



**Budapesti Műszaki és Gazdaságtudományi Egyetem**

Villamosmérnöki és Informatikai Kar

Automatizálási és Alkalmazott Informatikai Tanszék

Kovács Boldizsár

# **JÁRÓKELŐK MOZGÁSÁNAK SZIMULÁCIÓJA**

KONZULENS

**Dr. Goldschmidt Balázs**

BUDAPEST, 2022

# Tartalomjegyzék

<b>Kivonat.....</b>	<b>5</b>
<b>Abstract.....</b>	<b>6</b>
<b>1 Bevezetés .....</b>	<b>7</b>
1.1 Emberek mozgásának segítése.....	8
1.2 Tűzvédelmi szempontok .....	9
1.3 A feladat leírása .....	10
1.4 Irodalomkutatás és a feladatkör kiterjesztése .....	11
1.4.1 Programozási környezet megválasztása.....	11
1.4.2 Útvonal keresés.....	12
1.4.3 Dijkstra algoritmus .....	12
1.4.4 A* algoritmus .....	14
1.4.5 Helyszín átalakítása gráfra.....	16
1.4.6 Felületeket lefedő gráfok .....	17
1.4.7 Útpont gráf.....	18
1.4.8 Navigációs háló.....	19
1.4.9 Legrövidebb útvonal és útvonal kiegyenesítése .....	20
1.4.10 Emberek mozgásának alapjai.....	21
1.4.11 Emberek szélessége .....	21
1.4.12 Emberek mozgási sebessége egyedül és tömegben .....	22
1.4.13 Vézhelyzet kezelése .....	23
<b>2 Saját munka bemutatása: Első program.....</b>	<b>25</b>
2.1 A program elvárt működése.....	25
2.2 Emberek ütközésének elkerülése .....	26
2.3 Első program korlátai.....	26
2.3.1 Mezők összeszervezése.....	27
2.3.2 Útvonal és mozgás megtervezése .....	28
2.3.3 Időegység nagyságának megválasztása .....	28
2.3.4 Időpillanatonként állapotképek.....	30
2.3.5 Falak a játéktérben .....	31
2.3.6 Ütközések elkerülésének megvalósítása.....	32

2.3.7 Útvonalkereső algoritmus megalkotás.....	33
2.3.8 Megmaradt problémák orvoslása.....	34
2.3.9 Első program eredménye .....	34
2.3.10 Megalkotott program grafikus felülete .....	35
<b>3 Saját munka bemutatása: Második program .....</b>	<b>36</b>
3.1 Navigációs hálót határoló síkidomok létrehozása.....	37
3.1.1 A navigációs háló háromszögeinek létrehozása .....	38
3.1.2 A bejárható tér háromszögekre bontása.....	39
3.1.3 A szobák emeletté alakítása.....	42
3.1.4 Az útvonal tervezése.....	43
3.1.5 Szobákban generált útvonalak összekötése .....	45
<b>4 Járókelők mozgásának szimulációja .....</b>	<b>46</b>
4.1.1 Járókelők sebességének megválasztása .....	46
4.1.2 Normák betartása .....	47
4.1.3 Útvonal követése.....	48
4.1.4 A járókelők napirendje.....	48
4.1.5 Járókelők létrehozása .....	49
4.1.6 A járókelők életciklusa és az emelet létrehozása .....	49
<b>5 Összefoglalás, eredmények értékelése.....</b>	<b>50</b>
5.1 A szobák betöltése és emeletté alakítása .....	50
5.2 Az emelet navigációs hálójának létrehozása.....	51
5.2.1 Az algoritmus hatékonyságának növelése .....	52
5.3 Az útvonalkeresés .....	52
5.4 Összefoglalás .....	53
<b>6 Irodalomjegyzék.....</b>	<b>54</b>
<b>7 Függelék .....</b>	<b>56</b>

# HALLGATÓI NYILATKOZAT

Alulírott **Kovács Boldizsár**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző, cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2022.12.09.

.....  
Kovács Boldizsár

# Kivonat

Az élet számos területén szükség van arra, hogy az emberek igényeit kielégítően és az érdekeiket figyelembe véve tervezzenek meg közlekedési eszközöket, épületeket, kereszteződéseket vagy aluljárókat. Ehhez elengedhetetlen a járókelők mozgását ismerni, és a tervezek során ellenőrizni, hogy az épület helyiségei jól bejárhatóak legyenek, a tömegközlekedési eszközökön lehetőleg elférjenek egymás mellett az emberek, és a kereszteződésekben vagy aluljárókban ne alakuljon ki a forgalom miatt torlódás. A járókelők mozgásának valósághű szimulációja ezért a tervezek ellenőrzésének és minősítésének szerves része a jelenben és a jövőben is fontos részét fogja képezni.

A videójáték-iparban jelenleg a valósághű környezet kialakítása a cél a több éven át fejlesztett, nagy költséget idéző játékok tervezése során. Ehhez elengedhetetlen a környezet megtöltése éettel, és a szemnek természetesnek ható különböző járókelők, emberek, idegen lények vagy objektumok mozgatása. A videójáték-iparban a mozgás szimulációját kialakító algoritmusok hatékonysága és az elért eredmény arányának megfelelő kialakítása és megválasztása kardinális kérdés a véges számítási kapacitás miatt. Ezért nem csak a valósághű mozgás megvalósítása a fontos, hanem ennek egy hatékony megoldását megalkotni is feladat.

A dolgozatban a járókelők szimulációjára alkalmas program megalkotásának folyamatán megyek végig. A feladatot a témakör szakirodalmának áttekintésével és a meglévő megoldások megismerésével kezdtem. A szerzett ismeretekkel a pontos feladatomat leírtam és tapasztalati úton is ellenőriztem az újonnan szerzett információkat, majd felhasználva ezeket megalkottam egy - a feladatát ellátó - programot.

A programomat egyetemi környezetre terveztem, a program tesztelésének helyszínéül a Budapesti Műszaki és Gazdaságtudományi Egyetem I épületének 4. emeletét választottam, ezt lemodelleztem, és a szimulációban a járókelőket és azok mozgását felruháztam az egyetemi polgárok jellemzőivel. Végeredményül valósághű képet kaptam a járókelők mozgásáról az emeleten.

Az elkészített program továbbfejlesztési lehetőségeit is megvizsgáltam és használatára más alkalmazási területeket is kerestem (például tűzvédelmi terv).

# **Abstract**

In many areas of life, there is a demand for transport facilities, buildings, intersections or underpasses to be designed to meet people's needs and interests. This is why transport facilities, buildings, crossings and underpasses are designed to meet people's needs and interests. It is therefore essential to know how people move around and to ensure that the design of a building is easy to access, that public transport systems can accommodate people as well as possible, and that crossings or underpasses are not congested. Realistic simulation of pedestrian movements is therefore an integral part of the verification and rating of plans, both now and in the future.

In the video games industry, the current aim is to create a realistic environment for games with large budgets that are developed over several years. To achieve this, it is essential to bring the environment to alive, and to move different types of passers-by, people, alien creatures or objects that appear natural to the eye. In the video games industry, the efficiency of algorithms that simulate movement and the ratio of the results achieved must be designed and chosen appropriately, given the finite computing capacity. Therefore, it is not only important to achieve realistic motion, but also to create an efficient solution for it.

In this thesis I will go through the process of creating a program to simulate pedestrians. I started the project by reviewing the literature on the topic and looking at existing solutions. With the knowledge I gained, I wrote down my exact task and empirically verified the new information I had gained, and then used it to create a program - which would do the job.

I designed my program for a university environment, chose the 4th floor of the building I of the Budapest University of Technology and Economics as the location for testing the program, modelled it and simulated the pedestrians and their movements with the characteristics of university citizens. The result was a realistic picture of the movement of passers-by on the floor.

I have also investigated the potential for further development of the completed program and looked for other applications for its use (e.g. fire safety plan).

# 1 Bevezetés

Az élet alapvető és szerves részét képzi az, hogy tudunk tájékozódni és közlekedni. Tudatában kell lenni az emberek tulajdonságaival és viselkedésükkel, továbbá a normákkal ahhoz, hogy a többi emberrel együtt lehessen élni. Alkalmazkodni kell hozzájuk, és nyíltan sosem kimondu, együtt kell dolgozni velük. Ezek nélkül az együtt élés lehetetlen lenne. Egy mindenki által ismert példa erre a közlekedés.

Fontos, hogy az emberek legyenek egymásra tekintettel, ezekben segítenek a normák. Fontos ugyanúgy, hogy a helyzetfelismerő képességük is helyes legyen, ehhez figyelniük kell és alapvető ismeretekre van szükségük. Ahhoz, hogy a helyzetfelismerése helyes legyen az embernek, tudnia kell olyan alapvető dolgokat, mint hogy hol van, merre tart és kik vannak körülötte. Merre fele tud tovább haladni, korlátok, falak, lépcsők és sok más terepjektum milyen lehetőségekre korlátozza a továbbhaladását.

Megfelelően kell a járókelőnek megválasztania a sebességét, hogy a céljához eljusson. Sétálnia is elég, mert nem siet, állnia kell, mert piros a lámpa vagy liftben vagy villamoson van, esetleg futnia kell, hogy elérjen egy buszt. Ezen esetekben mind ismernie kell a környezetét, és saját maga tulajdonságait és korlátait. Nem tudunk tetszőlegesen gyorsan haladni, mert képtelenek vagyunk egy adott sebességnél gyorsabban futni. Nem tudunk minden kanyart belátni, minden sarkon túl ismerni a környezetet. Nem tudunk mindig beszállni a liftbe, mert nincs hely már számunkra, vagy tovább sem tudunk haladni, mert elfáradtunk.

Végtelenül összetett feladat az, hogy egy ember eljusson A pontból B pontba, miközben ezt a hétköznapokban maguk az emberek nem is érzékelik különösebben komplex problémának, pedig számos tény és ok alakítja mozgásukat és viselkedésüket. Informatikában ezek a problémák komplex feladatokká válnak.

Számos tényt és körülményt figyelembe kell venni ahhoz, hogy az emberek mozgásával kapcsolatban a valósághoz hasonlót sikerüljön lemodellezni. Nagyon sok kutatás és adat szükséges ahhoz, hogy az emberek mozgása végül természetesnek hasson és a valóságot minél jobban megközelítse, és hogy akár csak kezdetleges

megoldást, modellt készítsünk arra, hogy hogyan mozognak az emberek – a közlekedésben a járókelők.

## 1.1 Emberek mozgásának segítése

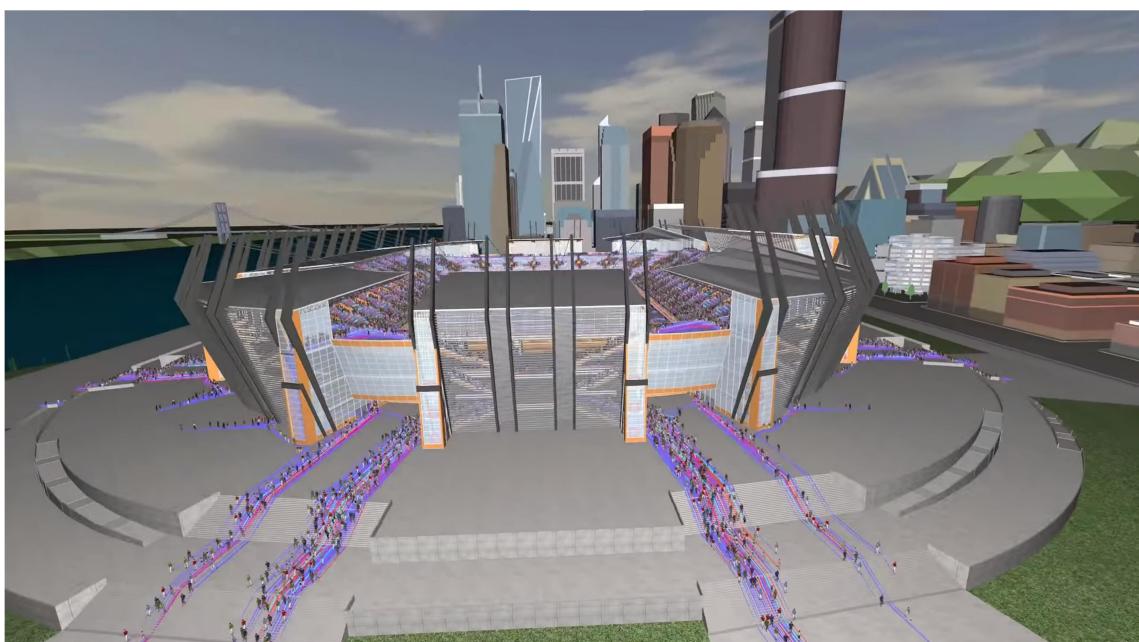
Amikor az emberek számára egy pillantás is elegendő, hogy felmérjék a közlekedési szituációt, miért szükséges bevonni az informatikát, amelyben ennek a problémának a megoldása nehezebb. Azért, mert gyakran nincs lehetősége az embereknek egy pillantást vetni a környezetre, esetenként a környezet sem létezik még. Azért fontos foglalkozni azzal, hogy helyesen lehessen szimulálni a járókelők mozgását, hogy a szimuláció futtatása során kiderüljön az, hogy ha egy tervben valami nem a legmegfelelőbb az emberek mozgását tekintve, akkor javítva azokat, az emberek élete kellemesebb, és komfortosabb lehet.

