvá válnak.

A Shape Context módszer kiemelkedő előnye a robusztussága és alkalmazkodó képessége. Képes figyelmen kívül hagyni a kisebb deformációkat és zajokat, így erősen zajos képek esetén is megbízható eredményeket nyújt. Az algoritmus széles körben alkalmazható, többek között kézírás-felismerés, objektumfelismerés, és képek közötti illesztési feladatokban.

Mindazonáltal, a módszernek vannak korlátai. [18] Egyik gyakori kritika, hogy a módszer önmagában nem elegendő ahhoz, hogy megbízható pontmegfeleltetéseket nyújtson zsúfolt ábrázoláskor. Nehézséget jelent a skálázási paraméter helyreállítása is, mivel a radiális távolságok normalizálása a ponttávolságok átlagával vagy mediánjával nem mindig vezet kielégítő eredményre. Továbbá a tárgyhoz közeli objektum és nemobjektum pontok nehézkesen különböztethetők meg pusztán a Shape Context alapján. Ezenfelül előfordulhat, hogy a modell alakzatán egymáshoz közel eső pontokat a képen távoli pontokhoz rendeli az algoritmus, ami pontatlansághoz vezethet a képek illesztésében.

Annak ellenére, hogy a Shape Context algoritmus rendkívül hatékony, számos kihívást is rejt magában. Az egyik ilyen a nagy számítási igény, amely különösen nagy pontszám esetén jelentkezik. A kutatók folyamatosan dolgoznak az algoritmus optimalizálásán és a számítási hatékonyság javításán, hogy az még több alkalmazási területen használható legyen. [1]

Poisson-disc mintavétel

A Poisson-féle mintavételi eljárás (Poisson-disc sampling)[2] egy népszerű algoritmus a számítógépes grafika és képfeldolgozás területén, amelyet arra használnak, hogy egy adott területen vagy térfogatban egyenletesen elosztott, de véletlenszerű pontokat generáljanak. Ez a módszer különösen hasznos azokban az esetekben, ahol szükség van a mintavételi pontok közötti minimális távolság fenntartására, így elkerülve azok túlzott összezsúfolódását vagy egyenlőtlen eloszlását. A Poisson-féle mintavétel ideális választás olyan alkalmazások számára, ahol fontos a mintapontok egyenletes elosztása, mint például textúrák generálása, modellfelületek elsimítása, vagy mintavételi pontok meghatározása térképi adatokhoz való illesztésekhez.

A Poisson-féle mintavételi eljárás alapelve, hogy minden újonnan generált pont esetében ellenőrizni kell, hogy az megfelel-e a minimális távolsági követelményeknek az összes többi már kiválasztott ponttal kapcsolatban. Ez a követelmény biztosítja, hogy a pontok ne legyenek túl közel egymáshoz, így elkerülve az összezsúfolódást és elősegítve egy egyenletes eloszlást.

A mintavételi eljárást használva az fejlesztett illesztési eljárás során generált pontokat egy előre meghatározott körön belül helyezzük el, ami alapot biztosít a gráf transzformációihoz. Ezek a pontok szolgálnak referenciaalapként, amelyekhez a gráfot adaptívan illesztjük, lehetővé téve, hogy a legjobb lehetséges illeszkedést érjük el az adott térképi adatokhoz. Az adaptív illesztési folyamat során a Poisson-féle mintavétellel generált pontok különösen hasznosak lehetnek, mivel lehetővé teszik a gráf finomra hangolt eltolását, forgatását és skálázását. Ezáltal az algoritmus képes azonosítani a térkép adott részéhez leginkább illeszkedő gráfkonfigurációt, maximalizálva ezzel az illesztés pontosságát és minőségét. A Poisson-féle mintavétel alkalmazása komplex gráf illesztési problémákban számos technikai kihívást vet fel, többek között a mintavételi pontok generálásának hatékonyságát és a transzformációk pontos alkalmazását. Az optimalizáció és a számítási hatékonyság javítása érdekében a kutatók különféle megközelítéseket és algoritmusokat fejlesztettek ki, mint például gyorsító adatszerkezetek használata a mintavételi pontok gyors kiválasztásához és a minimális távolságok ellenőrzéséhez.

2.3. Gráfelmélet alkalmazhatóságának felismerése

A gráfelmélet az alkalmazott matematika egyik alapvető területe, amelyet széles körben használnak a számítógépes tudományokban, különösen az algoritmusok és adatstruktúrák terén. A gráfok olyan matematikai struktúrák, amelyek pontok (csúcsok) és élek halmazából állnak, ahol az élek két pontot kötnek össze. Ez a reprezentációs forma rendkívül hatékony eszközt biztosít a kapcsolatok, összeköttetések vizualizálására és elemzésére számos különböző kontextusban.

A rajzok és képi adatok gráfokként való kezelése lehetővé teszi az alakzatok, mintázatok és összefüggések pontos és rugalmas reprezentálását.Ebben a kontextusban a pontok megfelelhetnek a rajz bizonyos kulcsfontosságú helyzetének, míg az élek a pontok közötti viszonyt, kapcsolatot jelenthetik. Ez a megközelítés különösen hasznos a digitális képfeldolgozás [15], a térinformatikai rendszerek [12] területén, illetve a képalkotó rendszerek, avagy az orvosi képalkotásnál [14] – például a gráfok használata agykérgi régiók szegmentálásában.

A Douglas-Peucker [13] algoritmus egy népszerű vonal-simplifikáló technika, amelyet arra használnak, hogy csökkentsék a pontok számát egy vonalon vagy görbén, miközben az alakzat általános formáját megőrzik. A gráfelméleti szempontból való megközelítés lehetővé teszi a rajzok és térképi adatok hatékony tömörítését, az adattárolás csökkentését, miközben az alapvető geometriai jellemzőket és információkat megőrzik.

Az útkereső algoritmusok, mint például az A* [5] algoritmus, kulcsfontosságú szerepet játszanak a gráf alapú modellekben. Ezek az algoritmusok lehetővé teszik a legrövidebb, legköltséghatékonyabb vagy egyéb szempontok szerint optimalizált útvonalak megtalálását két pont között. Az ilyen típusú algoritmusok használata kiterjed a robotika, a videojátékok mesterséges intelligenciájára, valamint a navigációs és logisztikai rendszerekre.

A Shape Context módszer különösen hatékony, amikor a rajzokat és képi adatokat gráfokként kezeljük. A gráfok pontjainak és éleinek elemzésével a Shape Context algoritmus képes felismerni és összehasonlítani különböző alakzatokat, függetlenül azok méretétől vagy orientációjától. Ez a módszer alapvető jelentőséggel bír az alakfelismerés, a képi adatok illesztése és a gépi tanulás területén.

2.4. Gráf egyszerűsítés Douglas-Peucker algoritmussal

A Douglas-Peucker algoritmus egy széles körben alkalmazott vonal-simplifikációs technika, amely lehetővé teszi egy poligon vonal vagy görbe pontjainak számának csökkentését, miközben az alakzat általános formáját nagy mértékben megőrzi. Ez az algoritmus különösen hasznos lehet a térinformatikai adatok és a grafikus felhasználói felületek kezelésekor, ahol az egyszerűsített geometriák gyorsabb feldolgozást és kevesebb tárhelyet igényelnek, miközben a lényeges információkat megőrzik.

Az algoritmust eredetileg Douglas és Peucker mutatta be 1973-ban. Az algoritmus lényege, hogy rekurzívan kiválasztja azokat a pontokat egy vonalról vagy görbéről, amelyek a legtöbb információt hordozzák az alakzat formájáról, és eltávolítja azokat a pontokat, amelyek kevésbé fontosak.

Az algoritmus működése a következő lépésekből áll:

1. Kiválasztja a vonal kezdő és végpontját, és meghatározza a legnagyobb távolságot ezek és a vonal többi pontja között.
2. Ha ez a távolság egy előre meghatározott küszöbérték alatt van, az algoritmus eltávolítja a köztes pontokat, mivel úgy ítéli meg, hogy ezek nem hozzáadott értéket jelentenek.
3. Ha a távolság meghaladja a küszöbértéket, az algoritmus a legtávolabbi pontot megtartja és két új szegmensre osztja a vonalat: egy a kezdőpont és a legtávolabbi pont között, és egy a legtávolabbi pont és a végpont között.
4. Az előző lépéseket rekurzívan alkalmazza mindkét új szegmensre.

A Douglas-Peucker algoritmus [13] alkalmazása során a fejlesztőknek dönteniük kell a küszöbérték megfelelő szintjéről, amely befolyásolja az alakzat simplifikálásának mértékét. Egy alacsonyabb küszöbérték több pontot tart meg, így jobban megőrzi az eredeti alakzatot, míg egy magasabb küszöbérték radikálisabb egyszerűsítést eredményez. Az algoritmus széles körben alkalmazott módszer a digitális térképezésben, a grafikus adatok kezelésében és az útvonaloptimalizálásban. Alkalmazása lehetővé teszi a nagy adatkészletek kezelésének megkönnyítését, az adattárolási követelmények csökkentését, és az adatok vizuális megjelenítésének javítását.

Azonban a módszer alkalmazása során több kihívással is szembesülhetünk:

- A küszöbérték megfelelő beállítása kritikus fontosságú, mivel ez határozza meg, hogy mennyire lesz pontos vagy egyszerűsített az alakzat reprezentációja.
- A pontok eltávolítása bizonyos esetekben fontos információk elvesztését eredményezheti, különösen, ha az alakzatoknak vannak kritikus geometriai jellemzői.
- A nagy adatkészletek esetén az algoritmus futtatása számításigényes lehet, ami optimalizálási kérdéseket vet fel.

Az algoritmus széles körű alkalmazhatósága és hatékonysága miatt számos területen került felhasználásra, amelyek túlmutatnak a digitális térképezésen és adatredukción. Az alábbiakban néhány példa található arra, hogy a Douglas-Peucker algoritmust és annak változatait hogyan alkalmazták különböző problémák megoldására, illetve hogyan inspirálták más kutatási területeket.

A Douglas-Peucker algoritmus felhasználható több iparágban és tudományterületen, elősegítve az adatok egyszerűsítését és kezelését:

1. Robotika és útvonaltervezés: A robotika területén az algoritmust az útvonalak és mozgási pályák egyszerűsítésére használják, csökkentve ezzel a navigációs algoritmusok számítási igényét. A simplifikált pályák segítenek a robotoknak hatékonyabban navigálni a környezetükben, minimalizálva az akadályokkal való ütközés kockázatát. [16]

2. Mesterséges intelligencia és gépi látás: A mesterséges intelligencia (MI) és gépi látás terén az algoritmus segítségével az objektumok kontúrjait egyszerűsítik, amely elősegíti az objektumfelismerést és klasszifikációt. A simplifikált kontúrok könnyebbé teszik a képfeldolgozási algoritmusok számára az alakzatok azonosítását és elemzését. [1]

3. Geoinformatika és térinformatika: A geoinformatikában az algoritmust a földrajzi adatok, például terepvonalak és vízrajzi hálózatok egyszerűsítésére használják. Ez lehetővé teszi a nagy mennyiségű térinformatikai adat hatékonyabb kezelését és tárolását, miközben a lényeges geográfiai információkat megőrzi. [12]

4. Mobilalkalmazások és webes technológiák: A webfejlesztés és mobilalkalmazások területén az algoritmust az adatok sávszélesség-barát továbbítása és megjelenítése érdekében használják. Az egyszerűsített alakzatok és vonalak gyorsabban tölthetők le és renderelhetők, javítva ezzel az alkalmazások teljesítményét és felhasználói élményét. [22]

5. Környezettudomány és ökológiai modellezés: Az ökológiai modellezésben az algoritmust a környezeti minták és élőhelyek határainak egyszerűsítésére használják, amely elősegíti az élőhely-fragmentáció és biodiverzitás kutatását. Az egyszerűsített modellek segítségével a kutatók pontosabban elemezhetik az ökológiai folyamatokat és a környezeti változások hatásait.

A Douglas-Peucker algoritmus és változatainak ezekben a különböző alkalmazási területeken való használata rávilágít arra, hogy az algoritmus alapvető jelentőséggel bír a modern számítástechnikában és adatfeldolgozásban. Az algoritmus adaptálhatósága és rugalmassága inspirálta a kutatókat és fejlesztőket, hogy új módszereket és technológiákat hozzanak létre, amelyek még hatékonyabban kezelik az adatokat és megoldják a komplex problémákat.

A további kutatások és fejlesztések várhatóan további innovatív alkalmazásokat és algoritmusokat fognak előtérbe hozni, amelyek a Douglas-Peucker algoritmus elveire épülnek. Ezek az új megoldások segítenek majd a tudomány és technológia különböző ágazataiban dolgozó szakembereknek még nagyobb kihívások leküzdésében.

2.5. Útkereső algoritmusok

Úttervező és útkereső algoritmusok kulcsfontosságú eszközök a számítástudomány, a robotika, a logisztika és a térinformatika területein. Ezek az algoritmusok lehetővé teszik az optimális vagy közel optimális útvonalak megtalálását két vagy több pont között egy gráfban vagy hálózatban. Az alábbiakban néhány közismert úttervező és útkereső algoritmus, valamint azok alkalmazási területei kerülnek bemutatásra.

A* algoritmus felhasználási területei

Az A* [5] (A csillag) algoritmust 1968-ban Peter Hart, Nils Nilsson és Bertram Raphael vezette be. Ez egy informált keresési algoritmus, amely heurisztikát használ az útvonaltervezés hatékonyságának növelésére. Az algoritmus különösen népszerű a videojátékok mesterséges intelligenciájában és a robotika területén, ahol gyors és hatékony útvonaltervezést igényelnek dinamikus környezetekben.

Az A*-algoritmus kulcsszerepet játszik a videojátékok mesterséges intelligenciájának kialakításában [11], ahol a karakterek és objektumok útvonaltervezését teszi lehetővé. Ez az algoritmus segít a játékfejlesztőknek dinamikus, kiszámítható és hatékony útvonalakat létrehozni, amelyek révén a játékbeli entitások intelligensen mozognak a virtuális világban. Az A*-algoritmus képes azonnal reagálni a környezet változásaira, például útakadályokra vagy ellenségek mozgására, így növelve a játékélmény realizmusát és kihívásait. A játékosok által tapasztalt mesterséges intelligencia ezáltal élőnek és kiszámíthatónak tűnik, ami fokozza a játék élményét és bevonását.

A robotikában [16] az A*-algoritmus a navigációs rendszerek alapköve, amely segít a robotoknak önállóan dönteni az optimális útvonalakról és mozgásukról. Az algoritmus lehetővé teszi a robotok számára, hogy hatékonyan tájékozódjanak bonyolult és dinamikusan változó környezetekben, például raktárakban. Az A*-algoritmus segítségével a robotok kiszámolják a legkevesebb költséggel járó útvonalat a kiinduló ponttól a célállomásig, figyelembe véve a környezeti akadályokat és a mozgásukra vonatkozó korlátozásokat. Az algoritmus alkalmazása növeli a robotok munkavégzésének hatékonyságát, csökkenti az ütközések és az útvonaltervezési hibák kockázatát, ami kulcsfontosságú az ipari automatizáció és az autonóm járművek területén.