Nagyon sok tervezés előzi meg egy épület megépítését, egy villamos összeszerelését, vagy egy kereszteződés átalakítását. A tervezéshez sok idő szükséges, hogy a lehető legjobb lehessen a végeredmény és minél kevesebb probléma legyen a kész állapottal. Egy épület esetében nem csak az a fontos, hogy minél magasabb vagy épp impozánsabb legyen, hanem többek között az is, hogy a részletek is rendben legyenek. Például ne nyíljanak egymásba az ajtók, a lépcsőfokok mélységének és magasságának aránya megfelelő legyen, ne legyenek a szobák és a folyosók túl szűkek, vagy épp túl elnagyoltak. A tervezés végére minden legyen a rendeltetésének megfelelő méretű, ergonómikus elrendezésű, az emberi szemnek természetesnek ható és komfortos.

Amennyiben a szimuláció a járókelők mozgásáról helyes, a terveket ellenőrizni lehet, hogy tényleg elfér-e ott annyi ember, hogy tényleg elegendő-e harminc másodpercnyi zöld idő egy kereszteződésnél, hogy a forgalomszámlálást követően megbecsült maximális száz ember átérjen a gyalogos átkelőhelyen. Megvizsgálható, hogy egy lakást az ember várhatóan kellőképpen ki fog-e használni, a bejáratósága lehetséges-e, és ezek mind a másokkal való együtt élés esetén is igazak maradnak-e. Leszimulálható, hogy egy épületet tűzriadó esetén el tudja-e hagyni adott időn belül az épületben tartózkodók mindegyike maximális kapacitás esetén is.

## 1.2 Tűzvédelmi szempontok

A probléma megoldásának fontosságát mutatja, hogy az épületek terveit jóvá kell hagyni azon szempontból Magyarországon, hogy a tűzvédelmi szempontoknak megfelel-e. Sok dologban kell megfelelniük a terveknek, amelyek közül az egyik, hogy tűz esetén várhatóan mindenkinél sikerüljön elhagynia biztonságban az épületet. Ezt kisebb épületeknél, nagyjából 500 főig tűzriadó próbával is meg lehet vizsgálni. Nagyobb létszám esetén jellemző, hogy szükséges a terveknél is ezt már leszimulálni. Erre a Thunderhead Engineering cég Pathfinder [1] programját használják az iparban Magyarországon, mert ennek a programnak a terveket megadva, és az emberek paramétereit meg nem változtatva elfogadott a szimulációban elért eredménye a tűzriadónak. Nagy létesítmények, akár stadionok tesztelésére is alkalmas.



1. ábra Egy stadion tűzriadó próbájának szimulációja<sup>1</sup>

Kell számolni azzal, hogy a tervezett objektum kiknek készül, például betegeknek, idősebbeknek, óvodásoknak vagy szélesebb embereknek. A lépcsők az idősebbek számára fontos kérdés. A szélesebb emberek számára a tömegközlekedésen fontos a székek szélessége. Kórházak és egyéb közintézmények esetén az akadálymentes közlekedést kötelező megoldani. Ezekkel mind lehet és egyes esetekben kell is a tervezés során foglalkozni.

---

<sup>1</sup> <https://www.youtube.com/watch?v=c6unBZoY9Ag>

Egy olyan programon dolgoztam, ami a járókelők mozgásának szimulációjára alkalmas zárt terekben. Arra használható, hogy egy adott épületnek egy adott emeletéről információkat szerezzünk. Például a tervezett maximális kapacitás esetén, hogy zajlana le egy tűzriadó, vagy csak a hétköznapokban hol okoz fennakadást egy szükület, mely ajtóknál vagy mely sarkoknál.

Ehhez szükségem volt az emberekre általánosságban elmondható tulajdonságok összegyűjtésére, a különböző paramétereiknek az átlagos értékeiknek a meghatározására, továbbá a közöttük lévő kapcsolatok megismerésére. Gondolok paraméterek alatt a sebességükre, a korukra, a magasságukra, a szélességükre, a privát szférájuk kiterjedésére többek között. Kapcsolatok alatt például az embertömeg sűrűsége és az egyén mozgásának sebessége közötti kapcsolatra gondolok.

A program elkészítése során szükségem volt egy teszt környezetre, amiben vizsgálhatom, hogy a programom helyesen működik-e. A programomat az egyetemi polgárokra terveztem és ehhez az egyetem I épületének negyedik emeletét modelleztem le.

### 1.3 A feladat leírása

A feladatom az, hogy alkossak egy olyan programot, amely képes a járókelők mozgásának, azaz a gyalogos közlekedés leszimulálására.

Első feladatom a problémafelvetés és az irodalomkutatás. A kettőt szükséges együtt végezni, ugyanis a már meglévő, mások által írt munkákban a problémakör nehézségeire gyakran kitérnék. Ennek okán olyan feladatokat és problémákat kellett találnom a témán belül, amikkel a problémaköröm kibővíthetem. Ezt követően folytattam az irodalomkutatást, hogy a kibővített problémakörre találjak már létező megoldásokat, vagy csak segítséget ahhoz, hogy kiindulási alapot szerezzek. Mások munkáit olvasva a program egyes részproblémáinak megoldására próbáljak egyre többféle megközelítést találni, és ezeket lehetőleg hasonlítsam majd össze.

A szerzett ismeretek alapján a következő feladatom, hogy specifikáljam a program elvárt működését, majd tervezzem is meg. Tervezés közben gondoljam végig, hogy mi szükséges a program elkészítéséhez és a különböző tervezői döntéshelyzetekben indoklással hozzak lehetőleg helyes döntéseket.

A program megtervezése során feladatom, hogy teszteljem a megoldásom helyességét. Fontos volt, hogy a megoldandó problémákat több részre osszam, hogy elkülönítve, a tesztelés csak egy-egy egységre történjen egyszerre. Ezzel segítetem azt, hogy a program felépítése egységekre legyen bontva, és megfelelően legyenek a részei összeszervezve. Ezt mind a működőképességért, az áttekinthetőségéért és a jövőbeli továbbfejlesztési lehetőség fenntartása érdekében teszem meg.

A program elkészítését követően megvizsgálom a működését és tesztelem helyességét. A szimuláció paramétereit a valóságnak megfelelően megválasztva összehasonlítom, hogy az eddigi ismeretekkel összeegyeztethető-e az eredmény. Ha nem az elvárt eredmény kapom, átvizsgálom az esetleges hibák után kutatva a programom, vagy indoklom, hogy mi okozhatja az eltérést.

Program készítése során feladatom már létező algoritmusok részletesebb megismerése, és számomra specifikus algoritmusok kifejlesztése és létrehozása. Fontos, hogy ezek az algoritmusok ne csak gyorsak legyenek, hanem hatékonyak is. Hatékonyság érdekében hasonlítsak össze több megoldást is. Ezekhez az algoritmusokhoz feladatom olyan adatmodellt alkotni, amellyel könnyen lehet dolgozni, és a felépítése logikus. A programot feladatom úgy létrehozni, hogy lehessen paraméterezni, például ne csak egy előre betáplált területtel, alaprajzzal dolgozzon. Lehessen új alaprajzokat megadni a programban, és lehessen változtatni az emberek főbb paramétereit az adott környezetre jellemző értékekre. Mindehhez egy olyan grafikus felületet alkossak, amely alkalmas az adatok olyan megjelenítésére, hogy könnyen értelmezhetőek legyenek, és a program működése szemmel látható legyen.

## 1.4 Irodalomkutatás és a feladatkör kiterjesztése

### 1.4.1 Programozási környezet megválasztása

Először a programozási nyelvet választottam meg. Olyan nyelvet kerestem, ami a feladathoz megfelelő és az iparban elterjedt a mai nap is. Egy széles körben elterjedt, számos iparágban [2] jelenleg is használatos nyelvet választottam így, amiben lehetséges az objektum orientált szemléletet követni. Az algoritmusok hatékony és gyors működése érdekében szükségesnek tartottam egy gyors nyelvet, hogy ez a tényező biztosan ne legyen hátráltató tényező az algoritmusok gyorsaságában. Ezért döntöttem a C++ [3] nyelv mellett.

A megjelenítéshez kellett választanom egy grafikus könyvtárat, és a Simple DirectMedia Layer 2-t [4] (SDL2) választottam. Mint grafikus könyvtár, az alapvető funkciókat ellátja, képes egyszerűbb geometriai alakzatok kirajzolására. Működése egyszerű, hatékonysága különösebben ennél a feladatnál nem számít, mert nem szükséges komoly látványos grafikát megvalósítanom. A feladathoz szükséges adatok értelmezését nagyban segítő reprezentáláshoz böven elegendő eszköztárral rendelkezik. Továbbá a felhasználói bemeneteket, mint az egér mozgatását és a vele történő kattintást és görgetést, továbbá a billentyűzet minden gombját is lekezeli. Ezért megfelelő a feladathoz.

#### **1.4.2 Útvonal keresés**

Előnyben részesítve a választott nyelvet, elkezdtem keresni a témaban munkákat, cikkeket, dolgozatokat, hogy a program tervezésénél minél több problémára tudjak figyelni és a szerzett ismeretek alapján azokat megoldani. Kiindulásként szükséges volt keresnem olyan, már létező megoldásokat, ahol az útvonalkereséssel már évek óta foglalkoznak. A játékiparban már évtizedek óta foglalkoznak azzal, hogy különböző egyedek, emberek vagy objektumok a gép által vezérelve eljussanak A pontból B pontba.

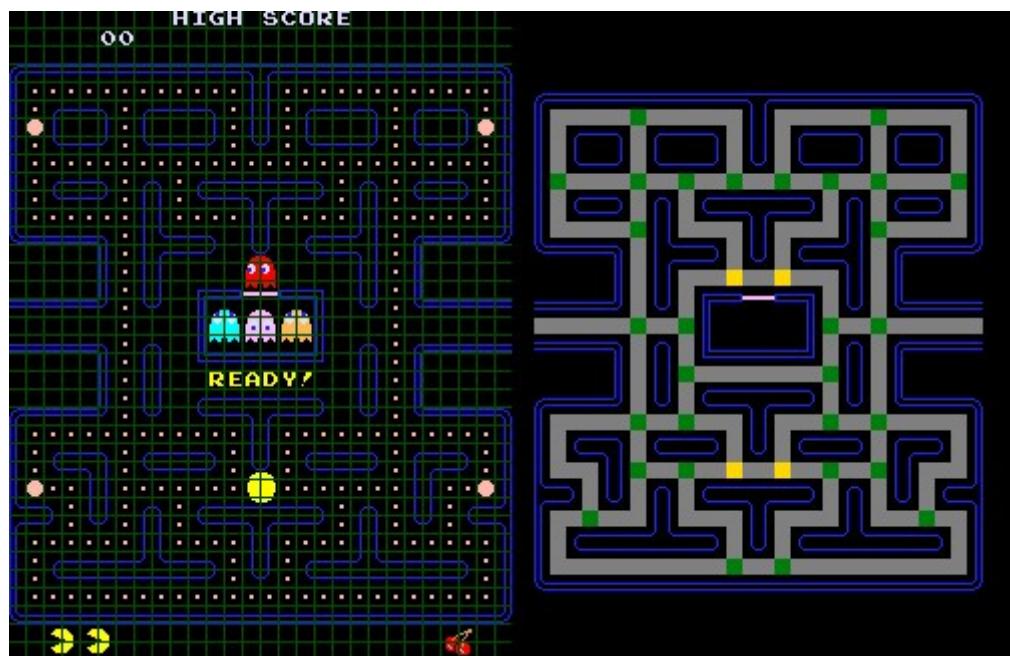
Egy régi példát említve. Több mint negyven éve alkották meg az első PAC-MAN-t [5], amelyben a játékos a játéktéren szellemek próbálták elkapni. Ebben a példában a szellemekből négy volt és mindegyiknek saját logikája volt. A szellemek útvonalkeresésénél is használhatták a Dijkstra algoritmust. [6] Az algoritmus alapját a későbbiekbén felhasználtam, így a teljesség kedvéért kifejtem.

#### **1.4.3 Dijkstra algoritmus**

Az algoritmus a legrövidebb út problémára megoldás. Edsger W.Dijkstra (1930 - 2002) holland matematikustól származik. Az algoritmushoz szükséges egy élsúlyozott gráf és csak abban az esetben működik helyesen, ha minden él súlya nemnegatív. Jelen példámban nincs erre precedens, így alkalmas megoldás az útvonalkeresésre, továbbá összefüggő is a gráf, így kevesebb problémát kell megvizsgálnia az algoritmusnak.

A játéktér tekinthető egy négyzetrács hálónak, jobban megnézve, ez a háló szabad mezőiből lehet csoportokat alkotni, amelyeket élekként és csúcsokként gráfba lehet szervezni. Az algoritmus mindenkor a legrövidebb útvonalat adja vissza a gráfban, és

futásideje is kedvező. Az algoritmus egy gráfot használ keresési térként, és két csúcsát kezdő- és végpontként. Az algoritmus kezdetben a kezdő csúcs minden szomszédos csúcsát egy értékkel látja el. Az csúcsba vezető élek súlyát veszi a szomszédos csúcsok értékének. Ezt követően kiválasztja azt a csúcsot, amelynek az értéke a legkisebb és még nem vizsgálta meg. A kiválasztott csúcs szomszédjainak az értékét úgy adja meg, hogy nem csak a csúcshoz vezető él súlyát veszi figyelembe, hanem azt is, hogy magának a csúcsnak mekkora az értéke. Azaz a csúcshoz vezető út élei súlyának mennyi az összege a kezdőpontból.



2. ábra PAC-MAN pályájának négyzethálós felbontása<sup>2</sup>

Fontos, hogy annak a csúcsnak amelynek értéket ad, ha már van értéke, és az kisebb vagy egyenlő az új értékhez képest, akkor nem bírálja felül, és az értékét nem változtatja meg. Ellenkező esetben felülírja az új értékkel. Ez az eset azt jelenti, hogy talált egy rövidebb utat a kezdőpontból az adott csúcsig. A csúcsok kifejtése, és az algoritmus futása addig tart, amikor a végpont csúcsát fejti ki. Ekkor ugyanis a végpont csúcsának értéke adja meg, a legrövidebb út hosszát a két csúcs között. Azért fontos, hogy ne akkor álljon le az algoritmus, amikor először felfedezi az végpont csúcsát, mert még utána lehet, hogy talál egy rövidebb utat a végpontig. Viszont, amikor a végpont csúcsát fejti ki az algoritmus, akkor az azt vonja magával, hogy más kifejtésre váró

---

<sup>2</sup> <https://gameinternals.com/understanding-pac-man-ghost-behavior>

csúcs értéke legalább annyi, mint a végpont csúcsa. Ekkor mivel nincsen negatív él, ezért a kifejtendő csúcsokból már biztosan nem lehet rövidebb utat találni a végpontba, így a feladatát az algoritmus elvégezte, és leállhat.

Fontos, hogy ezen algoritmus használata során felfedezett csúcsokhoz el kell tárolni az értékükön kívül a kiszámításuknak módját is valamilyen módon, különben egyetlen számértéket visszakapva a legrövidebb utat nem tudja meghatározni, csak az út hosszát. Például egy megoldás erre, hogy a felfedezett csúcsokhoz eltárolja az értékükön kívül azt is, hogy melyik csúcsból lett kifejtve a végső értékük. Ebben az esetben a végpontból kiindulva meg kell vizsgálni azt a csúcsot, amiből ki lett fejtve az értéke, és azt a csúcsot is, amelyből az ki lett fejtve, és így tovább egészen a kezdőpont csúcsáig. Bizonyítás nélkül, de igaz, hogy a legrövidebb út a két csúcs között így előáll a visszafele haladva megvizsgált csúcsokból.

#### 1.4.4 A\* algoritmus

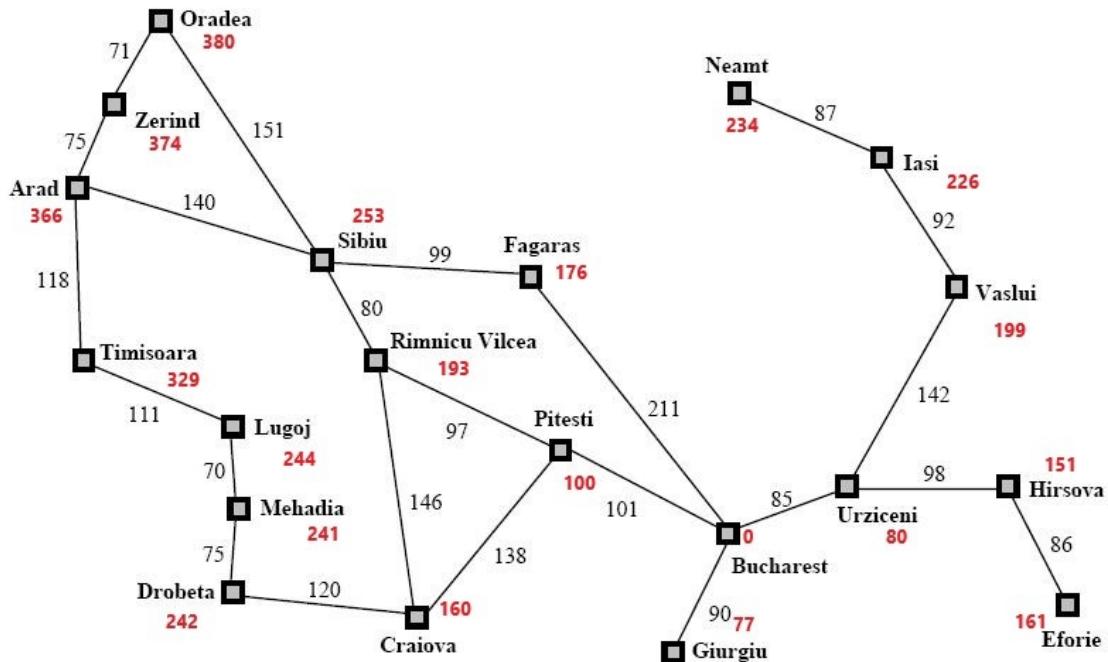
A Dijkstra algoritmus nagyon közel áll egy jelenleg is elterjedt, a játékiparban standardnak tekinthető A\* (kiejtve „a csillag”) algoritmushoz. [7] Ennek az algoritmusnak több átdolgozott verzióját is használtam a programomban végül. A Dijkstra algoritmus publikálását követően tizenkettő évvel később publikálták az A\*-ot először, mint a Dijkstra algoritmus kiegészítése. Az algoritmus abban több, és a gyakorlatban sokkal hatékonyabb, hogy a csúcsokhoz rendelt értéket új heurisztikával adja meg. Az A\*-ot akkor lehet használni, ha létezik olyan költségbecslést végző függvény, amely a végpont csúcsáig tetszőleges csúcsból képes olyan távolságot megbecsülni, amely sosem lehet több, mint a tényleges út hossza abba a csúcsból. A heurisztika a következő. A csúcs értékének nem csak az odáig vezető út élei súlyának összege adja meg, hanem külön hozzáadja ehhez a becslés végző függvény értékét a csúcsra nézve. Az onnan kifejtett csúcs értékét nem az előző csúcs értéke és az összekötő él súlya adja majd meg, hanem az előző csúcs értékének a becslést végző függvény értékének hozzáadását megelőző érték, az összekötő él súlya és a becslő függvény értéke az új csúcsra adja majd.

Az algoritmus minden csúcs kifejtése során minden szomszédos csúcsot kifejt, akkor áll le, amikor először felfedezi a végpont csúcsát. Ahhoz, hogy elég felfedezni a végpont csúcsát a leálláshoz, elengedhetetlen egy jó távolság becslést végző függvény. A heurisztika és az algoritmus végének módosítása hatékonyabbá teszi az

útvonalkeresést nagyságrendekkel a Dijkstra algoritmushoz képest. Gyakori megoldás megválasztani költség függvények a két csúcs valós távolságát. Erre lehetőség van olyan esetekben, amikor a gráf csúcsai valamilyen teret írnak le. Ilyen tér például lehet egy olyan gráf, ahol a csúcsok városok és az élek a városokat összekötő út hossza. Ekkor a távolságbecslő függvénynek alkalmas megválasztani a városok távolságát légvonalban.

PAC-MAN példáját véve az A\* nem használható könnyen, mert a játéktér két pontja között nem lehet könnyen helyes becslést végezni hasonló módon. Ugyanis a vízszintesen középen elhelyezkedő átjáró a játéktér egyik oldalát összeköti a másik oldalával. Ami a képernyőn egy egység távolságnak tűnik, az az átjáró miatt lehet nulla is. Emiatt olyan távolság becslő függvényt kéne megválasztani ahhoz, hogy az algoritmushoz megfelelő függvény legyen, amely ezt figyelembe veszi.

A városok és az azokat összekötő utak, illetve a PAC-MAN között a különbség az, hogy a legrövidebb út a térképnél az egyenes, de a PAC-MAN-nél nem szükségesen. Megjegyezném, hogy az olyan A\* algoritmus, amely a végpont csúcsát kifejtve áll le, és a becslést végző függvény csak nullát ad vissza, az a Dijkstra algoritmus.



3. ábra Szemléletes bemutatása a város térkép példának, az utak hosszával és a városok légvonalbeli távolságával Bucharest-től<sup>3</sup>

Azért fejtettem ki a két algoritmust részletesen, mert a kutatásom során fontos volt a megismerésük, és a programomban közülük az A\* algoritmust számos alkalommal használom fel. Azért elterjed az iparban ez az algoritmus, mert ha lehet helyes költségbecslő függvényt találni, akkor az algoritmus hatékonysága kiugróan jó. A legrövidebb út megoldására alkalmas algoritmusok időbeli komplexitása a csúcsok és az élek számával függenek össze. Az A\* algoritmus komplexitása viszont a heurisztikától nagyban függ, így csak konkrét eseteknél lehet ezt meghatározni.

#### 1.4.5 Helyszín átalakítása gráffá

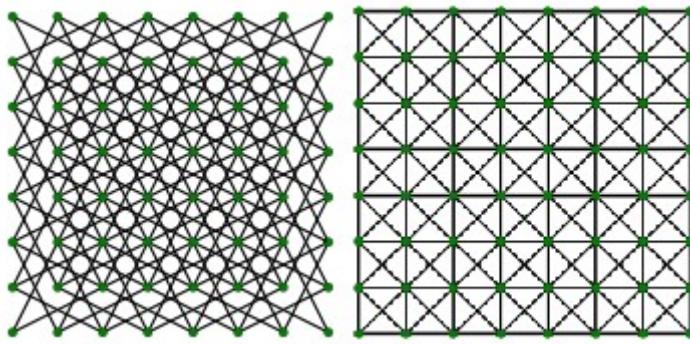
Az útvonalkereséshez kell egy gráf, amin lehet az algoritmust futtatni lehet. Ahogy a példákkal mutattam, nem minden esetben lehet kihasználni az A\* heurisztikáját, viszont az ember mozgásánál ezzel nincsen probléma. Ahogy a valóságban, úgy a gráfban is a legrövidebb út az egyenes lesz. A különböző komplexitású problémákra különböző gráfokat sikerült találnom.

Az A\* algoritmusnak megfelelő gráf például a sakkban a király lehetséges lépései tartalmazó gráf, a királygráf. A sakkban a játéktér véges, és a király helyzete

---

<sup>3</sup> <https://dzone.com/articles/from-dijkstra-to-a-star-a-part-2-the-a-star-a-algo>

csak diszkrét és véges sok lehet. Összesen hatvannégy mezőn állhat, és az A\* algoritmus is helyesen alkalmazható, mert a király esetén elmondható, hogy a legrövidebb út, az egyenes. Kifejezetten nem mondható ez el a király helyett a huszárról vagy a futóról például. A király esetében van hatvannégy csúcs a gráfban, és egy csúcs legalább három, de akár nyolc másik csúccsal is össze van éellel kötve, futó esetében nincs hatvannégy csúcsa a gráfnak, mert csak a saját színén maradhat és a távolságát egy másik mezőnek nem a fizikai távolsága adja meg. Ezt a véges mezőből álló játékteret annak ellenére megvizsgáltam, hogy nem írja le a valóságot. Egy ember tetszőlegesen sok helyen állhat egy szobában, nem csak pontosan hatvannégy helyen vagy mezőn.