Az úttervező és útkereső algoritmusok folyamatos fejlődése és innovációja lehetővé teszi a hatékonyabb, gyorsabb és pontosabb útvonaltervezést a modern technológiai és társadalmi kihívásokra válaszul. A kutatók és fejlesztők új algoritmusokat és megközelítéseket dolgoznak ki, hogy javítsák ezeket a technikákat, adaptálva őket a változó igényekhez, mint az autonóm járművek útvonaltervezése, a városi mobilitási tervezés és a nagy adatkészletekkel dolgozó hálózati analízis.

Az A* (A-csillag) [5] algoritmus egy hatékony és teljes keresési algoritmus gráfokban vagy térképeken való útvonaltervezésre, amelyet gyakran használnak számítógépes játékokban és robotika területén az optimális útvonal megtalálására a kezdőpont és a célállapot között. Az algoritmus Peter Hart, Nils Nilsson és Bertram Raphael munkáján alapul 1968-ból, és a "legjobbat először" keresési stratégiát alkalmazza, kombinálva a Dijkstra algoritmus és a mohó keresés (greedy search) legjobb tulajdonságait.

Az algoritmus két fő komponensre támaszkodik: a g(n) és a h(n) függvényekre, ahol n egy gráfcsomópont. A g(n) a kezdőponttól n-ig vezető útvonal tényleges költségét jelenti, míg h(n) egy heurisztikus becslés az n csomóponttól a célállapotig vezető útvonal költségére. Az h(n) függvény lehetővé teszi az algoritmus számára, hogy prioritást adjon azoknak a csomópontoknak, amelyek valószínűleg a célhoz vezetnek, ezzel csökkentve a szükséges lépések számát és növelve a hatékonyságot. Az algoritmus célja, hogy minimalizálja a f(n) = g(n) + h(n) függvény értékét, ahol f(n) az adott csomópontban a teljes költség becslése.

Az előnye, hogy optimális és teljes, ami azt jelenti, hogy ha létezik megoldás, az algoritmus megtalálja az optimális megoldást. A heurisztika megválasztása kritikus: ha az h(n) sosem becsüli túl a tényleges költséget a célig (h(n) konzervatív), az algoritmus garantálja az optimális útvonal megtalálását. Az ideális heurisztika 'admisszibilis', vagyis soha nem túlbecsüli a valós költséget a célig, és 'monoton', ami biztosítja, hogy az f(n) érték sosem csökken az útvonal mentén.

Implementációjában a nyitott halmaz (open set) tartalmazza azokat a csomópontokat, amelyek még vizsgálatra várnak, míg a zárt halmaz (closed set) azokat a csomópontokat, amelyek már ki lettek értékelve. Az algoritmus iteratívan választja ki a nyitott halmazból azt a csomópontot, amelynek f(n) értéke a legalacsonyabb, kiértékeli a szomszédait, és frissíti az útvonal költségét és a nyitott halmazt, amíg el nem éri a célcsomópontot.

Alkalmazásának sikere nagyban függ a heurisztika pontos és hatékony megválasztásától, mivel ez befolyásolja az algoritmus teljesítményét és a keresési tér méretét. Az algoritmus adaptálható különböző problémákhoz, például térképes navigációhoz, mesterséges intelligencia alkalmazásokhoz vagy többdimenziós útvonaltervezési feladatokhoz is. Az A* algoritmus egyik kulcseleme a heurisztika alkalmazása, amely egy becslési módszer arra, hogy milyen költséggel érhető el a cél a jelenlegi állapotból. A heurisztikus értékek segítenek az algoritmusnak abban, hogy prioritást adjon az ígéretesebb útvonalak felderítésére, ezzel gyorsítva az optimális megoldás megtalálását. Különösen a Manhattan-távolság, amelyet gyakran várostérkép vagy taxisáv távolságnak is neveznek, egy olyan heurisztikus eszköz, amelyet az A* algoritmusban használnak az optimális útvonaltervezéshez gráfokon vagy térképeken. Ez a heurisztika különösen alkalmas olyan térképekhez, amelyek rácsokra, mint például városi utcarácsokra, vannak osztva, ahol csak észak-déli vagy kelet-nyugati irányban lehet haladni, tehát a diagonális áthaladás nem megengedett.

Kiszámításának képlete egyszerű: az a távolság, amelyet egy pontnak meg kell tennie a rács mentén, hogy elérjen egy másik pontba, anélkül, hogy átlósan haladna. Formálisan, ha d a Manhattan-távolság, x_1 és x_2 a két pont x koordinátái, valamint y_1 és y_2 a két pont y koordinátái, akkor

$$d = |x_1 - x_2| + |y_1 - y_2|$$

Így megadja a két pont közötti abszolút vízszintes és függőleges távolságok összegét. Az algoritmus kontextusában a Manhattan-távolság tökéletes heurisztikaként szolgál, amikor a mozgás csak a rácsok mentén lehetséges, mivel pontosan tükrözi a valóságos költséget, amennyiben nem áll rendelkezésre rövidebb, diagonális útvonal. Ez teszi a Manhattan-távolságot 'admisszibilis' heurisztikává az A* algoritmus számára, ami azt jelenti, hogy soha nem becsüli túl a valós költséget a kezdőponttól a célállapotig vezető legrövidebb útvonalon.

Továbbá, a Manhattan-távolság használata jelentős mértékben csökkentheti a keresési teret az A* algoritmusban, mivel az algoritmus prioritást ad azoknak a csomópontoknak, amelyek közelebb vannak a célhoz ezen a metrikán keresztül mérve. Ezáltal az algoritmus gyorsabban találhatja meg az optimális útvonalat, csökkentve a szükséges számítási erőforrásokat és időt.

2.6. Projektépítés Gradle segítségével

A Gradle egy nyílt forráskódú automatizálási eszköz, amely népszerű a Java, Kotlin és más nyelveken írt projektjeinek építéséhez és kezeléséhez. A Gradle segítségével automatizálható a projekt fordítása, a függőségek kezelése, a tesztelés, a csomagolás és még sok más, ami jelentősen megkönnyíti és felgyorsítja a fejlesztési folyamatot.

A fő előnye a nagy teljesítményű, rugalmas automatizálási eszközként való használata, amely lehetővé teszi a fejlesztési ciklusok jelentős leegyszerűsítését és gyorsítását. A build konfigurációk írhatók Groovy vagy Kotlin DSL (Domain Specific Language) segítségével, amelyek mindkettő hatékony és olvasható módon kínálja a konfigurációs logikát. A Kotlin alapú Gradle scriptek, azaz a *build.gradle.kts fájlok*, lehetővé teszik a projekt build konfigurációjának írását Kotlinban, ami javítja a konfiguráció típusbiztonságát és jobban integrálódik a Kotlin projektjeinkkel.

- Plugins: Itt adhatjuk hozzá a projektünk számára szükséges bővítményeket, mint például az Android alkalmazásokhoz szükséges com.android.application, a Kotlin támogatást biztosító org.jetbrains.kotlin.android és még megannyi bővítményt.
- Android konfiguráció: A namespace,az alkalmazás azonosítója (applicationId), a verziókód, verzió név, tesztelési eszközök és a vektorgrafikák kezelése kapcsán adhatunk meg beállításokat. Továbbá, konfigurálhatjuk a build típusokat, aktiválhatjuk a minify funkciót, és hozzáadhatjuk a ProGuard szabályokat a kód optimalizálásához és obfuszkálásához.
- Compile és Kotlin opciók: Lehetővé teszi a forrás- és célnyelv verziójának beállítását, illetve a Kotlin specifikus konfigurációkat, mint például a JVM target verzió.
- Build features: Megadhatjuk, hogy melyik modern Android fejlesztési eszközöket szeretnénk használni, mint például a Jetpack Compose.
- Dependencies: Itt adhatók hozzá a projekt függőségei, beleértve a külső könyvtárakat, modulokat, teszt keretrendszereket és minden más szükséges komponenst, amelyre a projektünk épít.

2.7. XML használata a felhasználói felülethez

Az XML (Extensible Markup Language) egy általános célú jelölőnyelv, amely adatok strukturált reprezentációjára és szállítására szolgál. Az XML-t azért tervezték, hogy egyszerű, olvasható formátumot biztosítson mind az emberek, mind a gépek számára. Mivel rugalmas és testreszabható, széles körben használják különböző alkalmazásokban és rendszerekben az adatok leírására és cseréjére.

Az Android alkalmazásfejlesztés terén az XML különösen fontos szerepet tölt be a felhasználói felület (UI) megvalósításában. Az Android Studio, amely az Android alkalmazások hivatalos fejlesztői környezete, az XML-t használja a layout fájlok meghatározásához. Ezek a fájlok írják le, hogy az alkalmazás különböző elemei, mint például gombok, szövegmezők és képek, hogyan jelenjenek meg és rendeződjenek el a képernyőn.

- Olvashatóság és könnyű szerkeszthetőség: Az XML egyszerű és ember által olvasható formátuma miatt a fejlesztők könnyen megérthetik és módosíthatják a layout struktúráját, még szöveges szerkesztőben is.–
- Deklaratív nyelv: Az XML lehetővé teszi a felületek deklaratív leírását, vagyis a fejlesztők egyszerűen leírhatják, milyen UI elemekre van szükségük és hogyan kell azokat elrendezni, anélkül, hogy a megjelenítésük módját programkódban kellene megvalósítaniuk.
- Újrafelhasználhatóság és modularitás: Az XML layoutok segítségével egyszerűen újrafelhasználható és moduláris UI komponenseket hozhatunk létre, amelyeket különböző Activity-kben vagy Fragmentekben is felhasználhatunk.
- Szeparáció: Az XML használata lehetővé teszi a felhasználói felület elkülönítését a háttérbeli alkalmazáslogikától, ami javítja a kód átláthatóságát és karbantarthatóságát.

Bár a Kotlin az Android alkalmazások programozási nyelve, az XML fájlok mégis elengedhetetlenek a felhasználói felületek definiálásához. A Kotlin kód interakcióba lép az XML-ben meghatározott UI elemekkel, így programozottan kezeljül a felhasználói felületet, például eseménykezelőket rendeljenek hozzájuk, vagy dinamikusan frissítsük a felületi elemek tartalmát.

3. fejezet

Az alkalmazás megtervezése

A technológiai alapok tekintetében a dolgozat a Kotlin programozási nyelvet alkalmazza, amely mára az Android platform fejlesztésének sztenderdjévé vált, felváltva az eredetileg használt Javát. A Kotlin számos előnnyel rendelkezik, mint például a kifejezőképesség, a biztonsági jellemzők, valamint az, hogy lehetővé teszi az egyszerűbb és olvashatóbb kód írását. E technológiai választás különösen jelentőséget nyer, figyelembe véve, hogy a globális mobilpiac körülbelül ~71%-át [21] az Android eszközök alkotják, így az alkalmazás széleskörű elérhetőségét és használhatóságát garantálja. A piackutatás szempontjából fontos megvizsgálni az Android alapú alkalmazások piacát, mivel a mobil eszközök túlnyomó többsége ezen az operációs rendszeren működik. A hasonló alkalmazások elemzése és összehasonlítása kulcsfontosságú a piaci rés meghatározása és az új alkalmazás potenciális előnyeinek felmérése szempontjából. Az Android platform dominanciája a mobilpiacon lehetővé teszi, hogy az új alkalmazások széles körben elérhetőek és használhatóak legyenek a globális felhasználói bázis számára. Az Android Studio, amely az IntelliJ IDEA-ra épül, nem csupán egy fejlesztői környezet; ez a platform lehetővé teszi az Android alapú alkalmazások intuitív és hatékony tervezését és implementálását. Az integrált UI designer eszközök révén sikerült egy felhasználóbarát felületet létrehoznom, ami kulcsfontosságú volt az alkalmazás sikerességéhez. Az Android Studio által nyújtott környezet támogatja a fejlesztőket a kód írásától kezdve a tesztelésen át egészen a végleges alkalmazás kiadásáig.

A nyelvet kifejezetten úgy tervezték, hogy tömör, mégis kifejező legyen, lehetővé téve a fejlesztők számára, hogy kevesebb kódsorral, de több funkciót valósítsanak meg. A Kotlin az Android Studio hivatalos támogatásával az Android-fejlesztés egyik vezető nyelvévé vált. A nyelv elsődleges előnyei közé tartozik a null-biztonság, ami a fejlesztők számára lehetővé teszi a null értékű hibák kezelését fordítási időben, csökkentve a futásidejű NullPointerException-ok előfordulásának esélyét. Ez a jellemző különösen fontos a kontextuskezelés szempontjából, ahol az objektumok null állapota gyakran vezet hibákhoz.

Az osztály központi elemei közé tartozik a helymeghatározás és térképkezelés. A Kotlin null-biztonsági jellemzői (a null értékű hibák kezelése fordítási időben) lehetővé teszik, hogy az alkalmazás biztonságosan kezelje a potenciálisan null értékeket, például az engedélykéréseknél:

Programkód 3.1. Engedély ellenőrzése és kérés

```
if (ContextCompat.checkSelfPermission(this, Manifest.permission.
   ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
    ActivityCompat.requestPermissions(this, arrayOf(Manifest.
       permission.ACCESS_FINE_LOCATION), REQUEST_LOCATION)
}
```

Ez a kód biztonságosan ellenőrzi, hogy az alkalmazásnak megvannak-e a szükséges helymeghatározási engedélyei, és ha nem, engedélyt kér a felhasználótól, minimalizálva ezzel a futásidejű engedély hibák kockázatát. A Kotlin további előnyei közé tartozik a kivételkezelés és a lambda kifejezések használata, amelyek egyszerűsítik az eseménykezelők és callback függvények implementálását, javítva ezzel a kód olvashatóságát és karbantarthatóságát. A lambda kifejezések különösen hasznosak az Android UI események kezelésében, lehetővé téve a kódbázis egyszerűsítését és a redundancia csökkentését. Például a térkép zoom szintjének figyelésekor a Kotlin lehetővé teszi, hogy tömör és olvasható kódot írjunk:

Programkód 3.2. Térkép zoom szintjének figyelése

```
mMap.addMapListener(object : MapListener {
    override fun onZoom(event: ZoomEvent?): Boolean {
        val zoomLevel = mMap.zoomLevelDouble
        updateSaveButtonState(button, zoomLevel, selectedMode)
        return true
    }
})
```

A Kotlin mutableMapOf és mutableListOf funkciói segítségével a *Graph* osztály dinamikusan kezeli az élszomszédsági listáját, lehetővé téve a gráf szerkezetének hatékony módosítását:

Programkód 3.3. Graph osztály és élek hozzáadása

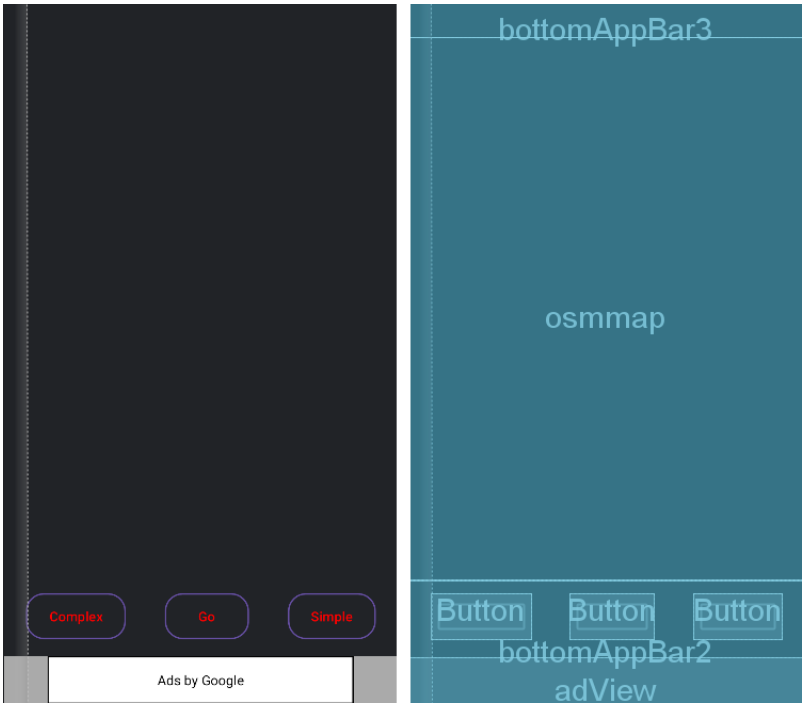
```
class Graph {
    val adjacencyList = mutableMapOf<Vertex, MutableList<Edge
       >>()

    fun addEdge(start: Vertex, end: Vertex, weight: Double) {
        adjacencyList.computeIfAbsent(start) { mutableListOf()
           }.add(Edge(end, weight))
    }
}
```

3.1. Alkalmazott technológiák

Az alkalmazás felületének tervezésekor fontos szempont volt, hogy a felhasználó gyorsan, egyértelműen és hatékonyan tudjon navigálni az alkalmazáson belül, valamint könnyedén hozzáférjen az összes szükséges funkcióhoz. Az alkalmazás alsó részén található navigációs sáv három gombbal teszi lehetővé a felhasználó számára, hogy könnyedén váltogasson az aktuális nézetek között. A gombok kialakítása egy intuitív visszajelző rendszert alkot, ahol a zöld és piros színek használata révén az alkalmazás azonnali visszajelzést nyújt a felhasználónak a gombok állapotáról, például arról, hogy egy adott funkció éppen elérhető-e vagy sem. Ez a megközelítés segít abban, hogy a felhasználói interakciók zökkenőmentesek és könnyen érthetőek legyenek. Minden nézetnek megvan a saját három gombja, ami a nézet funkcióit szolgáltatja.

A felhasználói felület megvalósításának részletei az XML layout [4] fájlokban találhatók, amelyek leírják az egyes UI elemek struktúráját és viselkedését. Ezek a fájlok biztosítják a felületi elemek elrendezésének flexibilitását, valamint a különböző eszközökön és képernyőméretekben való megfelelő megjelenítést. A részletes leírást és az XML layoutok elemzését a dokumentáció külön szakaszában található: 2.7 - 11. oldal.

Az itt bemutatott képen az Android Studio beépített tervezői környezetében készített nézet előnézete látható. A felületet úgy alakítottam ki, hogy a térkép szinte teljes képernyőt kitöltve helyezkedik el, és az y tengelyen legalul foglal helyet. Ezzel a megoldással a képernyő felső és alsó sávjai csak részben takarják, illetve levágják a térképet. Ez a kialakítás segít megoldani azt a problémát, hogy a térkép forgatása során a tile-ok furcsán eltűnnek vagy hiányosan töltődnek be, így a felhasználói élmény jelentősen javul, mivel a térképnézet zökkenőmentesebb és teljesebb marad.



3.1. ábra: A megtervezet előnézet.

A modern alkalmazásfejlesztés során egyre több fejlesztő fordul különböző külső könyvtárak és szolgáltatások felé annak érdekében, hogy gyorsabban és hatékonyabban valósíthassanak meg összetett funkciókat, valamint monetizálási lehetőségeket építhessenek be alkalmazásaikba. Az általam fejlesztett alkalmazás esetében is fontos, hogy éljek ezekkel a lehetőségekkel, amely nemcsak a fejlesztési folyamatot teszi gördülékenyebbé, hanem a projekt fenntarthatóságát is elősegíti a bevételszerzés lehetőségével.

A Firebase egy átfogó mobil- és webfejlesztési platform a Google-tól, amit széleskörű eszközök és szolgáltatások széles palettáját biztosítják a fejlesztőknek. Ezek közül az egyik legfontosabb szolgáltatás, amit az alkalmazásomban használok, a Firebase Analytics. Ezzel a szolgáltatással részletesen monitorozom, hogyan használják a felhasználók az alkalmazásomat. Ez kulcsfontosságú betekintést nyújt a felhasználói magatartásba, beleértve az alkalmazás használatának gyakoriságát, a felhasználók földrajzi elhelyezkedését, valamint az egyes funkciók népszerűségét.

A Firebase Analytics használatával jobban megértem felhasználóim igényeit és preferenciáit, ami alapvető fontosságú az alkalmazás folyamatos fejlesztése és a felhasználói élmény javítása szempontjából. Az adatok elemzésével képes vagyok azonosítani a felhasználói viselkedési mintákat, felismerni a potenciális problémákat, és informált döntéseket hozni az alkalmazás továbbfejlesztésének irányáról.

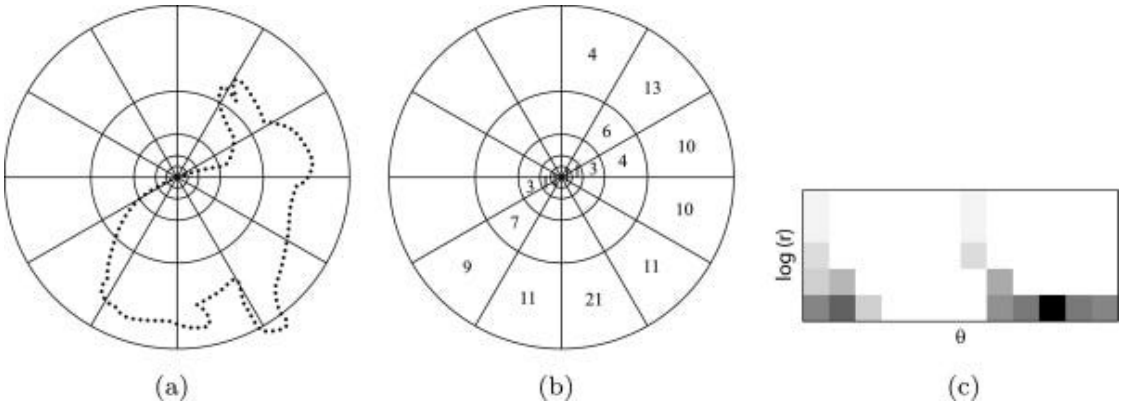
Ennek az integrálása rendkívül egyszerű volt, és nem igényelt komplex kódolást, így gyorsan és hatékonyan kezdhettem el használni ezt a szolgáltatást az alkalmazásomban. Ez az eszköz nemcsak, hogy segít a felhasználói élmény finomhangolásában, de a marketingstratégia kialakításában és az alkalmazás sikerének mérésében is közrejátszik.

Az AdMob a Google egyik vezető mobil hirdetési platformja, amely lehetővé teszi a fejlesztők számára, hogy alkalmazásaikban megjelenített reklámokból származó bevételeket realizáljanak. Az AdMob integrálása egyszerű módja annak, hogy monetizáljuk az alkalmazást anélkül, hogy negatívan befolyásolnánk a felhasználói élményt.

Alkalmazásomban az AdMob szolgáltatásait arra használom, hogy célzott és releváns hirdetéseket jelenítsek meg a felhasználóknak. Ezek a hirdetések gondosan vannak kiválasztva és elhelyezve, hogy minimalizálják a felhasználói interakció zavarását, miközben értékes bevételi forrást jelentenek a projekt számára.

A projekt során a Shape Context eljárás alkalmazása vált a központi elemmé az útvonalak és grafikai formák térképekhez való illesztésének és optimalizálásának folyamatában. Az eljárás integrálása érdekében szükségessé vált a gráfok adatainak és struktúrájának módosítása, ami lehetőséget biztosított a modern programozási paradigmák, mint az interfész és absztrakt osztályok alkalmazására. Ez a megközelítés nem csak a kód modularitását és újrafelhasználhatóságát javította, hanem a rendszer bővíthetőségét és karbantarthatóságát is elősegítette.

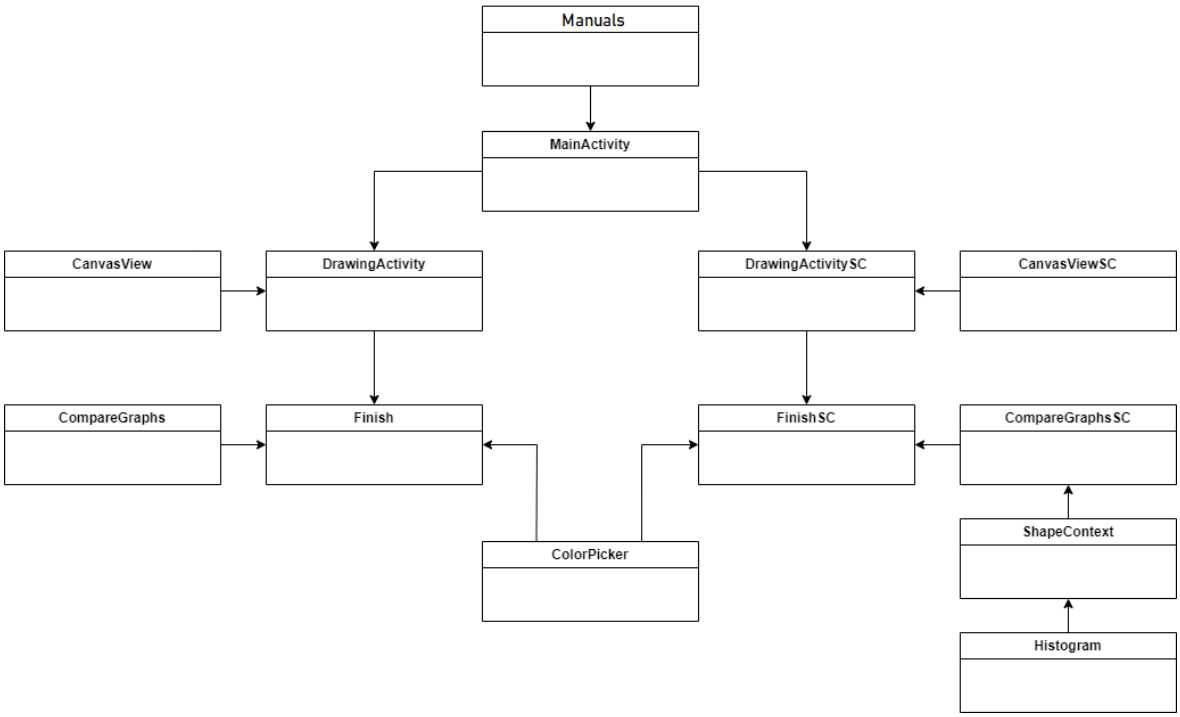
A Shape Context módszer bevezethetősége érdekében a gráfok kezelésére szolgáló osztályokat úgy terveztem meg, hogy támogassák a log-polár koordinátarendszeren alapuló reprezentációt, és a pontok közötti viszonyokat histogramok segítségével kvantitatívan reprezentálják.



3.2. ábra: A Shape Context eljárás illusztrációja.

Az alkalmazás fejlesztése során az OpenStreetMap térképi adatbázisát használtam fel, amely betekintést nyújtott a térinformatika világába. [19] Az OpenStreetMap hosszúsági és szélességi koordinátákat biztosít, amelyeket az alkalmazás egy beépített konverziós metódus segítségével kétdimenziós koordinátákká alakít át. Ez a konverzió tette lehetővé számomra, hogy a térképi adatokat közvetlenül felhasználjam a grafikai elemek illesztésekor és optimalizálásakor. Az adatok lekérésére az Overpass API-t használtam, amely lehetővé teszi a látható térképrészlet adatok lekérését egyetlen query hívással. [9] Az Overpass API használata kulcsfontosságú volt abban, hogy az alkalmazás dinamikusan alkalmazkodjon a felhasználó által megjelenített térképrészlethez, és csak a releváns adatokat töltsük le és dolgozzuk fel.

Az alkalmazás projektstruktúrájának megtervezésekor figyelmet fordítottam a kód rendezettségére és modularitására, hogy a fejlesztési folyamat során könnyen kezelhető és bővíthető legyen a forráskód. A középpontban a *com.wardanger3.gps_app* könyvtár áll, amely tartalmazza az alkalmazás összes releváns forráskódját, biztosítva ezzel a fejlesztési munka egységes kiindulási pontját. Ez a package szolgál a nagyobb osztályok tárolására, amelyek az alkalmazás magját alkotják. Ezen belül a logikailag összetartozó funkciókat külön csoportokba, úgynevezett alcsomagokba szerveztem, hogy a projektstruktúra átlátható és könnyen navigálható maradjon. A kód újrafelhasználhatóságának és a fejlesztési folyamat hatékonyságának növelése érdekében a közös jellemzőkkel rendelkező osztályokat részben kiszerveztem az *abstract_classes* és az *interfaces* nevezetű csomagokba. Az alábbi osztálydiagram bemutatja az alkalmazás architektúráját és a komponensek közötti kapcsolatokat, amelyek hozzájárultak a rendszer tervezéséhez és fejlesztéséhez.



3.3. ábra: Az osztályok kapcsolatai.

4. fejezet

Alkalmazás fejlesztése

Ez a fejezet részletesen bemutatja az alkalmazás felépítését, osztályról osztályra haladva. Itt kerülnek ismertetésre a projekt során alkalmazott algoritmusok, azok működési elvei, valamint konkrét implementációs részletei. Emellett speciális figyelmet fordítok a használt algoritmusok leírására és azok alkalmazásának indoklására, lehetővé téve ezzel a szoftver működési logikájának mélyebb megértését.

4.1. Mobilalkalmazás fejlesztés alapjai

Az Android alkalmazások struktúrájának és működésének megértése során szükséges az AndroidManifest.xml fájl részletes elemzése. Ez a konfigurációs fájl az alkalmazás minden komponensét regisztrálja, beleértve a felhasznált engedélyeket, az alkalmazás szintjén beállított tulajdonságokat, valamint az alkalmazás belépési pontjaként szolgáló Activity-ket. Az engedélyek, mint például a ACCESS_COARSE_LOCATION, ACCESS_FINE_LOCATION, és INTERNET, kulcsfontosságúak az alkalmazás által nyújtott funkcionalitás és a felhasználói adatok védelme szempontjából. Ezek meghatározzák, hogy milyen rendszerszintű erőforrásokhoz férhet hozzá az alkalmazás, biztosítva ezzel a szükséges adatokat a felhasználói élmény javítása érdekében, miközben fenntartják a biztonságot és a magánélet védelmét.

Az alkalmazás belépési pontjának meghatározása az AndroidManifest.xml fájlban történik a <activity> elemek és az ezekhez tartozó <intent-filter> részletezésével, ahol a MainActivity szerepel az android.intent.action.MAIN címkével, illetve az android.intent.category.LAUNCHER kategóriával. Ez jelzi a rendszer számára, hogy ez az Activity az alkalmazás indítási pontja, ami elsőként jelenik meg a felhasználónak az alkalmazás elindításakor.