4. ábra Egy huszár és a király által lehetséges lépések gráfja<sup>45</sup>, a nevük huszárgráf és királygráf

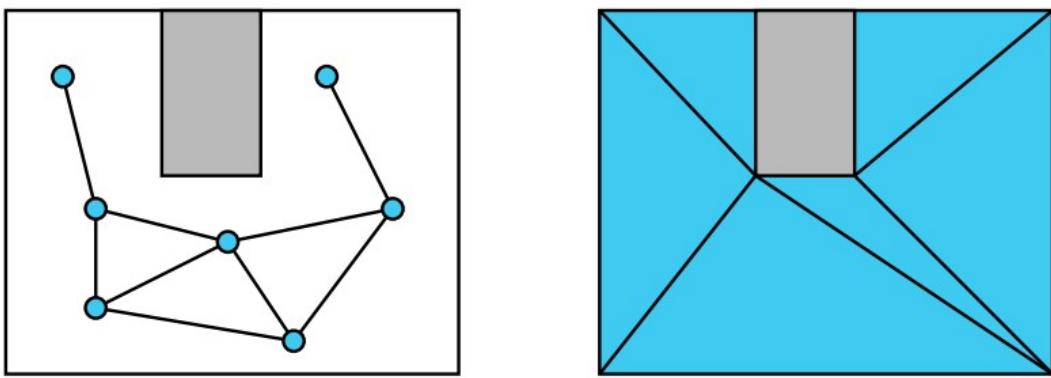
#### 1.4.6 Felületeket lefedő gráfok

Komplexebb probléma, amikor a gráf egy épület emeletének, vagy csak egy szobának a bejárhatóságát úgy írja le, hogy minden pontjára alkalmazható és nem csak véges számú pontra. Ez azt jelenti, hogy a bejárható tér még lehet véges, de a helyzete az embereknek már nem, így lényegében végtelen csúcsa lenne a helyszínt leíró gráfnak. Két nagyban különböző gráfot találtam ennek a végtelennek tűnő problémának a megoldására. A két gráf magyar nevére nincs általánosan elfogadott szó a szakirodalomban, így szabad fordítással a nevük útpont gráf és navigációs háló. [8]

---

<sup>4</sup> [https://en.wikipedia.org/wiki/Knight%27s\\_graph](https://en.wikipedia.org/wiki/Knight%27s_graph)

<sup>5</sup> <https://hu.wikipedia.org/wiki/Kir%C3%A1lygr%C3%A1f>



5. ábra Az útpont gráf és a navigációs háló

#### 1.4.7 Útpont gráf

Az útpont gráf azt a problémát kívánta megoldani, hogy ne végtelen csúcsa legyen a helyszínt leíró gráfnak. A helyszín végtelen pontja közül csak keveset tart meg, és a megmaradt csúcsok viszont elnagyolt, de jó képet adnak arról, hogy miként lehet bezárni a teret, hogy miként lehet eljutni a helyszín bármely pontjára. Lényegében ezeket a pontokat követve el lehet jutni az emelet esetében tetszőleges szobából, egy másik szobába. Figyelemmel van arra, hogy a gráf élei bezárhatóak legyenek, azokon végig sétálva falba, vagy más terepobjektumba ne ütközzön az ember. A csúcsai arra alkalmasak, hogy közük kiválasztva a kezdőponthoz a legközelebbi, és kiválasztva a végcélhoz is a gráf legközelebbi csúcsát, ki lehessen számítani a legrövidebb útvonalat. A legrövidebb útvonalat a gráfra számolja ki, nem a tényleges útvonalra.

Ez a megoldás komplexitása abban rejlik, hogy a gráf csúcsait megfelelően kell megválasztani ahhoz, hogy használható legyen a gráf útvonalkeresésre. Ha túl ritkák a csúcsok, lehet, hogy a szobában nincs is egy csúcsa sem a gráfnak. Ha túl sűrű, vagy nem egyenletesen vannak megválasztva a csúcsok, akkor a számítási idő fog feleslegesen megnövekedni. A komplexitása abban is megmutatkozik közvetetten, hogy ha nincs komolyabb energia fordítva a gráf kialakítására, és az emberek mozgásának kialakítására, akkor nem lesz valósághű a szimuláció. Kevés idő ráfordítása esetén, mindenki csak az éleken haladna a helyszínen. Az éleken haladás emberek mozgását a hangyákéra torzítaná. Egymást követve vagy szembe, de lényegében egy sorban haladnának. Az emberek kihasználják a teret, nem egy sorban haladnak általában. A nem megfelelő gráf megalkotása esetén az útvonal lehet nagyon szögletes, nem természetes, és se nem a legrövidebb, se nem az ember szokásos útvonala. Ezért komoly feladat az, hogy egy ilyen tökéletes gráfot megalkosson az ember.

Sajnos van egy problémája az útpont gráfnak. A gráf csúcsainak száma véges, így nem lehet minden pontjára a helyszínnek eljutni egyértelműen, csak a gráf csúcsaira, esetleg éleire. Amikor elindul a programban az ember, akkor először el kell jutnia a gráf egyik csúcsához, és amikor a gráf élein végighalad, akkor az utolsó csúcstól el kell találnia valahogyan a saját céljáig. Ez azt vonja magával, hogy két útvonalkereső algoritmust is kellene használni a programban. Természetesen ezen megoldás hibáinak a nagy részét az képzi, hogy automatikusan generálni ilyen hálót komplex probléma. Ha sikerül megfelelőt generálni, akkor kifejezetten hatékony, de az emberek természetes mozgatásának megvalósítása továbbra is kérdéses marad.

#### 1.4.8 Navigációs háló

A másik létező megoldás a problémára a navigációs háló, angol nevén Navigation Mesh. Ez a helyszín végtelen csúcs problémáját úgy oldja meg, hogy a helyszínt, a teret háromszögekkel, vagy egyéb konvex sokszögekkel fedi le. Ekkor lesz egy háromszögek éleiből álló háló, és egy plusz információ halmaz, ami az, hogy a gráf mely élei alkotnak háromszöget, és/vagy hogy mely élei határozzák meg a tér határait. A helyszín minden bejárható pontja valamelyik háromszög élére vagy belséjébe esik. A háromszögek nem lapolódnak át, így könnyű meghatározni, hogy a háromszögek melyikébe esik az adott pont.

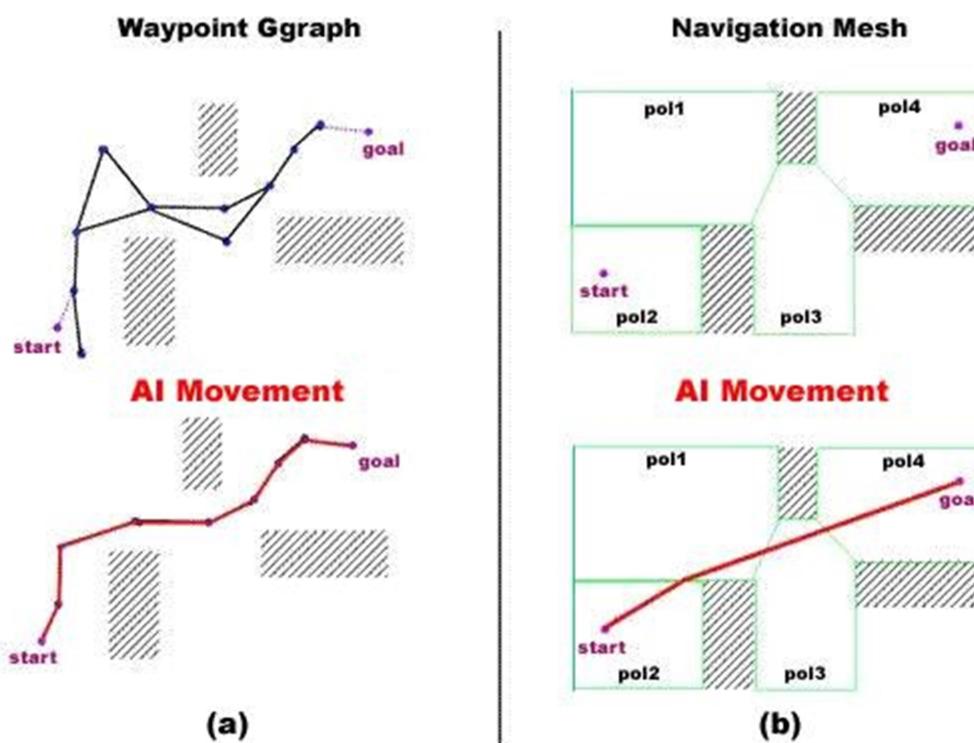
Az útvonal keresése ebben a megoldásban úgy lehetséges, hogy a kiindulási csúcshoz megtalálja azt a háromszöget, amiben benne van. Ennek a háromszögnek a csúcsaiból, vagy csak az egyikéből az A\* algoritmust lefuttatva eljut azon háromszög csúcsáig, amelyben benne van az elérni kívánt végcél.

A navigációs hálónál ugyan úgy nehézség, hogy a hálót létre kell hozni. Lehet például mindenkor csak egy olyan szakaszt behúzni az alaprajznak tekinthető gráfba, amelyre igaz, hogy két olyan szakaszt köt össze, ami részét képzi az alaprajznak és van egy közös csúcsuk, továbbá az alaprajz egyetlen másik csúcsát. Ahogy behúzta az új szakaszt, elmenti a többi behúzott szakasszal, és azt az alaprajz részének tekinti onnan, és az előbb említett két élt kitörli. Mivel minden egyes él behúzása után kettőt töröl, ezért lépésenként csökkennek a módosított alaprajz éleinek száma. Addig kell behúzna így szakaszokat, amíg egy háromszög marad csak. Akkor az algoritmus leáll, és az elmentett behúzott éleket hozzáadja a gráphoz. Ekkor a gráf éleiből háromszögeket képez és kész van a navigációs háló.

Ez a háló azért jól használható, mert szomszédos konvex sokszögekből épül fel. Nem szükséges háromszögekkel dolgozni, lehet téglalapokkal vagy más sokszögekkel is, csak az a lényeg hogy konvex legyen. Ezt a tulajdonságot kihasználva síkidomokon belül az egyes pontok közötti út, az a két pontot összekötő szakasz. Ha több síkidomot is érint az útkeresés, mert nincsenek egy síkidombban a kezdő és végpontok, akkor a síkidom láncon keresztül kell az útvonalat kialakítani.

#### 1.4.9 Legrövidebb útvonal és útvonal kiegynésítése

Komoly problémát jelent az útvonal kiegynésítése a legrövidebb út meghatározásában, mind a kettő megoldás esetén. Itt bonyolult számítást igényelhet nem csak a háromszögek oldalain, mint gráfon lefuttatni az A\*-ot, hanem ki is egyenesíteni az útvonalat. Az útpont gráfban csak a gráf pontjaival képes dolgozni, ami csak véges lehetőséget biztosít, így ritkán adja vissza a legrövidebb utat.



6. ábra Az útpont gráf és a navigációs háló útvonalának összehasonlítása

A programomban a navigációs háló megvalósítása, és használata mellett döntöttem. A generálható útvonal sokkal jobban hasonlít a valóságoshoz a navigációs hálóval, mint az útpont gráffal. Az útvonal generálása lassabb, mint egy tökéletes útpont gráfban, de a természetes útvonalra törekszem, hogy az emberek mozgása hiteles

lehessen. A navigációs hálót a mai napig használják a játékiparban, különböző játék motorokban, például a Unity-ben [9] a gép által vezérelt entitásokat ennek segítségével navigálja a rendszer.

A generált útvonalat lehet végtelenségig finomítani, természetessé tenni. Nem csak élek mentén lehet haladni, hanem különböző görbék mentén is, de erre a szakdolgozatomban nem térek ki.

#### **1.4.10 Emberek mozgásának alapjai**

Most, hogy tudatában vagyok annak, hogy milyen módon tudok útvonalat generálni, fontos a kiszámított útvonalat egy embernek végig is sétálnia. Ennek okán az emberek mozgásával kapcsolatos információkat kerestem. A programomban fontosnak tartom, hogy az emberek ne pontszerű lények legyenek, és a sebességük ne legyen se lassú, se gyors, hanem amit várhatóan a valóságban is megválasztanának sebességül.

#### **1.4.11 Emberek szélessége**

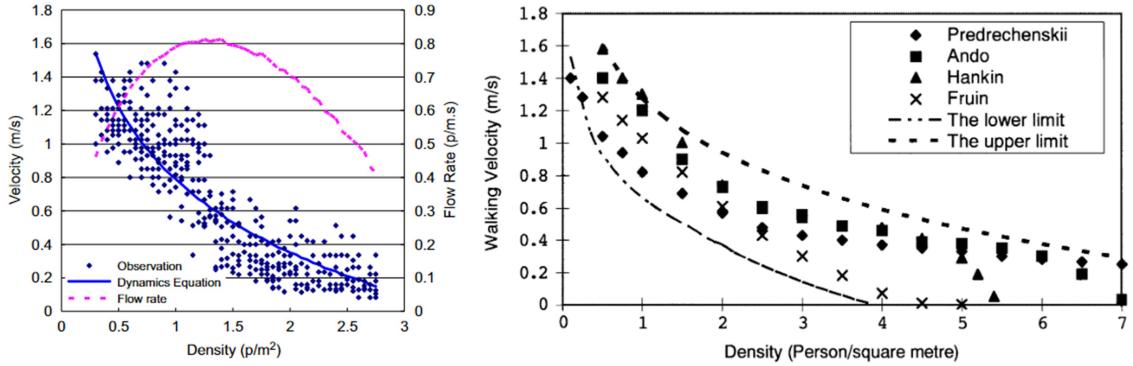
Különböző kutatások más-más eredményeket publikáltak, így nehéz megmondani, hogy mi az igaz. Az vállszélességét vettem alapul az ember szélességének. Ez az érték a nőknél és a férfiaknál eltérő, és az évtizedek alatt szemmel látható mértékben nőttek ezek az értékek az elmúlt hatvan évben. Az Amerikai Egyesült Államokban végzett felmérések szerint [10] az 1960-as években a felnőtt nők vállszélessége átlagosan 35.3 cm széles volt, a férfiaknak kicsivel több, 39.6 cm volt ez az érték átlagosan. Ezen értékek a 90-es évek elejére a nőknél 36.7 cm-re, a férfiaknál 41.1 cm-re növekedett. Egyértelmű különbség van köztük.

A megoldásomban, a kutatásokból szerzett információk alapján 30 cm és 50 cm közötti vállszélességű emberekkel fogok dolgozni. Más országok adatait ezen határok közé feltételezem. Egyetemi körülményeket tervezek szimulálni, ahol az emberek viszonylag nyugodtak, mozgásuk során nem futnak, hanem főképp csak sétálnak. Egyetemi környezet okán nem számolok gyermekek vállszélességével, mert az egyetemen szinte csak felnőtt ember fordul meg, nyíltnapokat leszámítva. És ezért nem számolok azzal sem a programom tervezése során, hogy valaki fut, mert ritkán tapasztalható csak az, hogy valaki a gyors, ütemesebb sétánál gyorsabban haladna.

### **1.4.12 Emberek mozgási sebessége egyedül és tömegben**

Utána néztem, hogy milyen sebességgel sétálnak az emberek. Az egyetemen hasonlóan viselkednek a mozgásuk szempontjából az emberek, mint az utcán. Ismerősökkel, vagy a csoporttal együtt tartanak, viszont ismeretlenektől próbálnak megfelelő távolságot tartani. Ennek okán mikor egy előadásnak vége, és egyszerre próbálja elhagyni akár kétszáz diák egy termet, akkor az egyén sebessége lelassul. Ugyanis várakozni kell másokra, és a kölcsönös távolságtartást sem lehet megtartani. Ezért a többiek mozgását folyton figyelni kell, mert a bizonytalanság nagyobb ilyen helyzetekben, mikor ritkábban vannak az emberek. A bizonytalanság esetén lassabb sebességet választunk meg, kisebb lesz a lépés hosszunk, és több energiát fordítunk a helyzet felismerésre. A lépéshossz rövidebbnek választása a másokra lépés elkerülése érdekében történhet, a helyzet felismerése, a tömeg megkerülése meg szintén valamivel több energiát igényel, mint ha egyedül sétálna az ember az utcán.

Az egyik legfontosabb paraméternek néztem így utána, hogy az embereknek mekkora az átlagos sebessége adott sűrűségű tömegben, milyen kapcsolat van a két érték között. Kutatási eredmények itt is sajnos eltérő számértékeket hoztak eredményként, mert más-más környezetben végezték a méréseket. Fontos paraméter, hogy van-e az embereknél valamilyen táska, esetleg bőrönd, vagy valamilyen olyan dolog, ami mozgásukat lassítja, és mások mozgását akadályozza. [11] Ennek okán, ha egyedül mozog az illető, haladhat akár 2 m/sec-mal, de akár csak 1.6 m/sec-mal is. A tömegben szintén eltérő eredményeket kaphatnak különböző mérések során. Az ember tömeg 1 ember/m<sup>2</sup> sűrűség esetén az egyén sebessége 0.7 m/sec és 1.3 m/sec között várható. Ez az intervallum 3 ember/m<sup>2</sup> esetén már csak 0.2 m/sec-tól 0.8 m/sec-ig tart. Egészen 7-8 ember/m<sup>2</sup>-ig lehet számolni azzal, hogy még a tömeg lényeges mozgást végez.



**7. ábra A járókelők sebessége és az egy négyzetméterre jutó emberek száma közötti összefüggés grafikonjai: különböző kutatások során más-más eredményeket kaptak [12]**

Az egyetemen nincs abból probléma, hogy valamire várakozni kell, de haladni is kell. Amíg vannak a teremben, addig nem kell bemenni oda. Ha el akarjuk hagyni az épületet, vagy a termet, azt szabadon megtehetjük. Mindig halad a tömeg, maximum a szűkületek szabnak határt a sebességnek. Kevés szituációban lehet valóban érdemes várakozni, például ha a lift használatáról van szó vagy egy ajtót kell kinyitnunk. A liftek előtt gyűlő tömeg ritkán éri el azt a kritikus szintet, hogy komolyan zavaró, és kifejezetten akadályozó legyen jelenlétéük.

#### 1.4.13 Vézhelyzet kezelése

Programomat nem tervezem arra, hogy váratlan okokból nem lehet valahol haladni, például leomlott a mennyezet vagy pánik tör ki az emberek között. Tűz esetén sem tör ki pánik az emberek között. Ezt rengeteg épülettúzból levont tapasztalat alapján ki lehet kijelenteni. [13] Az emberek ugyan úgy higgadtak maradnak, és nagy részük fel sem fogja kezdetben vagy el sem hiszi, hogy baj van. Mikor realizálódik bennük, hogy tényleges a veszély, akkor sem vesztik el higgadtságukat és próbálják a megfelelő lépéseket megtenni. Nem tapossák el egymást, hanem kifejezetten figyelnek másokra és segítenek is másoknak, ha rászorulnak. Kifejezetten ritkán kerül sor arra, hogy pánik törjön ki az egész épületben.

Actions	Percent of Population				
	First	Second	Third	Fourth	Fifth
Dressed	16.8	11.6	6.5	—	—
Opened door	15.9	11.7	6.7	3.4	—
Notified roommates	11.6	3.0	—	—	—
Dressed partially	10.1	7.5	4.5	—	—
Looked out window	9.7	5.7	—	—	—
Got out of bed	4.5	—	—	—	—
Left room	4.3	5.4	8.1	2.4	2.0
Attempted to phone	3.4	3.6	—	2.8	—
Went to exit	2.5	10.3	9.5	16.1	6.7
Put towels around door	1.6	2.5	3.0	6.8	7.7
Felt door for heat	1.3	2.3	—	—	—
Wet towels for face	1.3	3.7	6.3	4.6	7.9
Got out of bath	1.1	—	—	—	—
Attempted to exit	1.1	3.0	5.8	4.3	—
Secured valuables	—	6.8	4.3	—	—
Notified other room	—	3.4	2.2	—	—
Returned to room	—	—	3.9	8.4	4.1
Went down stairs	—	—	3.9	5.4	21.3
Left hotel	—	—	3.4	2.6	2.0
Notified occupants	—	—	3.0	—	—
Went to another exit	—	—	—	3.6	4.8
Went to other room	—	—	—	3.6	3.6
Went to other room/others	—	—	—	3.4	8.7
Looked for exit	—	—	—	2.4	—
Broke window	—	—	—	—	4.3
Offered refuge in room	—	—	—	—	1.8
Went upstairs to roof	—	—	—	—	2.9
Went to balcony	—	—	—	—	1.8
Other	14.8	19.5	28.9	30.2	20.4
Total (percent)	100.0	99.1	96.9	90.4	79.6
Number of guests	554	549	537	501	441

8. ábra Táblázat az emberek cselekedeteinek sorrendjéről egy szállodai tűz esetén

Arra viszont tervezem a programomat, hogy az emberek egymástól tartsanak kellő távolságot, és engedjék el egymást, ha a normák úgy diktálják. Legyenek figyelmesek arra, hogy ők éppen a teremből kijönnek, azaz elsőbbségük van, vagy a terembe terveznek bemenni, így elsőbbséget kell adniuk.

Fontos tényező az is, hogy az emberek a mozgásuk közben, ha befordulnak egy sarkon, akkor nem ismerik a sarkon túli környezetet, továbbá a sarokhoz közeledve sebességek le is csökken. Ennek megoldása komplex feladat, és csak próbálkozást teszek a programomban ennek figyelembe vételére.

## 2 Saját munka bemutatása: Első program

Egy program elkészítésénél első lépésként specifikálni kell az elvárt működését, majd ezt követi a megtervezése és lefejlesztése. Elő kívánom segíteni azt, hogy minél gyorsabban menjen a program fejlesztése, és minél kevesebb hiba lehessen benne. Ne kelljen a tervezést követően a fejlesztés során új funkciókkal bővíteni vagy új esetekre felkészíteni a programot, mert akkor már késő ezzel foglalkozni. Időigényesebb és gyakran a minőségre is rossz hatással van. Ezért a „nagy könyv” szerint próbálom a programomat fejleszteni. Ennek első lépése, hogy specifikációt készítsek. Azaz írjam le, mit kell tudnia majd a programomnak, mit kell megvalósítanom a tervezés során.

### 2.1 A program elvárt működése

Kutatómunkát elvégezve sok problémával sikerült találkoznom a témán belül. Ezeket összegyűjtve ki tudom alakítani azt, hogy mit kell megvalósítanom és mikre kell külön figyelmet fordítanom. A következőket várom el a programtól. Legyen működőképes és használható és tilos váratlanul leállnia vagy a külső beavatkozásokra nem reagálnia. Alapvető követelmény egy programnál, hogy megbízható legyen a működése, de mint követelmény, fontos megemlíteni.

Legyen a program felkészítve arra, hogy valamilyen módon, például a beviteli eszközök segítségével vagy fájlból beolvasva képes legyen új helyszín alaprajzával dolgozni. Legyen képes létrehozni olyan gráfot, amelyet használni képes az kutatómunka során kifejtett algoritmusok valamelyikével, vagy egy továbbfejlesztett változatával. Lehetőleg optimális és a lehető legrövidebb utat számítsa ki emberenként a program, ami elegendő, hogy csak pontok sorozata legyen. Legyen grafikus megjelenítése, ami a szimulációt úgy mutatja be, hogy az adatokat szemmel könnyen felfoghatóan ábrázolja. Lehessen különböző paraméterű, például különböző szélességű és sebességű embereket használni, de elegendő csak valós helyzetekre felkészíteni a szimulációt. Nem kell dolgozna váratlan eseményekkel, mint hogy kidől egy fal a helyéről és elállja az utat, ahogy az egyetemen rohanó emberekkel sem. Legyen szempont a program hatékonysága is.

## 2.2 Emberek ütközésének elkerülése

A valósághű szimuláció megalkotása nehéz feladat. Hatékony megoldást találni rá még bonyolultabb. A feladatot jobban át kívántam látni saját magam is, nem csak papíron olvasott tapasztalatokra alapozni a programomat. Először egy szűkebb problémakört kívántam megoldani, és ehhez egy megkötésekkel megalkotott környezetben dolgoztam.

Azt a problémát kívántam megoldani, hogy az emberek miként fognak nem összeütközni a programban. Ugyanis ennek a megoldása még teljes mértékben kérdéses. Mi az, amitől a szimulációban az emberek nem csak egyedül lesznek képesek természetes képet adni, hanem tömegben is. Jelenleg a használni tervezett A\* algoritmus csak a legrövidebb útvonalat biztosítja. Nem számol azzal, hogy az adott útvonalon hányan kívánnak szintén végighaladni. Az esetleges szembe fogalommal is számolnia kellene a programnak, továbbá hogy ezt milyen szinten kell figyelembe venni.

Fontos, hogy inkonzisztens állapotba ne lépjen a szimuláció sosem. Ne lójon egyik ember se a másikba, és nem mozoghat át ember kis időre sem falakon. Ez elengedhetetlen ahhoz, hogy a valóságot minél jobban megközelítse a megoldásom. Ezért minden pillanatban bármi is történik, például torlódás alakul ki ajtóknál, nem szabad, hogy a figyelembe vett normák és forgalmi szituációk ezeket az alapvető szabályokat megszegjék.

## 2.3 Első program korlátai

Az első programomat ennek a megoldására alkottam meg. Korlátozásokkal alkottam meg a szabályrendszerét, hogy fókuszálni tudjak a járókelők egymás kikerülésére. Az embereknek kiterjedésük volt ebben a megoldásomban már, viszont mindenki egyenlő széles volt. A helyszínt lehetett tetszőlegesen paraméterezni, statikus és mozgó falakat is le lehetett helyezni benne. Az embereknek csak egyszerű teendőjük volt, el kellett jutniuk A pontból B pontba. A helyszín leegyszerűsített volt annak érdekében, hogy négyzetrácson lehetett csak haladni. Mezőkből állt a környezet, így csak diszkrét értékekkel és állapotokkal dolgoztam.

Mivel minden ember vagy az egyik mezőn áll vagy egy másikon, így a programomban egy ütemezőt alkottam meg. Az ütemezőt arra alkalmaztam, hogy az

eseményeket pillanatokra osszam. Ennek okán a kifejlesztett algoritmusok olyan útvonalakat adnak meg, amelyek olyan kezdő és végpontok listájából álltak, amelyekhez egy időpillanat is társul lépésenként. Azt a program fejlesztése során alakítottam ki, hogy a megadott időpillanat az érkezés idejét írja le az adott mezőre, vagy az indulás idejét az adott mezőről. Ezeket mind alkalmaztam a dinamikus falakra is.

Ebben a megoldásban arra is kívántam figyelni, hogy az emberek minél okosabban járják be az útjukat, és nem csak magukat, hanem a többi közlekedésben résztvevőt is vegyék figyelembe.

Sok probléma van azzal, hogy a többi résztvevőre is figyelemmel legyenek. Ha a valóságban egy vonulás van, akkor azon keresztül jutni szinte lehetetlen. Ha az embernek nem adok meg minden tudást, még a jövőről is, akkor úgy is gondolhatja, hogy ez olyan, mint egy liftből kilépő ember csoporthoz, hogy egyszer elfogy, így érdemes várakozni. Ez azt a problémát veti fel, hogy egy-egy embernek, azaz az algoritmusnak milyen paramétereket adjak meg, milyen információkkal engedjem dolgozni. Tudhat-e mindenről, láthat-e a jövőbe, vagy csak a közvetlen környezetéből megszerezhető információval dolgozhat. Ebben a megoldásomban nem csak a közvetlen környezetéből levonható információkból engedem dolgozni az algoritmusokat, és törekszem arra, hogy minden információt megadjak az útvonalkereséshez.