A kotlin programozái nyelv szoros integrációja az Android platformmal kiterjedt támogatást nyújt az Android-specifikus funkciókhoz, mint például a kontextuskezelés. A kontextus az alkalmazás jelenlegi állapotára vonatkozó információkat tartalmazza, és elengedhetetlen a hatékony Android-alkalmazás-fejlesztéshez. A Kotlin nyelv lehetővé teszi a fejlesztők számára, hogy egyszerűen kezeljék a kontextust, kihasználva a nyelv biztonsági jellemzőit, mint például a biztonságos hívásokat (?.) és az elvis operátort (?:), amelyek segítenek elkerülni a null értékek okozta problémákat.

A MainActivity osztály a Kotlin nyelv kontextuskezelési képességeit kihasználva biztosítja az alkalmazás alapvető funkcionalitását. A kontextus itt az alkalmazás vagy az Activity állapotának kezelésére szolgál, lehetővé téve az erőforrások, mint például adatbázisok, preferenciák és UI elemek hozzáférését. A Kotlin null-biztonsági jellemzői segítségével a MainActivity képes biztonságosan kezelni a kontextust, minimalizálva az alkalmazás összeomlásának kockázatát.

Overpass API és Adatgyűjtés

Az első kihívás, amellyel szembesültem, az adatok előállítása volt. Kezdetben próbálkoztam gépi látás alkalmazásával egy képről az úthálózat kinyerésére, ami nem bizonyult eléggé megbízhatónak. Számos kutatás és próbálkozás után bukkantam rá az Overpass API-ra, amely végül ideális megoldásnak bizonyult az adatgyűjtési igényeimre. Használata kritikus fontosságú az alkalmazásom számára, amelynek fő célja a térképadatokból származó úthálózat vizualizálása és integrálása. Az OpenStreetMap (OSM) rendkívül gazdag forrás a globális térképadatok tekintetében, ám a nyers adatok közvetlen felhasználása és feldolgozása jelentős kihívást jelentett. Az Overpass API, mint egy erőteljes lekérdezési eszköz, lehetővé tette számomra, hogy pontosan és hatékonyan kérjem le azokat a térképi adatokat, amelyekre a projekt során szükségem van.Egyidőben az API elengedhetetlen a releváns térképi adatok - különösen az úthálózatra vonatkozó adatok - lekérésében. Az alábbi lekérdezés egy konkrét példája annak, hogyan kértem le azokat az útvonalakat, amelyek egy adott térképi kivágáshoz tartoznak. A lekérdezés a **highway** kulcsszó megadásával történt, így csak a releváns és olyan adatokat kaptam vissza, amellyeket képes voltam egyetlen nagy és bejárható gráffá alakítani.

Az alkalmazás használatának első lépése az, hogy a felhasználó a főképernyőn kiválasztja valamelyik illesztési módot. Ezután a felhasználó a megfelelő nagyítási szintre közelít, amíg a középen lévő **go** gomb zölddé nem válik. Ekkor indítható el a térképi adatok lekérdezése az Overpass API segítségével. A lekérdezés eredményeként visszakapott JSON formátumú adatok feldolgozása magában foglalja a visszakapott útvonalak és csomópontok átalakítását a belső gráf reprezentációba. Ez lehetővé teszi számomra, hogy egy olyan gráfot hozzak létre, amely pontosan tükrözi a valós világ úthálózatát a felhasználó által kiválasztott területen.

Programkód 4.1. Overpass API lekérdezés

```
val overpassQuery = """
    [out:json];
    (
        way["highway"](${boundingBox.latSouth},${boundingBox.
           lonWest},${boundingBox.latNorth},${boundingBox.lonEast
           });
    );
    out body;
    >;
    out geom;
    """.trimIndent()
```

4.1.1. Gráfelmélet és alkalmazása

A gráfelmélet [3] alkalmazása az alkalmazásomban alapvető jelentőséggel bír, mivel ez a matematikai keretrendszer biztosítja az úthálózatok modellezésének és elemzésének eszközeit. Ebben a részben bemutatom, hogyan definiáljuk és használjuk a gráfokat az alkalmazásban, valamint hogy a gráfelmélet hogyan segít az útvonaltervezésben és optimalizációban. Az alkalmazásban használt gráfstruktúrák jelentősége abban rejlik, hogy képesek leképezni a valóságban lévő úthálózatok bonyolult szerkezetét és lehetővé teszik azok hatékony feldolgozását. A gráfok segítségével képesek vagyunk szimulálni az útvonalakat, meghatározni a legrövidebb vagy legoptimálisabb utakat, és az illesztési pontokat az úthálózathoz. Az útvonaltervezés és optimalizáció során a gráfelméletet alkalmazom az útvonalak és illesztési pontok meghatározására. Az útvonaltervezés során az A* [5] algoritmust vagy más gráfkeresési algoritmusokat vetek be, amelyek lehetővé teszik a legrövidebb vagy költséghatékony útvonalak megtalálását a gráfban. Az algoritmusok figyelembe veszik az élek súlyát – amely az út hosszát, költségét vagy egyéb tényezőket jelképezheti –, így megtalálva az optimális útvonalat a két pont között. Ezek által a Shape Context képes lesz hatékonyan kihasznélni a gráfok strukturális jellemzőit, lehetővé téve az illesztési pontok pontos és hatékony meghatározását, illetve összevetését.

Az alkalmazás fejlesztése során egyik kulcsfontosságú kihívást a geokoordináták [7] és a kétdimenziós (2D) képernyő koordináták közötti konverzió jelentette. A földrajzi koordinátarendszer (latitűd, longitud) és a képernyő koordinátarendszer (x, y) közötti átalakítás alapvető ahhoz, hogy a valós világban lévő helyeket és útvonalakat pontosan tudjuk reprezentálni és manipulálni a digitális térképen. A földrajzi koordináták (latitude (szélesség) és longitud (hosszúság)) a Föld felszínén található pontokat írják le. A képernyő koordináták (x, y) pedig a digitális térkép egy adott pontjának képernyőn való helyzetét határozzák meg. A két koordinátarendszer közötti konverzió lehetővé teszi számunkra, hogy a valós világban lévő helyeket megjeleníthessük és kezelhessük a digitális térképen. Az OpenStreetMap által alkalmazott képfeldolgozási technika a Web Mercator [10], amely földrajzi koordinátákat képernyő koordinátákká konvertál. Ez a módszer különösen jól illeszkedik a webes térképszolgáltatásokhoz, mivel a Föld gömb alakját képes síkra vetíteni úgy, hogy a navigáció és a helymeghatározás könnyen kezelhető maradjon. A konverzió matematikai alapjait trigonometrikus és logaritmikus számítások képezik, amelyek a Föld gömbölyű formáját és a képernyő méreteit veszik figyelembe.

Nehézségekbe ütköztem a geokoordináták átalakításánál, mivel kezdetben saját függvényt szerettem volna írni az átalakítási folyamat megvalósításához. Azonban az első próbálkozások nem hozták meg a várt eredményeket, a képre vetített gráfok nem tükrözték pontosan az úthálózat valós elhelyezkedését. Ez a kihívás arra késztetett, hogy részletesebben tanulmányozzam a különböző átalakítási technikákat, mint például a Mercator- és a lineáris átalakítást. Végül az OpenStreetMap beépített átalakítási funkcióját választottam, amely a Web Mercator módszert használja, mivel így kaptam a legpontosabb eredményeket. [10] Ez a döntés lehetővé tette számomra, hogy pontosan és hatékonyan alakítsam át a geokoordinátákat képernyő koordinátákká. A Web Mercator technika speciális jellemzői, mint az egyenletes skálázás és a navigációs pontosság, ideálissá tették e módszer alkalmazását az alkalmazásomban, biztosítva ezzel a térképi adatok pontos megjelenítését és kezelését.

A matematikai kifejezése a következő:

$$x = R \cdot \lambda$$
$$y = R \cdot \ln\left(\tan\left(\frac{\pi}{4} + \frac{\phi}{2}\right)\right)$$

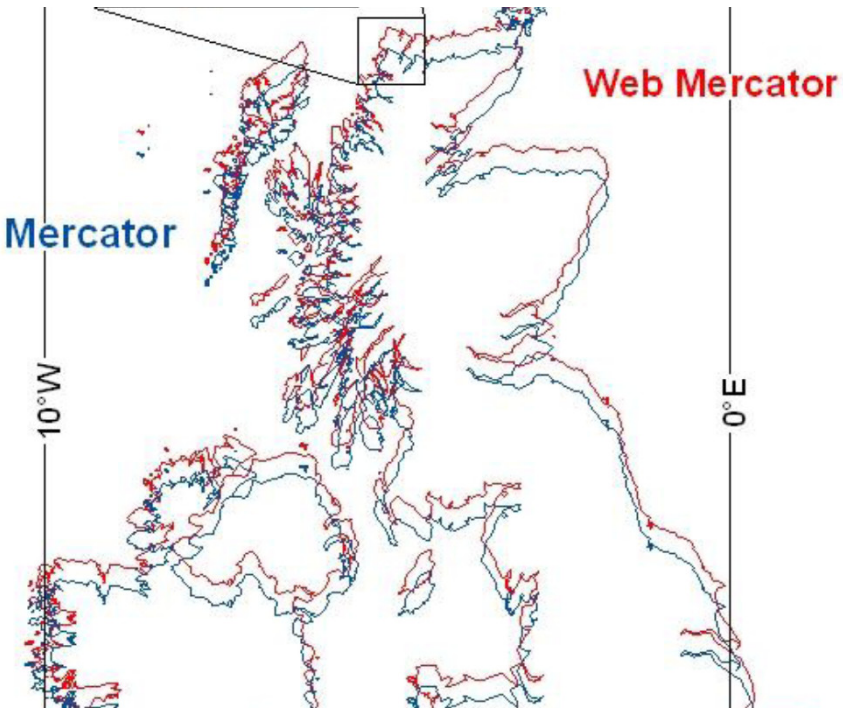
ahol:

- λ a hosszúsági koordináta radiánban,
- ϕ a szélességi koordináta radiánban,
- R a térkép sugara, általában $R = 6378137.0$ m (a WGS84 ellipszoid sugara),
- x és y a képernyő koordináták a digitális térképen.

Fontos megjegyezni, hogy ez a projekció torzítja a területeket, különösen a magas szélességi körökön. A torzítás mértékét a következő egyenlettel lehet kifejezni:

$$\text{Területi torzítás} = \sec(\phi)$$

ahol ϕ a szélességi koordináta radiánban. Ez azt jelenti, hogy minél távolabb vagyunk az Egyenlítőtől, annál nagyobb a területi torzítás.



4.1. ábra: Merchator és a Web Merchator projekciók torzítása egymáshoz képest. [20]

4.2. Rajzolási folyamatok és aszinkron programozás

Ezen osztályok a projekt két komponensét képviselik, amelyek közvetlenül kapcsolódnak a felhasználói interakciókhoz, a grafikus elemek kezeléséhez, illetve a felhasználó által rajzolt gráfok létrehozásában. Ezek az osztályok továbbá biztosítják a rajzolási folyamat alapvető funkcionalitásait, mint a rajzolás, törlés és mentés. Az aszinkron programozás kulcsfontosságú eleme a modern alkalmazásoknak, különösen olyan műveletek esetében, amelyek hosszabb ideig tartanak, és nem akarjuk, hogy blokkolják az UI szálat. A Kotlin korutinjai egy elegáns megoldást kínálnak erre a problémára, lehetővé téve számunkra, hogy aszinkron kódot írjunk szinkron módon, ami javítja a kód olvashatóságát és karbantarthatóságát.

A *DrawingActivity* és *DrawingActivitySC* osztályokban a korutinokat arra használom, hogy a háttérszálon végeztessem el a rajzolási adatok feldolgozását és a gráfok illesztését, így a felhasználói felület válaszképes marad. Például, amikor a felhasználó befejezi a rajzolást és menteni szeretné az eredményt, egy korutin indít egy aszinkron folyamatot a gráf adatok feldolgozására és optimalizálására, anélkül, hogy befolyásolná az alkalmazás válaszképességét. A hosszabb futású feladatok, mint például a gráf illesztése és a képek kirajzolása, blokkolhatják a felhasználói felület szálat, ha nem kezeljük őket megfelelően. A Coroutine-ok használata lehetővé teszi ezeknek a műveleteknek az aszinkron és nem blokkoló végrehajtását. Az Dispatchers.IO kontextusban futtatott műveletek, mint az adatfeldolgozás vagy a képfeldolgozás, nem zavarják meg a felhasználói felület sima működését, így javítva az alkalmazás válaszkészségét és az illesztési idő sem elhanyagolható.

A coroutine-ok előnyeit kihasználva az alkalmazásom kritikus részeinél, mint például az adatfeldolgozás és a képfeldolgozás, jelentős teljesítménynövekedést értem el. Ezek a műveletek a Dispatchers.IO kontextusban történő futtatásával aszinkron és nem blokkoló módon zajlanak, így nem befolyásolják negatívan a felhasználói felület működését. Ez a megközelítés javítja az alkalmazás válaszkészségét, mivel a UI szál szabad marad az interakcióra az adatfeldolgozás ideje alatt is.

A párhuzamosításnak köszönhetően az adatok feldolgozása és az illesztési idő javult. Egy konkrét mérésem során szemléletes példát találtam a párhuzamosítás hatékonyságára: míg a szekvenciális feldolgozás során az alkalmazás 5 perc alatt csupán 4%-os feldolgozási haladást ért el, addig a párhuzamosított folyamat több mint háromszoros sebességnövekedést mutatott, 14%-os átlagos illesztési aránnyal. Ez a jelentős teljesítménynövekedés nemcsak hogy gyorsította az adatfeldolgozást, hanem pozitívan befolyásolta az alkalmazás általános felhasználói élményét is.

Ezen adatok alapján világossá vált, hogy a coroutine-ok és a párhuzamosítás alkalmazása elengedhetetlen az olyan intenzív és nagy műveletek hatékony kezelésében, ahol a sebesség az első számú javítandó tényező.

Programkód 4.2. CoroutineScope használata, a háttérfolyamat indítása a mellkészálon

```
private val activityScope = CoroutineScope(Dispatchers.Main)

activityScope.launch(Dispatchers.Default) {
     Log.d(TAG, "Background task started")
     canvasViewSC.createGraphFromPathData()
     Log.d(TAG, "Graph creation completed")
 }
```

Az osztályokban a felhasználói interakciók kezelése elengedhetetlen tényező. A gombok és az érintőképernyős interakciók kezelése lehetővé teszi a felhasználók számára, hogy intuitív módon végezhessék el a rajzolást, törlést és a műveletek mentését, miközben a háttérben folyamatok zajlanak. Az eseménykezelők és callback függvények gondoskodnak arról, hogy a felhasználói műveletek azonnal feldolgozásra kerüljenek, és a felhasználói felületen megfelelő visszajelzést kapjanak.

Az Activity-k közötti kommunikáció a *DrawingActivity* és *DrawingActivitySC* osztályokban az Intent objektumok segítségével történik. Az Intentek lehetővé teszik számunkra, hogy adatokat továbbítsunk az egyik Activityből a másikba, és új Activity-ket indítsunk. Ez a mechanizmus kulcsfontosságú a felhasználói interakciók, mint például a rajzolási folyamat befejezése utáni eredmények megjelenítése vagy további szerkesztési lehetőségek elérésének kezelésében.