Minden információ birtokában elég csak egyszer kiszámítani az útvonalat és onnantól el van rendeltetve a kimenetele az eseményeknek. Ehhez szükséges az, hogy valóban tudjon mindenről minden résztvevő, és a jövőről is. Emiatt ebbe nem fér bele váratlan helyzet, egy fal hirtelen megjelenése vagy egy kiszámított útvonalról való letérés, mert minden útvonal elrendeltetett. Az elrendeltetett útvonal előnye, hogy a lehető leghatékonyabb áthaladást tudja biztosítani, viszont a dinamikusság rovására megy.

### 2.3.1 Mezők összeszervezése

A bejárható tér négyzethálójának megvalósítását egy két dimenziós tömbbel valósítottam meg, melynek elemei a mezők. A tömbben a pozíciójuk megfelel a valóságnak, ami a mezők esetében nem teszi szükségessé, a szomszédolási viszonyok eltárolását, hiszen a négyzetrácsból következik ez, és többletinformációval sem rendelkeznének ezen adatok. Ezzel az algoritmusok kisebb memóriaigényűek lehetnek,

és nem kerül számítási időben lekérdezni a szomszédjait, hanem tudni lehet a mező adatai nélkül is őket. Itt a terület széleire kellett figyelnem hibás eredmény és a túlindexelés elkerülése érdekében.

### 2.3.2 Útvonal és mozgás megtervezése

Az embereknek aktuális pozíciójuk van, de fontosabb, hogy van útpont láncolatuk is. A járókelők létrehozásánál kiszámítom az útvonalukat, és ők a kezdeti pozícióikból kiindulva ezek láncolatán haladnak végig egyesével a megadott időpillanatokban. Ez igaz a dinamikus falakra is. Itt az embereknek egyéni sebességet nem adtam meg paraméterül, így az útvonal kereső algoritmust még optimálisabba dolgozhattam ki. Ekkor a négyzethálót a térfelület pontos lenyomataként használva felhasználható az, hogy egy élszomszédos négyzetre történő ájtutás egy embernek egy időegységbe telik, a csak egy közös csúccsal rendelkező mezőre ez az érték  $\sqrt{2}$  időegység, az átlós mozgás miatt.

Az időegység minél kisebbnek való megválasztásával a szimuláció részletezettségét és pontosságát lehet növelni. A pontossághoz szükséges lenne az, hogy egyik időpillanatban még az egyik mezőn lehessen az illető, a másikban meg már a következőn. Viszont egy irracionális számmal dolgozni ezen esetben nehéz, mert számolni kéne azzal, hogy abban az időpillanatban ott volt már harmad időegységnél, viszont tovább is állt már. Ezzel való számolást felesleges komplexitásnak vélem, mert megfelelő időegység megválasztásával ez minden kiküszöböltető. Három megoldást fontoltam meg ebben a helyzetben. Amikor 10.000 időegységbe, 5 időegységbe és 2 időegységbe kerül az élszomszédos mezőre áthaladás.

### 2.3.3 Időegység nagyságának megválasztása

A három megoldás közül a végső választást az az információ segítette elő, hogy ha egy ember a négyzethálón, csak élszomszédokra, vagy átlósan léphet, mint a sakkban a király, akkor száz időegység alatt nem egy száz időegység alatt megtehető távolságnak megfelelő sugarú, négyzetekből álló „kör” valamely mezőjére lehet eljutni. Az átlós és a szomszédos lépés időszükséglet szerinti arányának függvényében más-más alakzatot ad bejáratot a térfelületen. Ha ez az arány 1 és 2 között található, akkor egy nyolcszögöt ad eredményül. Ezen nyolcszög csúcsai a következők. A négy főégtájban száz időegységnél halad élszomszédos mezőkön egyenesen, továbbá a fő mellékégtájak irányában száz időegységnél halad átlósan szintén egyenesen, ekkor 8 mezőre jut el a

kezdeti mezőről a király, és ezek a mezők, mint csúcsai a síkidomnak, alkotják a nyolcszöget. A nyolcszög alakazzal magyarázható, hogy a nyolc irányon kívül nem lehet más pontját a körnek elérni egyenes vonalú mozgással, csak törtvonalat leíró mozgással. Ez esetben a király több utat tesz meg, mint légvonalban nézve az út végére, ezzel a kör alakja torzul és egy nyolcszöggé alakul.

Ezen okból kifolyólag tetszőleges pontossággal megközelíthettem a  $\sqrt{2}$ -t, mint az élszomszédos és sarokszomszédos mezőkre történő áthaladáshoz szükséges idő arányát, viszont nem értem volna el vele különösebb javulást, mindenkor csak nyolcszöget kapnék bejárható területnek egy adott időn belül. A részletes, például 10.000-es időfelbontásról, ahol 10.000 időegységbe kerül egy élszomszédos mezőre áthaladni, és 14.142 időegységbe kerül egy átlós mezőre történő átmozgás, nagyságrendekkel több ideig futna az algoritmus, de szemmel látható javulást nem eredményezne. Többi járókelő mozgását is figyelembe kell venni időegységenként. Ezt az útvonal számításánál kifejezetten számításigényes vizsgálni, mert minden egyes időpillanatban felszabadulhat a szomszédos mező, amire átkívánnak haladni a cél érdekében. Ezért egy kisebb, 10x10-es négyzethálón keresztbe áthaladni, még ha csak kis forgalom van 1.4 millió időpillanatot át kellene vizsgálni. Amint egy ember vagy fal elállja valamelyik időpillanatban a legrövidebb útját, akkor őt meg kell várni vagy alternatív útvonalat kellene kiszámítani. Várakozás során lehet, hogy egy következőt és egy azt követő embert is meg kell várni, de akár egy statikus falra is várhat a szimuláció végéig. Az emberek esetében a többi embertől tudná meg azt, hogy ők mikor terveznek tovább haladni. A fal esetében a fal tulajdonságát venné figyelembe. Ha a fal mozog, úgy kezeli, mint egy embert, ha statikus, akkor a szimuláció végéig gátolja az adott mezőn keresztül a mozgást. Ezt azért kívántam elvetni, mert nem tette lehetővé ennek a környezetnek a maximális kihasználtságát.

Fontos megjegyezni, hogy csak akkor érkezhet meg egy ember a mezőre, hogy ha az üres és az első lehetséges tovább haladási lehetőségeig legalább még üres is marad és út közben nem ütközik senkivel. Nem természetes az kimondani, hogy egy ember mozgása során nem ütközik senkivel. Ugyanis ezen embereknek adtam kiterjedést, köröknek vettem őket. Ekkor az egyik mezőről a másikra történő áthaladás estében meg tudom, és meg is vizsgálom azt, hogy történik-e ütközés. Erre részletesebben kitérek később. Jelenleg kisebb időegységnek kívánom választani az élszomszédos mezőre való áthaladás időszükségét. Hogy pontos legyen viszonylag a végeredmény, fontos, hogy az

átlós lépés időigényét ne kerekítsem sokat. Ez a gondolat szülte a 0.2-es időegységgel dolgozó megoldást, hogy mind az 1, mind az 1.4 egész számú többszöröse legyen. Egy ennek egy változata az az, hogy 5 időegységbe kerül az élszomszédos mezőre történő átmozgás mozgás és 7-be az átlós. Ezen kerekítés nem hagyható figyelmen kívül, de a jelen környezetben elengedhető pontatlanság a nyolcszög alakú bejárható terület miatt.

Ennek nagyságrendekkel kisebb számításigénye van, de még ezt is gyorsítani kívántam egy okos megoldással kiváltva az emberek kérdezgetését a tovább haladással kapcsolatban. Ekkor véglegesítettem, hogy a  $\sqrt{2}$  -t 1.5-nek veszem. Ez már komolyabb kerekítés, viszont ezzel a szomszédos mezőre az áthaladás csak 2 időegységbe, az átlósba meg három időegységbe kerül. Ennél egyszerűbb az 1 és 2 vagy az 1 és 1 lenne, de az még a nyolcszög alakot is tönkre tenné és négyzetté alakítaná át, amit eredetileg körnek szántam, így elvetettem ezen egyszerűbb lehetőségeket. A 10x10-es négyzethálón az átlós áthaladás időpillanat igényét 1.4 millióról radikálisan, 70-en keresztül 30-ra csökkentettem. Ez utat engedett annak a megoldásnak, hogy minden időpillanatot leképezhessen külön.

### 2.3.4 Időpillanatonként állapotképek

Azzal, hogy minden időpillanatnak külön van egy állapota, azzal lehetőségem nyílt arra, hogy megadjam azt minden egyes mezőre, hogy abban a pillanatban melyik mezőkre lehet onnan tovább haladni. Fontos, hogy az ember cselekedete nem mindig jár mozgással, célszerű lehet egy vagy több időegységnöt várakozni is. A várakozás az emberek útvonalában szintén útpont. Nem igényel sok számítást az, hogy két mozgás közötti időt vizsgáljam meg, vagy el legyen tárolva az, hogy csak áll abban az időpillanatban, így eltároltam. Az útvonal kiszámítása után nem foglalkozom tovább az adott járókelővel, így már csak a megjelenítésnél használtam fel újra az útvonalát, amelynél segítség volt a várakozást is eltárolni.

Minden egyes mezőhöz minden egyes időpillanatban hozzárendeltem egy 3x3-as táblázatot, amelyben igaz vagy hamis értékeket tárolok el. Arra használtam, hogy minden időpillanatban el legyen tárolva mezőnként, hogy melyik irányba szabad a tovább haladás, vagy még lehetséges-e az egyhelyben várakozás. Ez memóriát igényel. Egy kisebb, 10x10-es pályán egyetlen ember keresztbé történő áthaladásához 30 időpillanat kell legalább. Azaz körülbelül 30 pillanatkép szükséges minimum a játéktérről, és annak minden mezőjéről, így már 30x10x10, 3000 mezőt tárolok el

különböző időpillanatokból, továbbá mezőnként eltárolom mind a 3x3 igaz agy hamis értéket. Ez 27.000 bool (igaz vagy hamis) típusú érték, ami 27 kB (kilobájt) memóriát foglal le, és ez csak a nyers adat. Sajnos, a pálya méretével, és a szimuláció hosszával a memóriasükséglet exponenciálisan nő. A szimuláció hosszát érdemesnek tartottam a pálya méretéhez megválasztani, mert a mérete adja meg, hogy mennyi időegységbe kerül átérni átlósan minimum. Ezért a nagyobb oldalát vettetem a játéktérnek, mint téglalapnak, és azt szoroztam be 3-mal, az átlós mozgás időigényével, és ezt szoroztam meg 10-zel, hogy legyen elegendő idő a közlekedésnek kialakulni, és erősödő, illetve csillapodó forgalmat is jól kivehetően vizsgálni.

A szimulációt futtatni már egy 9x9-es négyzet alakú pályán is érdemes, mert annak mérete már elegendő ahhoz, hogy a járókelők ki tudják egymást kerülni, és már vizsgálni lehessen a járókelők mozgását. Egy 100x100-as pálya 270 MB (megabájt) nyers adatot igényel. Ezen méretet a felső határnak vettetem, így maximum 100 egység széles vagy magas pályákkal dolgoztam legfeljebb. Ezen pályákat már önmagukban nem szimuláltam, mert nagyon sok járókelőt igényelt volna ahhoz, hogy komolyabb kerülőutakat kelljen kiszámolnia az útvonalkereső algoritmusnak. Ezért a négyzetek megtörése érdekében falakat helyeztem le a játéktérbe.