Programkód 4.3. A következő activity indítása

```
val intent = Intent(this@DrawingActivitySC,FinishSC::class.java)
startActivity(intent)
```

A modularitás figyelembevételével minden oldalt külön activitybe szerveztem ki. Ezáltal az activityk közötti navigáció nemcsak releváns, hanem rendkívül hasznos is, mivel lehetővé teszi a felhasználó számára, hogy intuitív módon navigáljon a különböző funkciók között, anélkül, hogy elvesztené a kontextust vagy az adatokat. Ez a megközelítés továbbá elősegíti az alkalmazás karbantarthatóságát és bővíthetőségét, mivel minden egyes activity külön modulként funkcionál, ami könnyebbé teszi új funkciók hozzáadását vagy meglévők módosítását anélkül, hogy az az egész alkalmazás architektúráját befolyásolná. Ezen felül, az alkalmazás tesztelési folyamatát is egyszerűsíti, mivel lehetővé teszi az egyes komponensek izolált tesztelését. Az egyes activityk közötti adatátvitel kezelése az Intent objektumok segítségével nemcsak hogy rugalmas megoldást biztosít, hanem a fejlesztők számára egy erőteljes eszközt is nyújt az adatok átadására és az alkalmazás állapotának kezelésére a különböző felhasználói interakciók során.

4.3. Grafikai interakciók és adatkezelés optimalizálása

Ezek az osztályok a *Canvas* API-t használják a rajzolási műveletek kezelésére, amely magában foglalja a felhasználó érintéseinek észlelését, a vonalak és formák rajzolását a képernyőn, valamint az elkészült rajzok digitális reprezentációjának tárolását. Miután a térképi adatok lekérdezése és feldolgozása megtörtént, a felhasználó átkerül a rajzoló oldalra, ahol elkészítheti alkotását. A *Canvas* osztály, mint az Android SDK része, a grafikai műveletek végrehajtását teszi lehetővé, biztosítva, hogy minden rajzolási művelet simán és pontosan kerüljön végrehajtásra, miközben dinamikusan kezeli a felhasználói bemeneteket. A *Canvas* osztály az Android SDK részét képező funkciókészlet, amely a rajzolási és grafikai műveletek végrehajtását teszi lehetővé. A Canvas API fontos funkciói közé tartozik:

- onDraw metódus: Itt valósul meg a rajzolási logika, ahol a *Canvas* példányának draw‘metódusait használva különböző grafikai elemeket rajzolunk.
- onTouchEvent metódus: Ez kezeli a felhasználói interakciókat, mint az érintés, húzás, ami lehetővé teszi a dinamikus rajzolást.
- Paint osztály: A rajzolási stílusok, mint a szín, vonalvastagság és töltés beállítására használjuk.
- Rajzolási primitívek: A Canvas API lehetővé teszi vonalak, körök, ívek és egyéb alakzatok egyszerű rajzolását.

Nem elhanyagolható, hogy írjak a gráfok felépítéséről sem, mert a gráf adatszerkezetek kritikus fontosságúak a rajzolt útvonalak és formák analízisében és kezelésében. A gráfok lehetővé teszik a csomópontok (pontok) és élek (vonalak közötti kapcsolatok) strukturált reprezentációját, ami elősegíti az útvonaltervezést, az útvonalak és formák optimalizálását. A gráf alapú megközelítés rugalmasságot és bővíthetőséget biztosít az alkalmazás számára, lehetővé téve különböző geometriai manipulációk és analízisek végrehajtását. Mindezek előtt finomítani kell az eltárolt adatokat, hogy az alkalmazás optimalizáltabban tudjon működni.

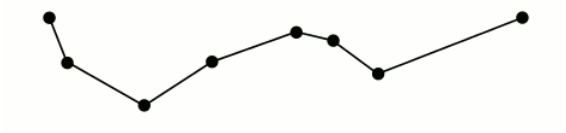
A Canvas API alkalmazása a *CanvasView* és *CanvasViewSC* osztályokban nem csak a rajzolási funkciókra korlátozódik, hanem a felhasználói interfész és a grafikai elemek kezelésére is kiterjed. Ez magában foglalja a rajzolási terület létrehozását, a felhasználói interakciók grafikus válaszainak megjelenítését, valamint az elkészült rajzok és gráfok vizualizációját. Az osztályok biztosítják, hogy az alkalmazás grafikai felülete intuitív és felhasználóbarát legyen, miközben magas szintű testreszabhatóságot és vizuális visszajelzést nyújt a felhasználók számára.

4.3.1. Douglas-Peucker algoritmus használata

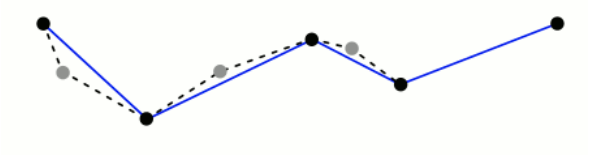
A rajzolt adatok egyszerűsítésében kulcsfontosságú szerepet játszik a Douglas-Peucker algoritmus. Az algoritmus célja, hogy csökkentse a vonalat alkotó pontok számát anélkül, hogy érdemben befolyásolná a vonal geometriai formáját. A matematikai hátteret tekintve, az algoritmus iteratívan vizsgálja meg a pontokat, meghatározva, hogy melyik pontok hagyhatók el anélkül, hogy érdemben befolyásolná a rajz általános formáját. Az algoritmus alkalmazása kritikus a teljesítmény szempontjából, mivel jelentősen csökkenti az adatok mennyiségét, amit kezelni kell, miközben javítja a rajzolási eredmények minőségét.

Algorithm 1 Douglas-Peucker Algoritmus

```
1:  procedure DouglasPeucker(Pontok, ε)
2:      d_max ← 0
3:      index ← 0
4:      N ← Pontok száma
5:      for i = 2 to N − 1 do
6:          d ← PontEgyenestőlValóTávolság(Pontok[i], Pontok[1], Pontok[N])
7:          if d > d_max then
8:              index ← i
9:              d_max ← d
10:         end if
11:     end for
12:     if d_max > ε then
13:         Eredmny1 ← DouglasPeucker(Pontok[1 . . . index], ε)
14:         Eredmny2 ← DouglasPeucker(Pontok[index . . . N], ε)
15:         return Eredmny1[1 . . . vége − 1] ∪ Eredmny2
16:     else
17:         return Pontok[1] és Pontok[N]
18:     end if
19: end procedure
```



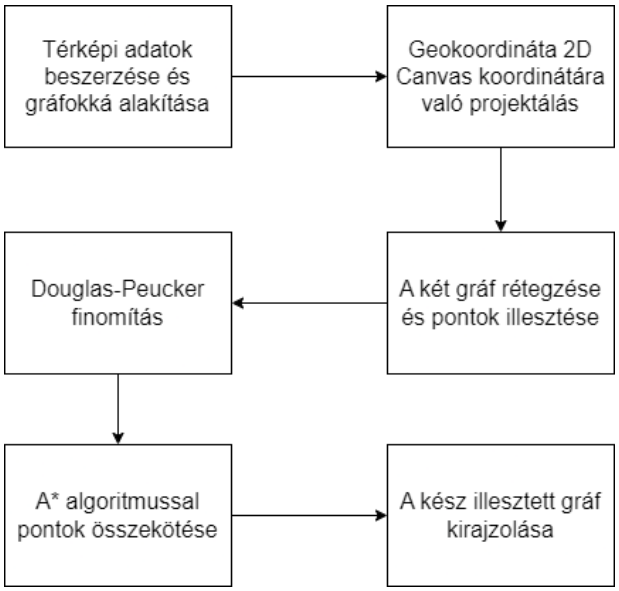
4.2. ábra: Pontokból álló gráf.



4.3. ábra: Miután alkalmaztuk rajta a Douglas-Peucker algoritmust.

4.4. A* útkereső és DFS gráfbejáró algoritmusokon alapuló módszer

A *CompareGrahs* osztály azonnali és egyszerű ráillesztést valósít meg, amely különösen hasznos olyan helyzetekben, ahol a felhasználónak már van konkrét elképzelése arról, hogy hova szeretné illeszteni a gráfot az úthálózaton belül. Ez az eljárás a felhasználói bemenetre és a geometriai illeszkedési pontok azonnali azonosítására épül, lehetővé téve a gyors és hatékony ráillesztést anélkül, hogy bonyolultabb gráfelméleti vagy gépi látási technikákat kellene alkalmazni.



4.4. ábra: Az illesztési mód menete.

Technikai megvalósítás és illesztési folyamat

A *findBestRotationMatch* metódus a *CompareGraph* osztály egy fontos funkciója, amelynek célja, hogy megtalálja a legjobb forgatási egyezést két gráf között: egy nagy, referencia gráf és egy transzformált gráf között. A funkció komplex algoritmusok és adatstruktúrák összehangolt használatát igényli a geometriai távolságok mérésére és az optimális útvonalak megtalálására. Ebben ez a fő metódus hívódik meg legelőször és ennek az eredménye tér vissza, mint illesztési megoldás. Ez a folyamat több lépésből áll, amelyek összetettsége és jelentősége kiemeli a gráfok összehasonlításának és illesztésének bonyolultságát.

Az adatok kezelését egy belső objektum, a DataHolder látja el, amely a gráf illesztési pontjait tartalmazza egy dinamikusan kezelt listában (matchedEndPoints). Ez a lista az illesztés során kapcsolt pontpárokat tárolja, amelyek a gráf illesztési folyamatát követik és dokumentálják. Az adatok tisztítására a clearData metódust használjuk, amely eltávolítja az összes korábban tárolt illesztési pontot, így a rendszer minden új illesztési folyamatot tiszta lappal kezdhet.

1. Legközelebbi pontok megkeresése: A folyamat első lépése a nagy gráf és a transzformált gráf közötti legközelebbi pontpárok azonosítása. Ezt a Manhattan-távolság segítségével végzik, amely egy egyszerű, mégis hatékony módszer a gráfok közötti kezdeti illesztési pontok meghatározására. A megfelelő pontok kiválasztása elengedhetetlen, mivel ez jelentősen befolyásolja az illesztés végső sikerességét.

2. Teljes útvonal építése az A* algoritmussal: Az A* algoritmus implementációja a *AStarNode* belső osztályon keresztül történik, amely a gráf csomópontjait képviseli a szükséges adatokkal, mint a csúcs (vertex) és a f-heurisztikai érték (fValue). Az fValue az adott csúcs kezdőponttól való távolságát és a becsült költséget jelzi a célcsúcsig. A csomópontokat egy prioritásos sorban (PriorityQueue) tárolják, ami biztosítja, hogy az algoritmus mindig a legígéretesebb csúcsot dolgozza fel először. Ezt követően, a kezdeti pontpárok azonosítása után, az A* algoritmust alkalmazzák az optimális összeköttetések létrehozására a nagy gráfban. Az algoritmus a gráf topológiáját és távolságait figyelembe véve építi ki a teljes útvonalat, amely biztosítja, hogy a transzformált gráf pontjai logikusan és hatékonyan kapcsolódjanak be a nagy gráf struktúrájába.

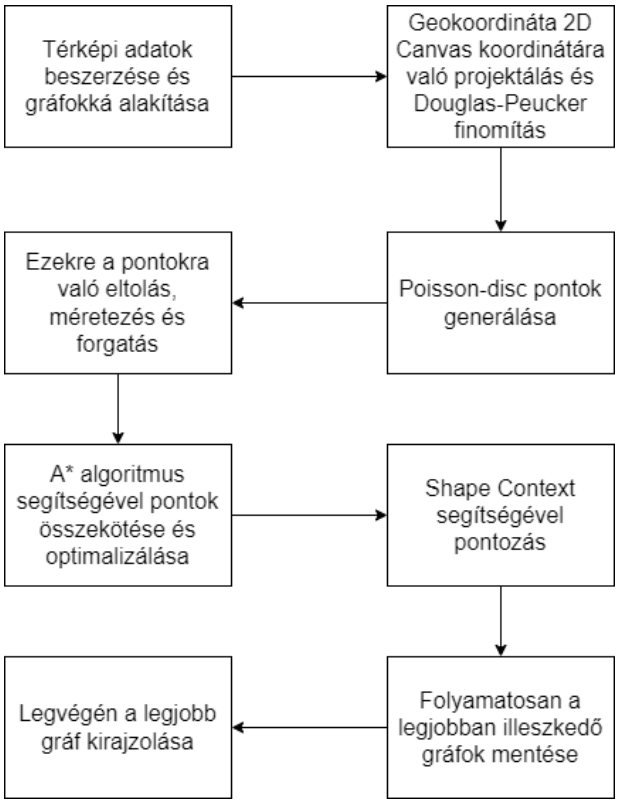
3. DFS alapú bejárás a gráf finomítására: A kezdeti útvonalépítést követően a DFS alapú bejárás történik a gráf finomítására. Ez a lépés lehetővé teszi az algoritmus számára, hogy felülvizsgálja és finomítsa az útvonalakat. Ebben a szakaszban eltávolításra kerülnek az esetleges redundáns vagy nem kívánatos élek, javítva ezzel az illesztés minőségét.

A metódus segítségével a gráf minden csúcsát bejárjuk, és csak azokat az éleket tartjuk meg, amelyek valóban hozzájárulnak az illesztési célkitűzésekhez. A isDesiredEdge függvény segít eldönteni, hogy egy adott él megfelel-e a kritériumoknak, figyelembe véve az élek hosszát és az élek közötti szögeket.

4. Túllógó élek visszavágása: Az utolsó lépésben a túllógó éleket vágja vissza a finomított gráfból. Ez a folyamat javítja az illesztés pontosságát, eltávolítva azokat az éleket, amelyek nem járulnak hozzá az optimális útvonalhoz, vagy amelyek torzíthatják az illesztés eredményét. Ez biztosítja, hogy a végső illesztett gráf tiszta és pontos legyen, megfelelve a felhasználói és alkalmazási követelményeknek.

4.5. Alakzat felismerésen alapuló módszer

Az osztály egyedi eljárást alkalmaz a gráfok összehasonlítására és illesztésére, mivel a Shape Context algoritmust integrálja, ami egy erőteljes gépi látási módszer. A Shape Context algoritmus kifejezetten arra lett kifejlesztve, hogy formákat és alakzatokat tudjon felismerni és összehasonlítani, így kiválóan alkalmazható a gráfok közötti illesztések pontosítására és optimalizálására. Ez a módszer kiemelkedik azzal, hogy képes a gráfok pontjainak lokális geometriai tulajdonságait figyelembe venni, amely segítségével sok potenciális illesztés közül kiválasztható a legjobb. A Shape Context algoritmus ereje abban rejlik, hogy minden egyes pontot egy alakzati kontextus vektor ír le, ami a pont körüli alakzatot jellemzi egy referencia pontból nézve. Ezáltal a pontok közötti összehasonlítás nem csak a távolságon, hanem a formájukon és relatív elhelyezkedésükön is alapul, lehetővé téve a gráfok közötti mélyebb és strukturáltabb összehasonlítást.



4.5. ábra: Az illesztési mód menete.