### 2.3.5 Falak a játéktérben

A falak olyanok, mint az emberek, tudnak akár mozogni is, például ha egy forgóajtóról van szó. Nem ütközhetnek bele ebbe sem az emberek. Két típusa van, egy kerek, ami mellett átlósan el lehet haladni, és egy szögletes, ami mellett nem lehet ezt megvalósítani. Az áthaladás korlátozását úgy oldottam, meg, hogy a játéktér statikus falai az összes időpillanatban a szomszédos mezőkön letiltja a fal irányába történő haladást és a fal mezőjéről történő bármilyen haladást is. Ezzel az útvonal kereső algoritmus nem kíván a falakba belevezetni embereket, mert az adott mezőről tilos arra tovább haladni minden időpillanatban. A nem lekerekített oszlopok több tiltást jelentenek, ugyanis az élszomszédos mezők feléjük irányuló két átlós mozgását is külön letiltják minden időpillanatra. Ezt lehet úgy tekinteni, mintha egy lenyomatot hagyna a fal a játéktérnek a pillanatképein. A dinamikus lekerekített falaknak és az embereknek a lenyomatai viszont nem triviálisak. Ezek a tiltó lenyomatok összefüggenek az emberek szélességével. Ha túl nagyok lennének, akkor nem lehetne átlósan, ütközés nélkül

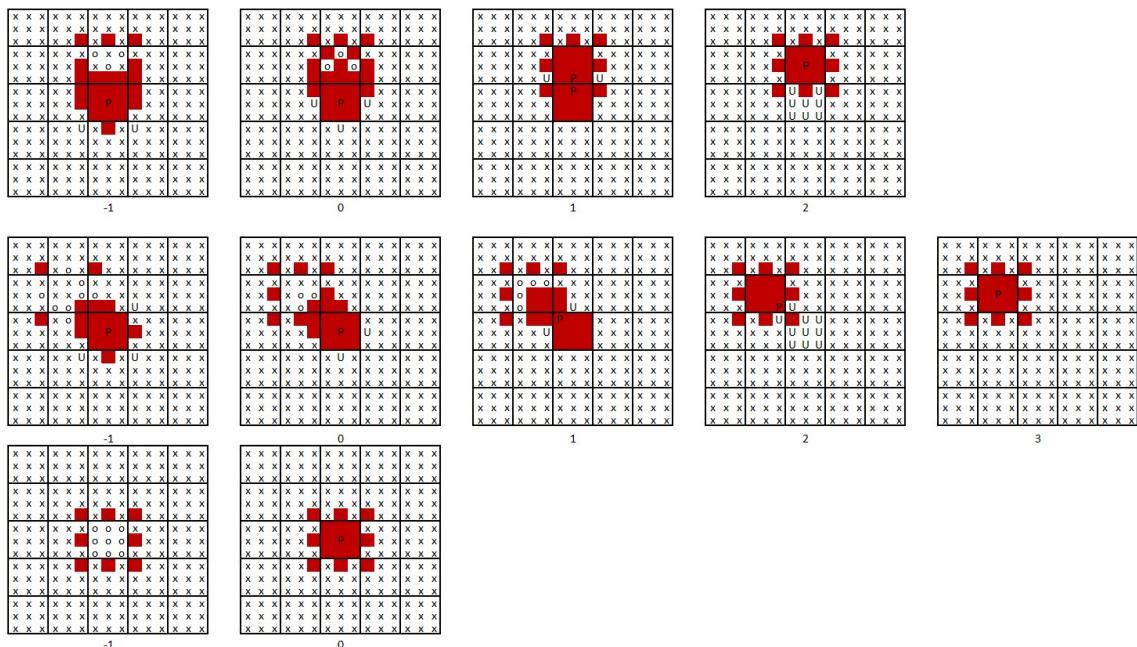
egymás mellett elhaladni. Akkorára választottam a szélességüket, hogy az átlós elhaladás lehetséges legyen, sőt még annál valamivel kisebbre.

### 2.3.6 Ütközések elkerülésének megvalósítása

Az időpillanatokra osztott megvalósításban mindenképp el akartam érni, hogy folyamatos is lehessen a mozgásuk, azaz a mozgásuk ne egy pillanat alatt történjen meg, hanem a megjelenítésnél ezt folyamatosnak is lehessen ábrázolni. Ekkor az átmozgások során ütközések is megtörténnének. Ezen ütközéseket kívánom feloldani.

Azért terveztem kisebbre az emberek szélességét, hogy ne csak az átlós haladást könnyítse meg, hanem a mások helyére érkezését is. Ne kelljen megvárni azt, hogy a másik ember a mezőről teljesen ellépjön, hanem elegendő az is, hogy már félig kint van, mert akkor szomszédos mezőről megkezdhetem az áthaladást. Ha ugyan abba az irányba tervez haladni egy ember, mint a másik ember, és a szomszédos mezőről lép a helyére, akkor még egy időpillanattal korábban is megkezdheti az áthaladást, ha az emberek szélességét fél egységnek választom. Ezért fontos az ember szélességének meghatározása, mert ezek alapján kell elkészítenem a mozgásuknak a tiltólenyomatát.

A 9. ábrán az átlós mozgást, a vízszintes vagy függőleges áthaladást és a várakozást mutatom be, hogy ezeknek milyen tiltólenyomatuk van a mozgás megkezdéséhez képest egyes időpillanatokban.



9. ábra A tiltólenyomatok adott időpillanatban adott irányú mozgás esetén

Az ábrán az látható, hogy a különböző mozgások során a mozgás megkezdésének időpillanatát megelőzően és követően mely mezők mely irányait tiltja le a mozgás tiltó lenyomata. A végső programban csak a mozgási folyamatok lenyomatai közül csak a mozgás megkezdése előtti lenyomattól vettem csak használatba a lenyomatokat. Itt a lenyomatok tervezése során figyelnem kellett arra, hogy letiltsam azokat a mozgásokat is, amelyek ugyan azon célmezőre történtek volna. Vagy csak olyan szögből közelítsék meg, ahonnan legalább egy időpillanatuk még van tovább állni. Ezen lenyomatok kialakítása után a kiszámolt útvonalak már nem metszették egymást.

### 2.3.7 Útvonalkereső algoritmus megalkotás

Az útvonalkereső algoritmus, amit itt használtam, az az A\* tovább gondolt változata. Az A\* egy gráffal dolgozik, viszont itt időpillanatonként van egy-egy külön gráf, azaz akár több száz gráffal kell dolgoznia egy útvonal kiszámítása során. A szomszédságok nem mindenkor ugyan azok a különböző időpillanatból leképzett gráfban és az adott élek súlyozottak. Megoldandó probléma volt megoldani az A\* algoritmushoz, hogy ha a játékteret gráfnak veszem, akkor van hurokél is benne, lehet egy helyben várakozni, továbbá lehet, hogy nem az egyhelyben állás segít a tovább haladáson, hanem egy oda és egy visszalépés az embertömegben, hogy valakit elengedjen például. Ezért nem old meg minden problémát csak a célmező légvonalbeli távolságával számolni, többek között azért, mert egy csúcsát a gráfnak lehet többször is be kell járni ahhoz, hogy időben a legrövidebb utat adja meg. Ettől a heurisztikát teljesen át kellett gondolnom és az algoritmus leállításának okát szintén.

Az A\* algoritmus akkor áll le megfelelő heurisztika esetén, amikor kifejti vagy kifejtené a célmező csúcsát. Viszont erre itt nagyon sokára kerülhet sor, akár sohasem, ha nincs lehetőség A csúcsból B csúcsba eljutni. Ennek meghatározása nem triviális, mert időben változik a játéktér. Például az utolsó pillanatig próbálkozni fog azzal, hogy hátha elsétál az embertömeg előtte, vagy a fal megszűnik előtte, ami nem statikus, hanem dinamikus, és odébb mozdul. Erre csak nagyon bonyolult megoldást sikerült kitalálnom, amely nem is oldotta meg minden esetben a problémát. Arra jutottam, hogy vizsgálja meg az algoritmus, hogy van-e egy olyan falnak tekinthető része a térnek, amin keresztül a szimuláció végéig nem lehet átjutni. Ennek felismerésére, kezelésére és

eltárolására jó megoldást nem találtam, így próbáltam máshonnan megközelíteni a problémát.

A heurisztikában próbáltam a megoldást meglelni. Az emberek vagy az idő, vagy a távolság vagy a költség szerinti legkedvezőbb útvonalat keresik. Ahogyan a leggyorsabban átjutnak a csomópont túlsó pontjába, vagy fáradtság okán inkább a rövidebb útvonalat preferálják, ahol lehet, hogy a tömegben sokat kell várakozni, de az út maga rövid, vagy a költségét próbálják minimalizálni. Az utolsóra inkább nagyobb környezetekben kerülhet sor, mikor tömegközlekedési eszközöket vesz igénybe a járókelő, de jellemzőbb a gépjárművel közlekedőkre ezen szempont. Sajnos a leggyorsabb út nem volt kedvező megoldás megválasztani a heurisztika alapjának, mert akkor a kiindulási mezőhöz legközelebbi mezőket fejti ki először, hiszen azokra tud eljutni legkorábban, amivel egy szélességi keresésre hasonlító algoritmust kaptam volna végül. Amikor a megtett út és a hátramaradó út becsült hosszával dolgoztam, az a helyben állást segítette elő, mint hogy a járókelők kikerülését. Erre azt a megoldást adtam, hogy a helyben állást is távolságnak számítom. Ezzel valamivel közelebb került az algoritmus az időt alapul vevő megoldáshoz, viszont mégsem ragadt le a kezdő mező környékén. Ezen megoldás az eltelt idő és a megtett, illetve a becsült hátramaradt távolság összegével dolgozott, aminek eredménye működőképes szimuláció lett.

### 2.3.8 Megmaradt problémák orvoslása

Ez a megoldás nem oldotta meg az el nem érhető mezők problémáját, arra egy maximális időlimitet és távolságlimitet adtam meg az algoritmusnak. A valóságban is az emberek gyakran meggondolják magukat, ha ésszerűtlen útvonalat kéne bejárniuk, vagy forgalom nélkül az idejének többszörösét venné igénybe az, hogy megtegye ugyan azt az utat. Ezért az algoritmusnak megadtam a pályamezők számának kétszeresét távolságlimitnek, és a háromszorosát időlimitnek. Így teret engedtem komplex útvonalaknak is, de a feleslegesen bonyolultakat kiszűrtem.

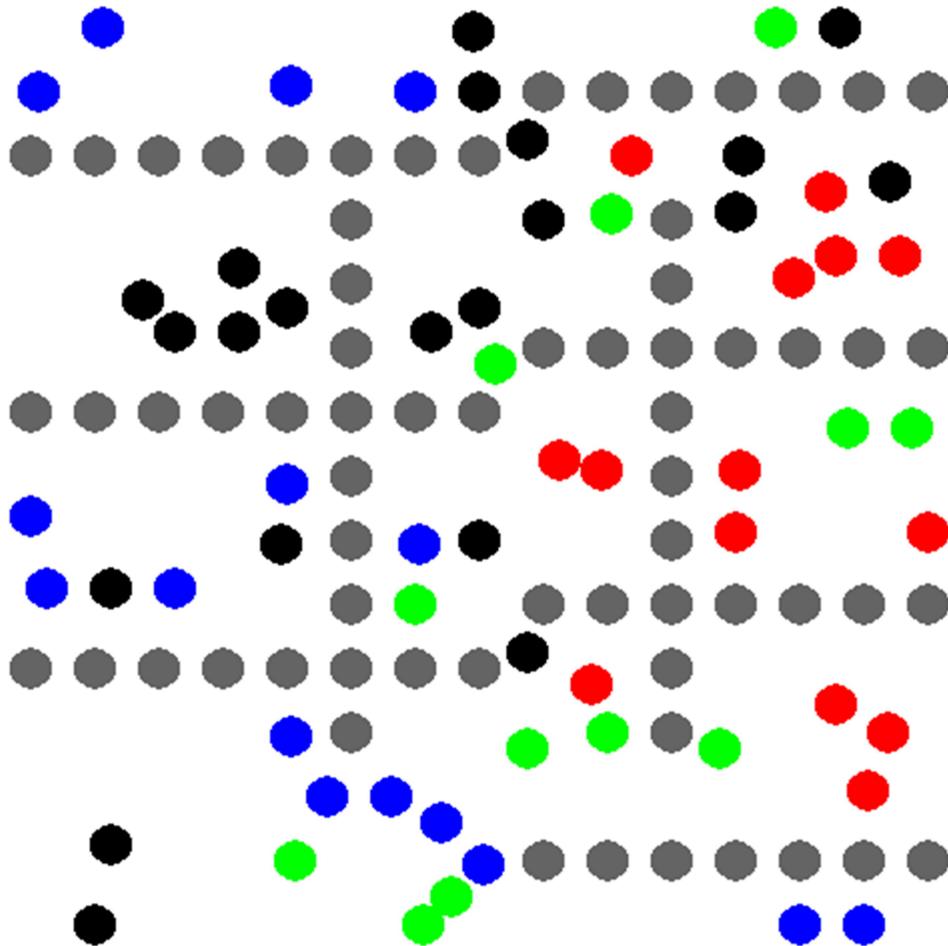
### 2.3.9 Első program eredménye

A kezdeti kikötések minden abban segítettek, hogy koncentráltan az ütközések elkerülésével tudják foglalkozni. Erre az első program teljesen elégséges volt, viszont a mezőkre bontott térrel nem lehet valóságos szimulációt megoldani. Ezért a diszkrét értékeket el kell engednem a következő programban. Az útvonalkeresés eleve elrendeltetett a járókelő létrehozása során. Ez időpillanatokra osztott környezetben

megvalósítható kis állapottéren, viszont a mezők megszüntetésével ez az állapottér nagyságrendekkel több állapottal fog rendelkezni, amit memóriában eltárolni már nem lehetséges. A megalkotott algoritmus a mezőket használta fel gráfnak, így átemelni nehéz egy az egyben, viszont fontos tanulság volt számomra, hogy az ütközések elkerülésének megoldását komplex probléma globálisan előre kiszámolni, ezért a következő programban időpillanatról időpillanatra, lokálisan tervezem megoldani a problémakört.

### 2.3.10 Megalkotott program grafikus felülete

A megalkotott program grafikus felületét is véglegesítettem. Az emberek színes körök, akik előre, a színüknek meghatározott sarokba haladnak. A szürke körök a falak.