Az algoritmus log-polár koordináta rendszert használ, amely egy innovatív megközelítést kínál a pontok térbeli elrendezésének analízisére. Ebben a rendszerben egy kör, amely a referencia pont körül helyezkedik el, különböző szélességű gyűrűkre van osztva, és ezek a gyűrűk további egyenlő részekre, vagy szektorokra vannak felosztva. Ezeket a részeket gyakran hívják bin-eknek, amelyek minden egyes pont számára egyedi, lokális kontextust biztosítanak. A log-polár elrendezés lehetővé teszi a pontok relatív távolságának és szögének hatékony kódolását, így az alakzatok összehasonlítása sokkal informatívabb és pontosabb. A koordináta rendszer méretezése során minden pontot egy logaritmikusan skálázott sugár és egy szög jellemez.

Ez a módszer különösen alkalmas arra, hogy az alakzatok finom részleteit is figyelembe vegye, és a kisebb távolságokat nagyobb részletességgel reprezentálja, míg a nagyobb távolságokat kevésbé részletesen. A sugár logaritmikus skálázása azt jelenti, hogy a pontok távolságát az origótól logaritmus alapján számítjuk ki. Ez segít abban, hogy a közelebbi pontok közötti finom különbségeket nagyobb hangsúlyt kapjanak, míg a távolabbi pontok esetében ezek a különbségek kevésbé hangsúlyosak.

A logaritmikus sugár, logRadius, kiszámítása a következő képlet alapján történik:

$$\text{logRadius} = \ln(\text{radius} + 1)$$

ahol ln a természetes logaritmus jelölése, és +1-gyel növeljük a sugárat annak érdekében, hogy elkerüljük a logaritmus 0-ban való értelmezhetetlenségét.

4.6. Histogramok és Chi-négyzet alkalmazása

A Shape Context algoritmusban a histogramok kulcsfontosságú szerepet játszanak. Egy histogram ebben a kontextusban a bin-ekben lévő pontok gyakoriságának grafikus reprezentációja. Minden egyes bin egy histogram oszlopot alkot, amelynek magassága a bin tartalmazta pontok számát jelzi. Ezáltal egy pont shape context-e egy histogramban van kódolva, ami leírja a többi pontnak ehhez a referencia ponthoz viszonyított elhelyezkedését. Az összehasonlítás során két shape context histogramját hasonlítják össze, hogy megállapítsák, mennyire hasonlóak a két alakzat vagy gráf lokális geometriai tulajdonságai. [8]

Az összehasonlítás során a különböző pontok shape context-einek összehasonlítása lehetővé teszi a formák közötti hasonlóságok és különbségek pontos azonosítását. Ez a módszer rendkívül hatékony eszköz olyan alkalmazások számára, ahol a formák pontos összehasonlítása és azonosítása elengedhetetlen, például a képfelismerésben, gráfok illesztésében vagy digitális képfeldolgozásban. A log-polár koordináták alapján histogramokat készítünk, amelyek kvantálják a pontok eloszlását a térben. Minden pont számára egy histogram készül, amely megmutatja, hogy a többi pont milyen gyakorisággal fordul elő a különböző sugár- és szögtartományokban. A histogramok segítségével össze lehet hasonlítani, hogy két alakzat mennyire hasonlít egymáshoz a pontok relatív elhelyezkedése alapján. A Shape Context módszerben a "bin" egy adott tartományt jelöl a távolságok és szögek számításához. A binek gyakorlatilag "tárolók" vagy "rekeszek", amelyek segítenek osztályozni a pontpárok közötti relatív távolságokat és szögeket histogram formájában. A binek száma mind a sugár, mind a szög esetében befolyásolja a részletességet és az összehasonlítás pontosságát. Több bin esetén finomabb részletek figyelhetők meg, de a számítási igény is növekszik. A histogramok összehasonlítása során a Chi-négyzet tesztet vagy más hasonlósági mértékeket alkalmazhatunk a különbségek kvantitatív értékelésére.

A Chi-négyzet (χ^2) teszt képlete a következő:

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}$$

ahol:

- n a bin-ek (osztályok) száma,
- O_i az i-edik bin megfigyelt gyakorisága az egyik histogramban,
- E_i az i-edik bin elvárt (vagy másik histogramban mért) gyakorisága,
- $\sum$ a szummázás jelölése, tehát összeadjuk az összes binhez tartozó értékeket.

A képletben a $(O_i - E_i)^2$ kifejezés az egyes bin-ek megfigyelt és elvárt gyakoriságainak különbségét négyzetre emeli, ami azt jelzi, mennyire térnek el egymástól a gyakoriságok. Ezt a különbséget aztán elosztjuk az elvárt gyakorisággal (E_i), hogy normalizáljuk az eltéréseket a bin-ek gyakoriságainak abszolút mértékéhez képest. A χ^2 értéke tehát azt mutatja, hogy az összehasonlított histogramok mennyire különböznek egymástól statisztikailag jelentős módon. Minél nagyobb a χ^2 érték, annál nagyobb a különbség a két histogram között, ami azt sugallja, hogy a két alakzat vagy gráf kevésbé hasonlít egymáshoz.

Az alakzatillesztési folyamat során elengedhetetlen lépés az egyezőségi pontszám kiszámítása. Ez a mérték határozza meg, hogy mennyire hasonlítanak a két gráf. Az egyezőségi pontszám számítása a Shape Context algoritmusban történik, amely magában foglalja a histogramok összehasonlítását. Ezek a histogramok tartalmazzák a gráfok pontjai közötti távolságokat és szögeket, amelyeket bin-ekbe rendeznek. Az egyezőségi pontszám a különböző bin-ekben szereplő értékek alapján számított statisztikai eltérésekkel, például a Chi-négyzet teszttel van meghatározva.

A két gráf közötti hasonlósági pontszám, SimilarityScore, kiszámítása a következő módon történik:

$$\text{SimilarityScore} = \sum_{i=1}^{N} \sum_{j=1}^{M} \sum_{k=1}^{L} \frac{(O_{ijk} - E_{ijk})^2}{O_{ijk} + E_{ijk}}$$

ahol:

- N a histogramok száma,
- M és L az egyes histogramokban lévő bin-ek számát jelöli a sugár és szög dimenziókban,
- O_{ijk} az i-edik histogram j-edik sugárbinjének és k-edik szögbinjének megfigyelt értéke az eredeti gráfban,
- E_{ijk} az i-edik histogram j-edik sugárbinjének és k-edik szögbinjének megfigyelt értéke a transzformált gráfban.

Ebben a képletben a Chi-négyzet teszt alkalmazásával számoljuk ki a megfigyelt (O_{ijk}) és elvárt (E_{ijk}) értékek közötti eltéréseket. Az egyes bin-ekre vonatkozó eltérések összegezése után kapjuk meg a teljes hasonlósági pontszámot, ami jelzi a két gráf közötti hasonlóság mértékét. Minél kisebb ez az érték, annál nagyobb a hasonlóság a két gráf között.

4.7. Poisson-disk mintavétel generálása

A Poisson Disk mintavétel egy elterjedt módszer a számítógépes grafikában és szimulációkban, amely lehetővé teszi, hogy egy adott területen belül pontokat generáljunk úgy, hogy azok egyenlő távolságra legyenek egymástól. Ez a technika különösen hasznos olyan esetekben, amikor szükség van a minták egyenletes eloszlására anélkül, hogy szabályos rácsstruktúrát hoznánk létre, így természetesebb megjelenést eredményezve.

A generatePoissonDiscPoints függvény egy példa arra, hogyan lehet a Poisson Disk mintavételt alkalmazni egy gráf illesztési folyamat során. A függvény célja, hogy generáljon egy sor pontot a vásznon belül, amelyek mindegyike egy minimális távolságra van egymástól. Ezáltal a függvény egy sor potenciális illesztési pontot hoz létre, amelyekre aztán az illesztendő gráfot el lehet tolva, forgatni és méretezni, ezzel rengeteg illesztési próbálkozást létrehozva.

1. Kezdőpont hozzáadása: Az algoritmus egy kezdőponttal indul, amely általában a vászon középpontjában helyezkedik el. Ezt a pontot hozzáadjuk az aktív lista és a mintapontok listájához.

2. Pontok generálása: Minden aktív pont körül az algoritmus próbál új pontokat generálni, figyelembe véve egy előre meghatározott minimális távolságot (minDist) és egy maximális sugárt. Ezeket a pontokat véletlenszerűen választja ki a körön belül.

3. Távolság és helyzet ellenőrzése: Minden új pont esetében az algoritmus ellenőrzi, hogy az adott pont elegendő távolságra van-e a többi ponttól és hogy a pont a kijelölt körön belül helyezkedik-e el. Csak azok a pontok kerülnek hozzáadásra a mintákhoz, amelyek megfelelnek ezeknek a kritériumoknak.

4. Iteráció az aktív lista üresedéséig: Az algoritmus addig folytatódik, amíg az aktív lista ki nem ürül, vagy amíg el nem éri a kívánt pontszámot.

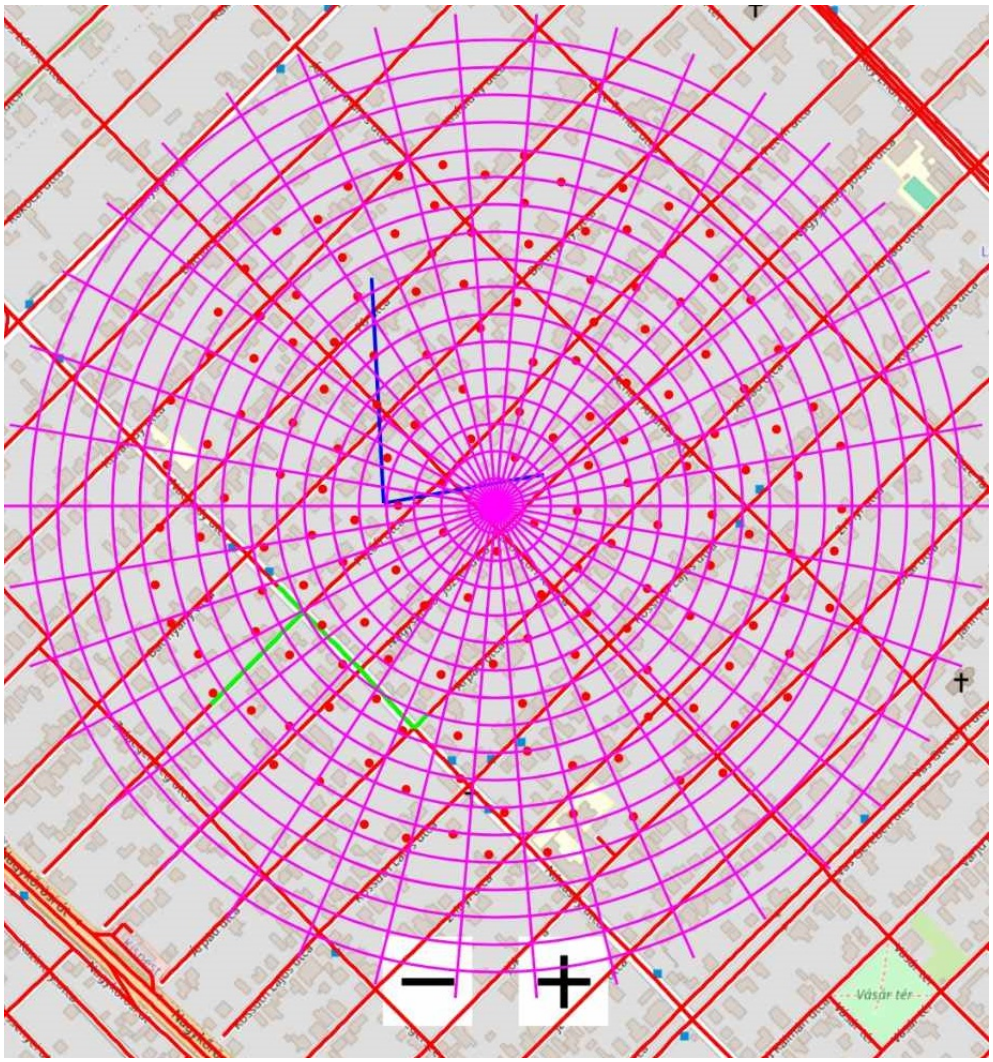
A Poisson Disk mintavétel alkalmazása a gráf illesztési folyamatban lehetővé teszi a kísérletezési tér széles körű feltárását, ahol az illesztendő gráfot különböző helyzetekben, forgatásokban és méretezésekben próbálhatjuk ki. Ezáltal nagyobb esélyt ad arra, hogy megtaláljuk a legjobb illesztési eredményt, miközben elkerüljük a minták túlzsúfolását vagy a természetellenes elrendeződést.

A következőképpen néz ki:

$$x' = x + r\cos(\theta)$$
$$y' = y + r\sin(\theta)$$

ahol:

- x', y' az új pont koordinátái,
- x, y a referencia pont koordinátái,
- r a generált pont távolsága a referencia ponttól, ami nagyobb, mint a minimális távolság ($minDist$),
- θ egy véletlenszerűen választott szög 0 és 2π között.



4.6. ábra: Poisson-féle mintavételezési pontok és a Log-polár koordináta rendszer.

4.8. Az illesztési folyamat és felhasználói visszajelzés

A gráf illesztési folyamat utolsó lépéseit kezelik a *Finish* és *FinishSC* osztályok, véglegesítve a megjelenítést és biztosítva a felhasználói visszajelzést. Ezek az osztályok elhanyagolhatatlanok a végső képi eredmény előállításában és annak bemutatásában a felhasználónak. A felhasználó ezen az oldalon tudja újrakezdeni a folyamatot vagy pedig személyreszabni az illesztést.

Az egyesítési folyamat során az osztályok a *Bitmap* és *Canvas* objektumokat használják az eredeti térkép és az illesztett gráf együttes megjelenítésére. A térkép egy *Bitmap* (.png kiterjesztésű kép) objektumként szolgál, amelyre a *Canvas* segítségével rajzolódik rá a gráf élei és csúcsai. Ezáltal a felhasználók vizuálisan értékelhetik, hogyan helyezkedik el a gráf a térképen, és mennyire pontos az illesztés.

A végső kép előállítása során a következő technikákat használtam a hatékony és esztétikailag vonzó megjelenítés érdekében:

- Bitmap kezelése: A térképet reprezentáló *Bitmap* objektum betöltése az első lépés. Ez pedig a *MainAcitivty*-ról való elnavigáláskor kimentett térkép részletetet tartalmazza, amit a külső memóriában tárolunk a procedúra végéig.
- Canvas használata: A *Canvas* objektumot arra használjuk, hogy a *Bitmap*-re rajzoljuk a gráfot. A *Canvas* lehetővé teszi a gráf éleinek és csúcsainak pontos és tiszta rajzolását, különböző színek és vonalvastagságok alkalmazásával. Ez segít a gráf kiemelésében a térképen, megkönnyítve az illesztés vizuális ellenőrzését.
- Vonalak és formák rajzolása: A gráf éleit vonalak formájában rajzoljuk ki. A szín- és vastagságváltozatok segítségével növeljük a vizuális kontrasztot, így a gráf könnyen megkülönböztethető a térkép többi elemétől.
- Felhasználói interaktivitás: A végső megjelenítés interaktív elemeket is tartalmazhat, mint például a zoomolás és a mozgatás képessége, ami lehetővé teszi a felhasználók számára, hogy részletesebben vizsgálják meg az illesztési eredményeket.

Ezek a technikák együttesen biztosítják, hogy a *Finish* osztály hatékonyan és esztétikusan jelenítse meg a gráfot a térképen, lehetővé téve a felhasználó számára, hogy értékelje az illesztési folyamat végeredményét. Az ilyen típusú vizuális visszajelzés nélkülözhetetlen a felhasználói élmény szempontjából, mivel konkrét betekintést nyújt az illesztés sikerességébe és az esetleges hibákba.

A felhasználói interakció és visszajelzés

A felhasználó felé való visszajelzés fontos az alkalmazások interaktivitásában és a felhasználói élmény javításában, különösen olyan komplex folyamatok esetén, mint a gráf illesztése. A *FinishSC* osztályban implementált visszajelzési mechanizmusok segítenek a felhasználónak nyomon követni a folyamat előrehaladását. A haladási állapotot százalékos értékekkel jelöljük, amelyek 0-tól 100-ig terjednek, jelezve az illesztési folyamat teljesítettségét. A százalékos jelzés mellett a *ProgressBar* komponenst is használjuk, amely vizuálisan ábrázolja a folyamat előrehaladását. A folyamatosan frissülő

haladási sáv segít a felhasználóknak látható módon követni az illesztés állapotát. Az alkalmazás lehetővé teszi a felhasználók számára, hogy interaktív módon változtassák a gráf megjelenítésének módját vagy a gráf színét. A *ColorPickerActivity* indításával a felhasználók kiválaszthatják a kívánt színt, amelyet aztán a gráf megjelenítéséhez használunk. A megjelenítési mód váltó gomb segítségével pedig a felhasználók választhatnak az eredeti, transzformált vagy mindkét gráf egyidejű megjelenítése között. Az illesztési folyamat állapotának figyelésére és a felhasználói felületen való megjelenítésére a *ProgressListener* interfészt használjuk, amely lehetővé teszi, hogy a *FinishSC* osztály reagáljon az illesztési folyamat különböző állapotaira. Az interfész *onProgressUpdate* metódusán keresztül kapott százalékos értékeket a felhasználói felületen jelenítjük meg, ezzel biztosítva a folyamatos visszajelzést a felhasználó felé. Ezek a mechanizmusok növelik az alkalmazás interaktivitását és a felhasználói élményt, lehetővé téve a felhasználók számára, hogy aktív résztvevőivé váljanak a gráf illesztési folyamatnak.

4.7. ábra: Az illesztendő Amung Us figura rajz (bal) és az illeszési eredmény (jobb)

4.9. Az illesztések személyre szabása

Jetpack Compose egy modern toolkit az Android platformra, amely a Kotlin nyelvet felhasználva lehetővé teszi a felhasználói felületek deklaratív megközelítésben történő kifejezését. E keretrendszer segítségével a fejlesztők közvetlenül a kódban írhatják le az UI elemeket, anélkül, hogy XML layout fájlokat kellene használniuk. Ez szorosabb integrációt és nagyobb rugalmasságot biztosít a felület és a háttérlogika között.

A Compose a reaktív programozási elveket követi, így az alkalmazás felülete automatikusan frissül az adatok változásakor, lehetővé téve dinamikus és interaktív felületek hatékony létrehozását. A keretrendszer egyszerűsíti a kódbázist és felgyorsítja a fejlesztési folyamatot, kihasználva a Kotlin nyelv modern funkcióit, mint a lambdák, bővítő funkciók és a coroutines. Ezáltal jelentősen csökken a redundáns kód és egyszerűbbé válik az UI logika nyomon követése, miközben a fejlesztők kevesebb erőfeszítéssel hozhatnak létre dinamikus, interaktív felhasználói felületeket. Például, egy gomb megjelenítése a Compose-ban a következőképpen nézhet ki:

Programkód 4.4. Compose gomb

```
Button(onClick = { /* kezeles */ }) {
    Text("Kattints ide")
}
```

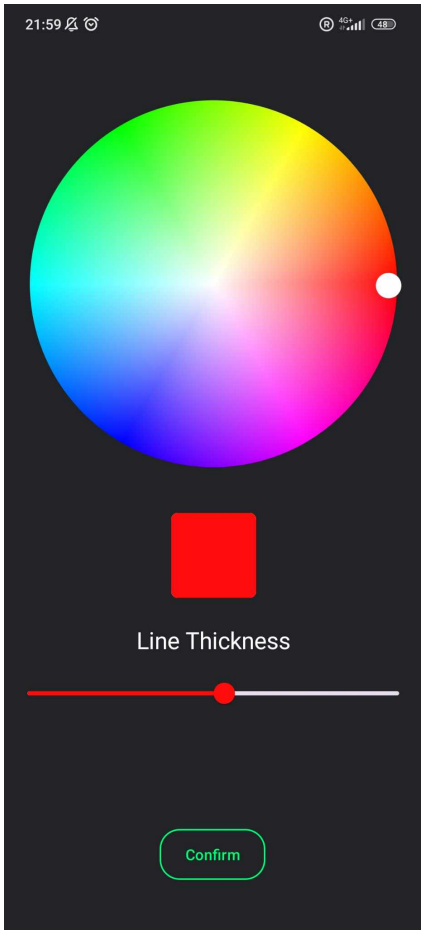
Ebben a példában a Button és a Text funkciók Composable függvények, amelyek a felület egyes elemeit írják le. Az onClick lambda kifejezés biztosítja a gombra kattintáskor végrehajtandó műveletet.

- Gyors Iteráció: A Compose deklaratív és reaktív jellege lehetővé teszi a fejlesztők számára, hogy gyorsabban iteráljanak a felhasználói felület tervezésekor, mivel a változások azonnal láthatóvá válnak az alkalmazásban.
- Egységes nyelv: Mivel a Compose a Kotlin nyelven alapul, a fejlesztőknek nem kell külön nyelvet tanulniuk a UI és a háttérlogika kezelésére, ezáltal egységesítve a fejlesztési élményt.
- Komponens alapú: A Compose lehetővé teszi az újrafelhasználható és összetevő alapú UI építését, ami javítja a kód modularitását és karbantarthatóságát.

Ezen technológiát kihasználva egy kész korszerű színválasztó felületet implementáltam. [17] A színválasztó felület középpontjában egy HSV (Hue, Saturation, Value) színkör áll, amelyet a HsvColorPicker Composable funkcióval hozok létre. Ez a színkör lehetővé teszi a felhasználók számára, hogy intuitívan válasszanak színt a teljes spektrumból, miközben azonnal látják a kiválasztott szín előnézetét.

- AlphaTile: Az aktuálisan kiválasztott szín vizuális visszajelzését egy AlphaTile elem biztosítja, amely dinamikusan frissül, hogy megmutassa a kiválasztott színt nagyobb méretben. Ez a vizuális visszajelzés segít a felhasználónak abban, hogy pontosan lássa, melyik színt választotta, még mielőtt alkalmazná azt a gráfra.
- Színváltozás dinamikus csúszka: A vonalvastagság beállítására szolgáló csúszka szintén dinamikusan változik a kiválasztott szín alapján. Ez a design elem nem csak esztétikailag vonzó, hanem a felhasználói élményt is javítja, mivel vizuálisan összekapcsolja a csúszka működését a kiválasztott színnel.

- Csúszka a vonalvastagság beállításához: A felhasználók a vonalvastagságukat is finomhangolhatják egy csúszka segítségével, amely a Slider Composable funkcióval van megvalósítva. Ez a csúszka lehetővé teszi a felhasználók számára, hogy pontosan állítsák be a gráf vonalainak vastagságát, így azok a megjelenítés során jobban kiemelkednek.



4.8. ábra: A színválasztó az alkalmazásban.

5. fejezet

Ellenőrzés és validálás

Az alkalmazás rendszerigénye és teljesítménye

A processzor és memória intenzív használatának egyik oka a szálkezelés, amely lehetővé teszi a feladatok egyidejű végrehajtását, ezzel növelve az alkalmazás hatékonyságát és válaszidejét. Azonban ez a megközelítés növeli a rendszerigényt és szükségessé teheti a fejlettebb hardverek használatát.

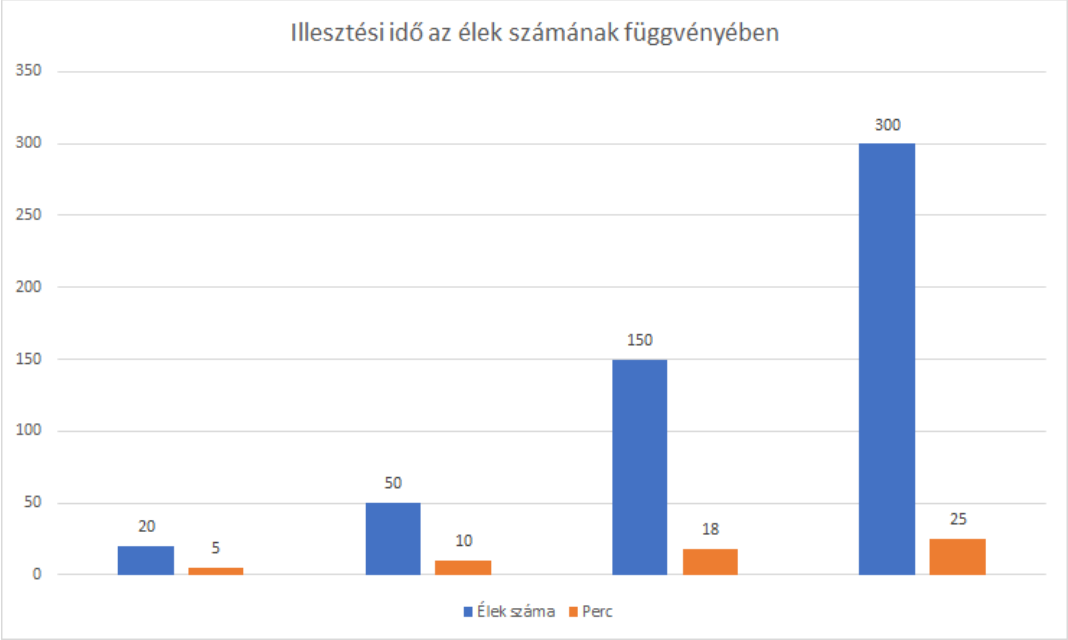
Az alkalmazás zavartalan használatához ajánlott egy közép- vagy magas szintű hardverkonfiguráció, amely magában foglalja:

- Legalább 4 magos processzor, előnyösen magas órajellel rendelkező, hogy kezelni tudja a magas processzorterhelést.
- Minimum 4 GB RAM, hogy a telefon kibírja az illesztést és a többi program esetleges háttérben való futtatását. A modern környezet miatt az alkalmazás kiélvezi a garbage collector (GC) előnyét.
- Stabil és gyors internetkapcsolat, amely elengedhetetlen az úthálózat adatok letöltéséhez és az alkalmazás frissítéseihez.

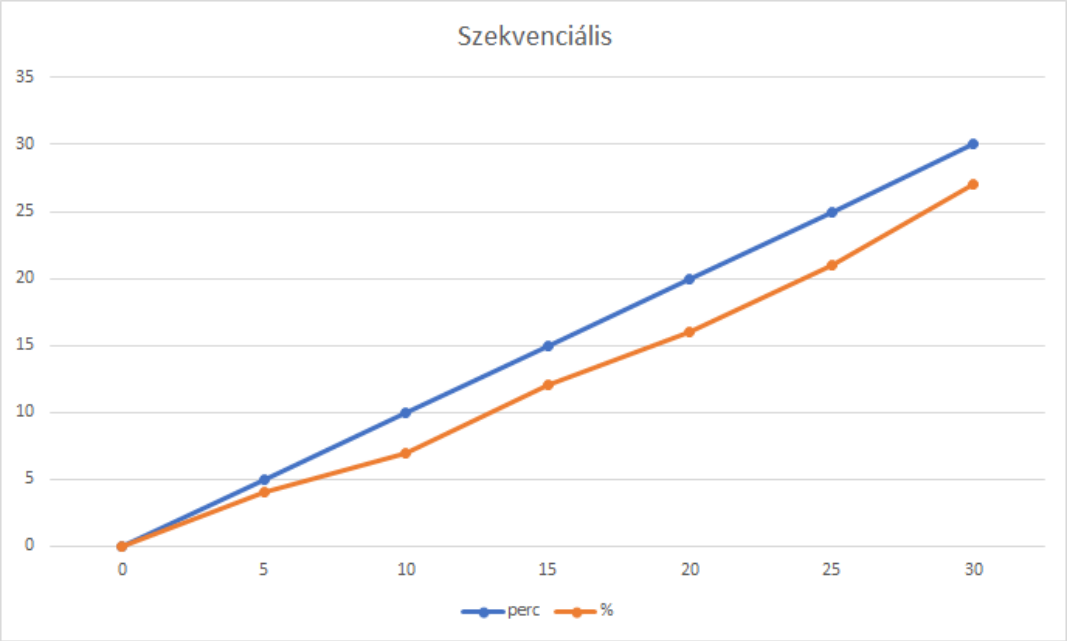
A GC automatikusan kezeli a nem használt objektumok memóriából való eltávolítását, így csökkenti a memóriaszivárgás kockázatát és optimalizálja az alkalmazás memóriahasználatát. Az alkalmazásom fejlesztése során tapasztaltam, hogy a GC tevékenységének köszönhetően az alkalmazás memóriahasználata nem haladja meg a 900 MB-ot, még intenzív műveletek végrehajtása során sem. Ez a jelenség különösen fontos, mivel biztosítja, hogy az alkalmazás megfelelő teljesítménnyel fusson, anélkül, hogy túlzottan terhelné az eszköz memóriáját.

Az alkalmazás fejlesztése során nagy hangsúlyt fektettem a teljesítmény és a felhasználói élmény optimalizálására. A fejlesztett mód rendszerigényeinek ismerete és a megfelelő hardver biztosítása lehetővé teszi a felhasználók számára, hogy a legtöbbet hozzák ki ebből az illesztési módból.

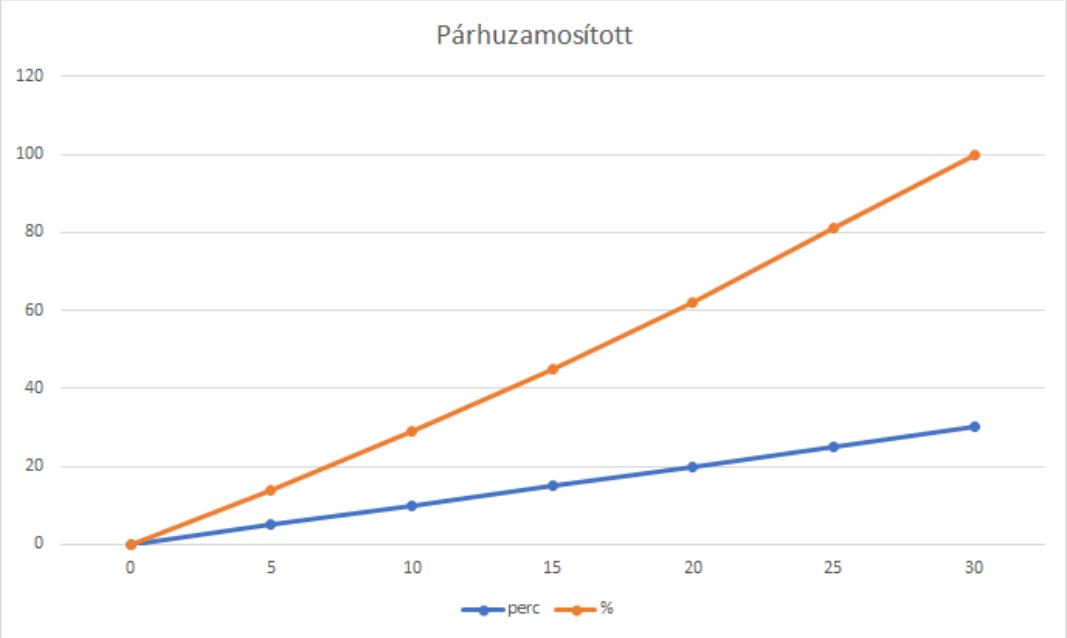
5.1. ábra: A rendszer kihasználtság.



5.2. ábra: Illesztési idő az élek számának növekédésének függvényében.



5.3. ábra: Szekvenciális illesztés.



5.4. ábra: Párhuzamosított illesztés.

6. fejezet

Fejlesztési lehetőségek

A fejlesztés során szembesültem a szálkezelés által okozott problémákkal is, amikor a folyamat atomikussága veszélybe került. Ez a komplexitás növelése során okozott nehézségeket, mivel több szál egyidejű változtatási kísérlete kivétel dobásához vezetett a kotlinban.

A Shape Context módszer további finomításra és optimalizálásra szorul, hogy gyorsabb és hatékonyabb legyen. Mindazonáltal, tekintettel a személyes fejlődésemre és az elért eredményekre, úgy érzem, hogy a dolgozatom során jelentős mértékben hozzájárultam a téma elmélyítéséhez és újszerű megoldások kifejlesztéséhez.

A jövőbeli tervek között szerepel az összetett mód továbbfejlesztése annak érdekében, hogy több, egymástól független gráfot is képes legyen hatékonyan illeszteni anélkül, hogy nem kívánt kapcsolatokat hozna létre. A másik terület, ahol jelentős fejlődési potenciált látok, az az illesztési folyamat időbeli hatékonyságának javítása. Jelenleg a Shape Context algoritmus, különösen az összetett illesztési eljárás, jelentős számítási erőforrásokat igényel, ami hosszabb várakozási időket eredményezhet. Ennek megoldása érdekében szeretnék kutatni a párhuzamosítás lehetőségeit, különösen a modern okostelefonok GPU-jainak (grafikus feldolgozó egységek) felhasználásával. A GPU-k párhuzamosítási képességei lehetővé tennék az illesztési algoritmusok gyorsabb végrehajtását, csökkentve ezzel az illesztési időt és javítva az alkalmazás válaszidejét.

Ezen túlmenően, a fejlesztés során szeretném kiterjeszteni az alkalmazás képességeit új funkciók bevezetésével, amelyek még inkább kihasználják az adatok és a gráfelmélet kombinációját. Ez magában foglalhatja a felhasználó által definiált paraméterek alapján történő dinamikus útvonaltervezést, ahol a felhasználók testre szabhatják az illesztési kritériumokat, hogy a legjobban megfeleljenek saját igényeiknek.

Végül a felhasználói visszajelzések és a közösségi hozzájárulások integrálása a fejlesztési folyamatba fontos szerepe lesz. A felhasználók tapasztalatai és javaslatai értékes betekintést nyújthatnak az alkalmazás továbbfejlesztéséhez szükséges területekre, segítve ezzel, hogy az alkalmazás még inkább megfeleljen a felhasználói elvárásoknak és szükségleteknek.

7. fejezet

Összefoglalás

A kutatásom során mélyrehatóan foglalkoztam az informatika, térinformatika és gráfelmélet területeivel. Kezdetben a projekt indítása nagy kihívást jelentett, azonban a folyamat végére a munka már sokkal könnyebbnek és gördülékenyebbnek tűnt. E fáradozások eredményeképpen az alap illesztési mód kifogástalanul működik, ami jelentős sikernek számít. Ezzel szemben a fejlettebb mód további kihívásokkal szembesített. Jelenlegi állapotában csak egyetlen vonalból álló gráfok illesztésére képes hatékonyan, mivel több, egymástól független gráf esetén a rendszer nem kívánt kapcsolatokat hoz létre az A* algoritmus segítségével és gond van az algoritmus atomikusságával.

A fejlesztés előrehaladtával számos megközelítést teszteltem, amelyek közül végül sikerült kifejleszteni egy rendszert, amely képes gráfokat ráilleszteni más gráfokra, miközben megőrzi azok struktúráját és így az eredeti rajzok felismerhetőek maradnak. Ebben az összetett folyamatban különböző technológiákra és megoldásokra támaszkodtam. A projekt során számos matematikai és informatikai területtel kerültem kapcsolatba, köztük a Poisson-disc mintavétellel, amely lehetővé tette számomra a ponteloszlások optimalizálását a gráfokon, a Shape Context algoritmust, amely az alakzatfelismerésre fókuszál, és a Chi-négyzet tesztet, amely a pontok közötti statisztikai összefüggések mérésére szolgál.

Emellett mélyebben elmerülhettem a Kotlin programozási nyelv használatában, amely a fejlesztés során a kódolás hatékonyságát és olvashatóságát javította. A Kotlin modern jellemzői, mint a null-biztonság és a coroutine-k kezelése, nagymértékben hozzájárultak az alkalmazás teljesítményének és megbízhatóságának javításához. A GPS-alapú térinformatikai technológiák megértése és alkalmazása szintén fontos játszott a projektben, lehetővé téve a valós idejű helymeghatározást és térképi integrációt.

Összességében, bár voltak kihívások, amelyeket le kellett küzdeni, és vannak területek, ahol a fejlesztés folytatódni fog, büszke vagyok arra, amit sikerült elérnem. Optimista vagyok az alkalmazás jövőbeli fejlődését illetően, és hálás vagyok azért a sokrétű tapasztalatért, amit a projekt során szerezhettem. Ezek az ismeretek és készségek nemcsak hogy megerősítették szakmai alapjaimat, de a jövőbeni karrierem során is értékes segítségemre lesznek.

8. fejezet

Köszönetnyilvánítás

Szaldolgozatom elkészítése során számos ember támogatását élveztem, akik nélkül ez a projekt nem valósulhatott volna meg. Szeretném kifejezni hálámat és köszönetemet mindazoknak, akik hozzájárultak a munkámhoz.

Elsőként szeretném megköszönni témavezetőmnek, Szabó Martinnak, aki nemcsak hogy felvetette a téma ötletét, de szakmai útmutatásaival és támogatásával végigkísért az elkészítés folyamatában.

Hálás vagyok konzulensemnek, Dr. Veres Laurának, aki a projekt matematikai aspektusainak megértésében nyújtott segítséget.

Nem maradhatnak említés nélkül barátaim sem, akik lelki és mentális támogatást nyújtottak számomra a projekt során. Ők voltak azok, akik mellettem álltak a kihívások és nehézségek idején, biztatásukkal és jó szavaikkal erőt adtak a folytatáshoz.

Források

[1] Serge Belongie, Jitendra Malik és Jan Puzicha. *"A new descriptor for shape matching and object recognition"*. Elérhető: https://proceedings.neurips.cc/paper_files/paper/2000/file/c44799b04a1c72e3c8593a53e8000c78-Paper.pdf.

[2] Robert Bridson. *"Fast Poisson Disk Sampling in Arbitrary Dimensions"*. Elérhető: https://www.cs.ubc.ca/~rbridson/docs/bridson-siggraph07-poissondisk.pdf.

[3] Britannica. *"Graph Theory"*. Elérhető: https://www.britannica.com/topic/graph-theory. 2024.

[4] Android Developers. *"Layouts in Views"*. Elérhető: https://developer.android.com/develop/ui/views/layout/declaring-layout. 2024.

[5] GeeksforGeeks. *"A* Search Algorithm"*. Elérhető: https://www.geeksforgeeks.org/a-search-algorithm/. 2024.

[6] Jennifer Langston. *"UW mapping app turns art into a sharable walking route"*. Elérhető: https://www.washington.edu/news/2015/05/06/uw-mapping-app-turns-art-into-a-sharable-walking-route/.

[7] D.H Maling. *"Coordinate Systems and Map Projection"*. Elérhető: https://books.google.hu/books?hl=hu&lr=&id=tcL-BAAAQBAJ&oi=fnd&pg=PP1&dq=geo+coordinates+to+map&ots=YqLmr4iOAm&sig=u6qoE2lvVoTjaVlYAvyZ7bbxac8&redir_esc=y#v=onepage&q&f=false. 1992.

[8] National Library of Medicine. *"The Chi-square test of independence"*. Elérhető: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3900058/. 2013.

[9] Overpass. *"Overpass API Documentation"*. Elérhető: https://osmlab.github.io/learnoverpass/en/docs/. 2015.

[10] Proj. *"Web-Merchator"*. Elérhető: https://proj.org/en/9.2/operations/projections/webmerc.html. 2023.

[11] ResearchGate. *"A*-based Pathfinding in Modern Computer Games"*. Elérhető: https://www.researchgate.net/publication/267809499_A-based_Pathfinding_in_Modern_Computer_Games. 2010.

[12] ResearchGate. *"GIS based road connectivity evaluation using Graph Theory"*. Elérhető: https://www.researchgate.net/publication/352361595_GIS_based_road_connectivity_evaluation_using_Graph_Theory. 2019.

[13] Mohneesh S. *"Applications of Ramer-Douglas-Peucker Algorithm in Machine Learning That You Might Not Have Heard of."* Elérhető: https://mohneesh0.medium.com/applications-of-ramer-douglas-peucker-algorithm-in-machine-learning-that-you-might-not-have-heard-63b0c4f15a43. 2021.

[14] ScienceDirect. *"Automatic segmentation of left and right cerebral hemispheres from MRI brain volumes using the graph cuts algorithm"*. Elérhető: https://www.sciencedirect.com/science/article/abs/pii/S1053811906007476. 2007.

[15] ScienceDirect. *"A survey of graph theoretical approaches to image segmentation"*. Elérhető: https://www.sciencedirect.com/science/article/abs/pii/S0031320312004219. 2012.

[16] ScienceDirect. *"Path Planning with Modified a Star Algorithm for a Mobile Robot"*. Elérhető: https://www.sciencedirect.com/science/article/pii/S187770581403149X. 2014.

[17] skydoves. *"Color picker"*. Elérhető: https://github.com/skydoves/colorpicker-compose. 2023.

[18] Firefly's space. *"Disadvantage of Shape Context"*. Elérhető: https://avaminzhang.wordpress.com/2012/12/10/disadvantage-of-shape-context/. 2012.

[19] Ted Steiner. *"OpenStreetMap Documentation"*. Elérhető: https://openstreetmapjl.readthedocs.io/en/stable/. 2014.

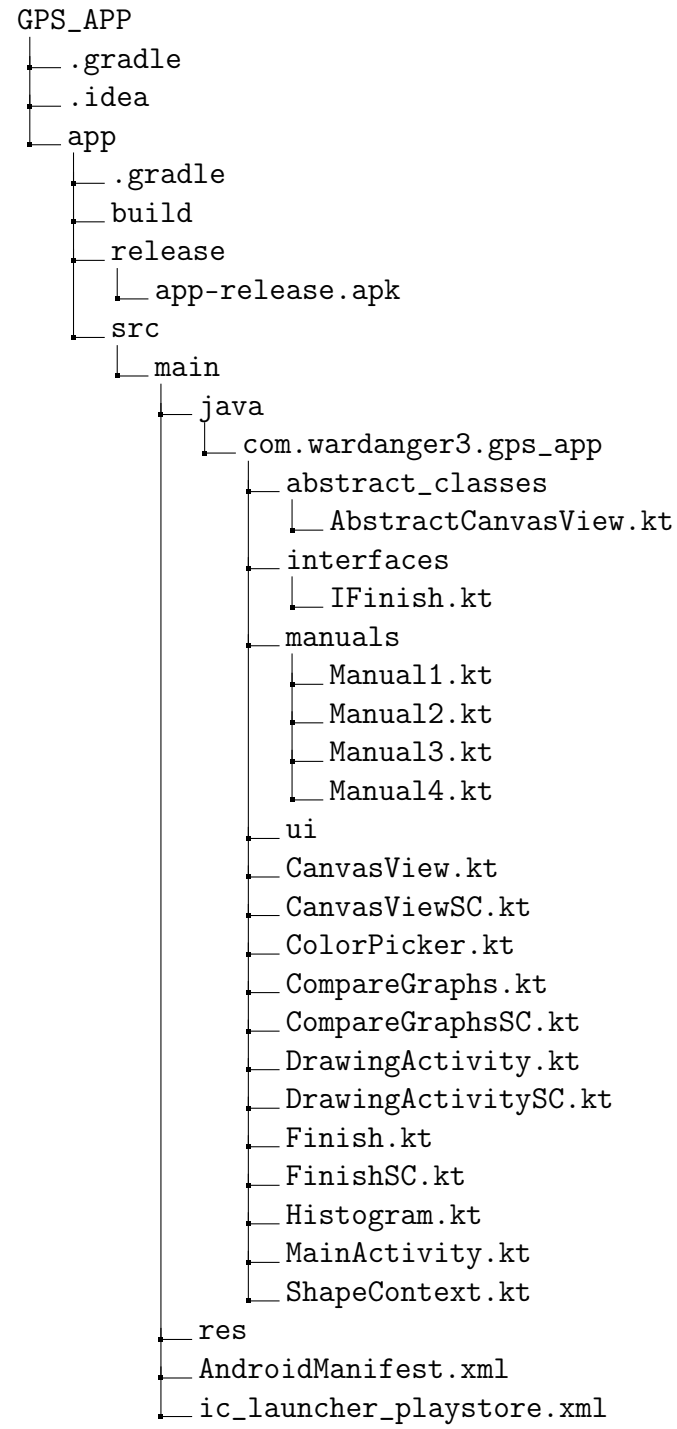
[20] ThatsMaths. *"Maps on the Web"*. Elérhető: https://thatsmaths.com/2015/05/28/maps-on-the-web/. 2015.

[21] Ash Turnel, Marlene Centeno és Kim Juanillo. *"How Many Android Users Are There? Global and US Statistics (2024)"*. Elérhető: https://www.bankmycell.com/blog/how-many-android-users-are-there. 2024.

[22] IEEE Xplore. *"A real-time parallel implementation of Douglas-Peucker polyline simplification algorithm on shared memory multi-core processor computers"*. Elérhető: https://ieeexplore.ieee.org/abstract/document/5620612. 2010.

A mellékelt adattároló tartalma

A projekt fájlstruktúrája a következőképpen néz ki:



Az alkalmazás elindítása

A project mappán belül keressük meg az **release** mappát, amiben megtalálható maga a telefonra telepíthető alkalmazás. Egy adatbákel segítségével másoljuk át a telefonunkra, majd pedig indítsuk el a telepítést. Ha nincs adatkábel a közelben, akkor egyszerűen a PlayÁrúház megnyitása után keressünk rá a MapCanva nevű alkalmazásra.