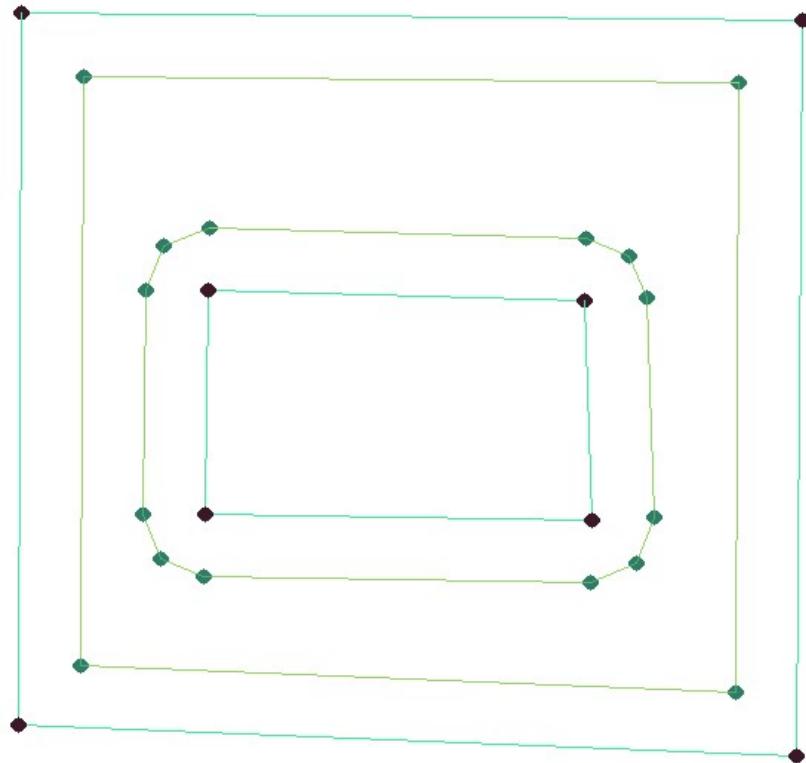
10. ábra Pillanatkép az első programról: kékek a jobb alsó sarokba haladnak, a pirosak a bal felsőbe, a feketék bal alsó sarokba és a zöldek jobb felsőbe

### **3 Saját munka bemutatása: Második program**

Ezt a programomat négy jól elkülöníthető részre lehet osztani. A navigációs háló csúcsainak, azaz a teret határoló síkidomok létrehozásáért felelős rész, az első része a programnak, amely az alaprajzot figyelembe véve képes megalkotni ezeket a határoló síkidomokat. A második része az, ami a határoló síkidomok által alkotott teret háromszögekre bontja. A harmadik része az, ami keretet ad ennek a háromszöghálónak, helyiségeket köt össze és tárol el róluk különböző adatokat. Továbbá van a fő feladatot megvalósító negyedik része a programnak, amely a járókelők eltárolásáért, és a mozgatásukért felelős. Az utolsó kettő szorosabban összetartozik, és komplexitásában megegyezik az első két résszel.

### 3.1 Navigációs hálót határoló síkidomok létrehozása

A határoló síkidomok azon sokszögek lesznek a helyszín síkjára vetítve, amelyeken a járókelők középpontja tartózkodhat. Emiatt a járókelők szélességével ez szorosan összefügg, ugyanis az alaprajz csak a falakat tartalmazza, a járókelők szélességével nem számol. Ezért a létrehozott határoló síkidomok a különböző szélességű járókelőkre minden mások. A létrehozott síkidomokat a falakat leíró síkidomuktól pontosan egy megadott távolságra lévő pontok halmaza képzi. Egy kör alakú fal - például egy oszlop - esetében a létrehozott síkidom egy nagyobb sugarú kör lenne. Egy téglalap alakú fal esetében egy téglalapot ad vissza, viszont szélesebbet, és hosszabbat, de minden toldását a sarkoknál lekerekít (11. ábra). Ennek két megoldását alkottam meg. Először én adtam meg ezt a létrehozott csúcshálót, és ahhoz rendeltem a falakat, ami remekül működött, de nem ez az elvárt irányba a folyamatnak, és kézzel kellett volna megalkotnom különböző szélességű járókelőkre minden egyes ilyen csúcspont halmazt. Második megoldásom során ezt automatizáltam.

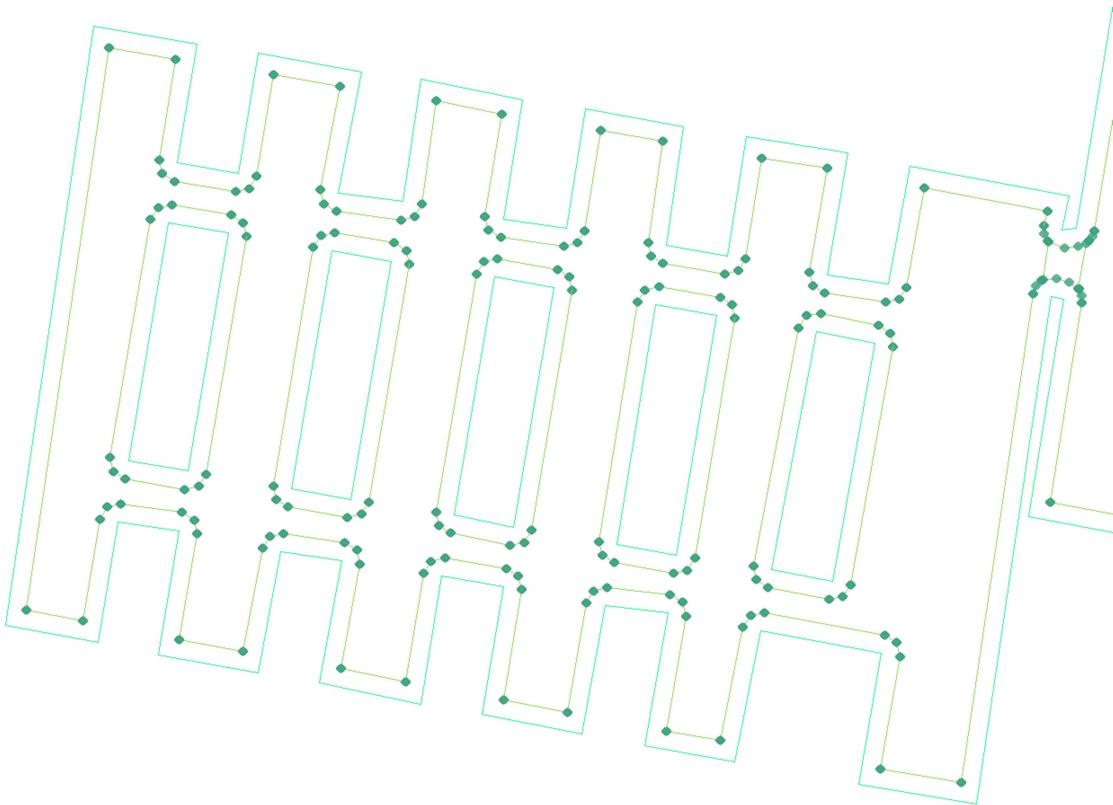


11. ábra Kívülről és belülről bejárható téglalap alakú falak határoló síkidomai

Téglalap esetén a határoló síkidom alakja lekerekített, ha a síkidom belseje nem bejárható, míg ha a szoba határát jelzi, azaz belülről járható be a síkidom, akkor téglalap lesz a határoló síkidom

### 3.1.1 A navigációs háló háromszögeinek létrehozása

A programrész paraméterül kap egy alaprajzot és egy hosszt, ami a járókelő szélességét írja le. Ennek alapján legenerálok különböző határoló síkidomokat, és ezeket adom tovább a program második részének, ahol ezekkel a paraméterekkel a síkidomok közötti részt háromszögekre feldarabolja és hézagmentesen és átlapolódás mentesen lefedi.



12. ábra Az IB413-as terem határoló síkidomai és falai

A síkidomok oldalait követve alkotja meg az itt alkalmazott algoritmus a navigációs háló szélét alkotó csúcsokat. Két szabályt alkalmaztam. Az új pontok a következő vizsgált két oldal szögfelezőjén helyezkedjen el és az oldalaktól a megadott távolságra. Ez egy geometriai probléma, amire képletet hoztam létre.

$$\sqrt{-\left(\left(\frac{b^2 + c^2 - a^2}{2c}\right)^2 - b^2\right)} = x$$

Itt a szakasz két végpontjától mért távolsága a pontnak „a” és „b”. A szakasz hossza „c” és a pont távolsága a szakaszra illeszkedő egyenestől a kapott „x” érték.

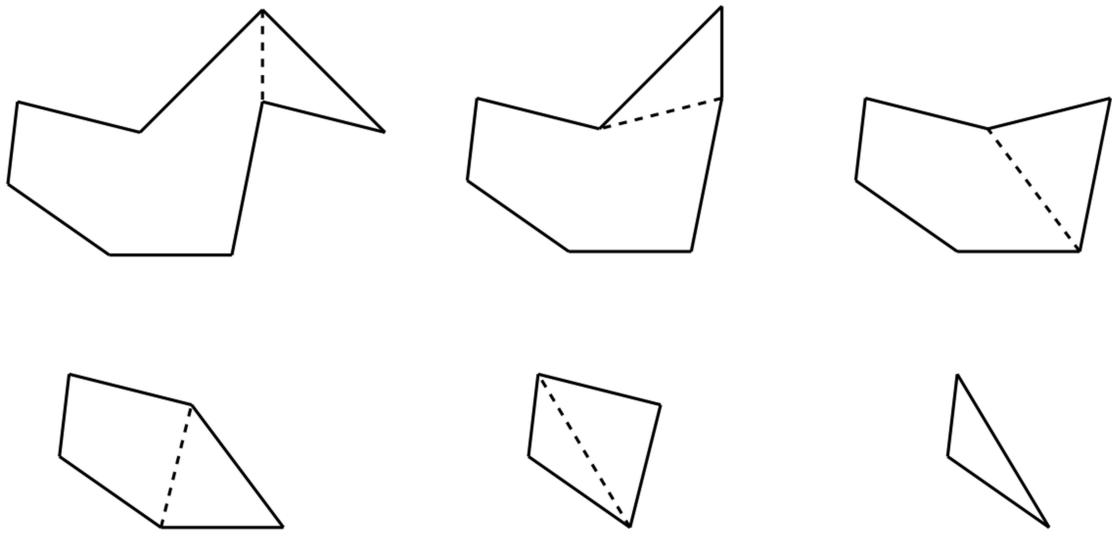
A másik szabály az az, hogy kiszögellésekknél a síkidomnak körívet kellene tartalmaznia. Ezt én három csúccsal cseréltem fel, a csúcsemberek minimalizálása érdekében. A három csúcs közül a középső az előző szabály alkotta csúcs, a másik kettő meg a két oldal közös csúcsától az oldalakra merőlegesen adott távolságra elhelyezkedő pontok adják. (11. ábra belső téglalap határoló síkidomának sarkai)

Az említett két szabály alapján egy emelet alaprajzát fel tudja dolgozni helyesen a program. Nagyon széles emberek esetében lehetne ezzel csak probléma. Ugyanis akkor fel kéne készítenem olyan esetekre is a programot, amikor egy szoba elérhetetlen az illető számára, mert az ajtón nem fér be. Ekkor a létrehozott síkidomok metszenék egymást, és megfelelő vágást kellene eszközölni közöttük. Erre nem tértem ki a munkám során, mert a 30 és 50 cm széles emberek esetében erre nem volt szükségem. Egy emelet alaprajzát emberekre tervezik, ezért nem merülnek fel szélsőséges esetek. Egyetlen esetre készítettem fel a programom ezen részét csak, amikor ajtókeretekről van szó, amelyek csak keskenyen szögellenek ki a falból. Ekkor a kiszögellésekknél lévő első és utolsó csúcs lehet, hogy közelebb kerül a falhoz, mint az ember szélessége. Ezt a problémát ennek külön ellenőrzésével oldottam meg, azaz nem csak a két vizsgált oldal távolságát vettem alapul, hanem az azt megelőzőt is és az azt követőt is. Ezzel a vizsgálattal megoldottam minden felmerült problémát az egyetem I épület 4. emeletének lemodellezésénél.

### **3.1.2 A bejárható tér háromszögekre bontása**

Az irodalomkutatás leírása során kifejtettem, hogy miként lehet ezt a problémát megoldani könnyen. Azt abban a részben nem fejtettem ki, hogy ez nem annyira könnyű, mint a leírtak. A „fülező” módszer – (13. ábra) amit leírtam, de itt röviden újra leírom - csak leírva egyszerű elgondolás. A módszer során két szomszédos oldalt kívánok minden kiválasztani a határoló síkidomokból, és megnézem, hogy a nem közös csúcsukat összekötő szakasszal alkotott háromszög jó-e a tér lefedésére. Ha igen, akkor az új szakaszt hozzáadja a határoló síkidomokhoz a két kiválasztott szakasz helyére illesztve. Így minden behúzott szakasszal kettőt kitöröl, azaz lépésenként egyre kevesebb él lesz, így véges számú él esetén véges futás idejű az algoritmus. Ez az algoritmus akkor nem jó, ha a határoló síkidomokból több is van, ekkor ugyanis össze kell öket olvasztani egyére. Ha nem tenné meg az algoritmus az elején ezt a lépést, akkor nem találna akár egyetlen egy megfelelő behúzható szakaszt sem. Példa erre egy

négyzet alakú ajtó nélküli szoba, amelyből egy téglalap ki van vágva. (11. ábra) Ekkor semelyik két szomszédos élből nem alkotható olyan háromszög, ami azt a teret fedi le, és csak is azt, amit le kívánok fedni.



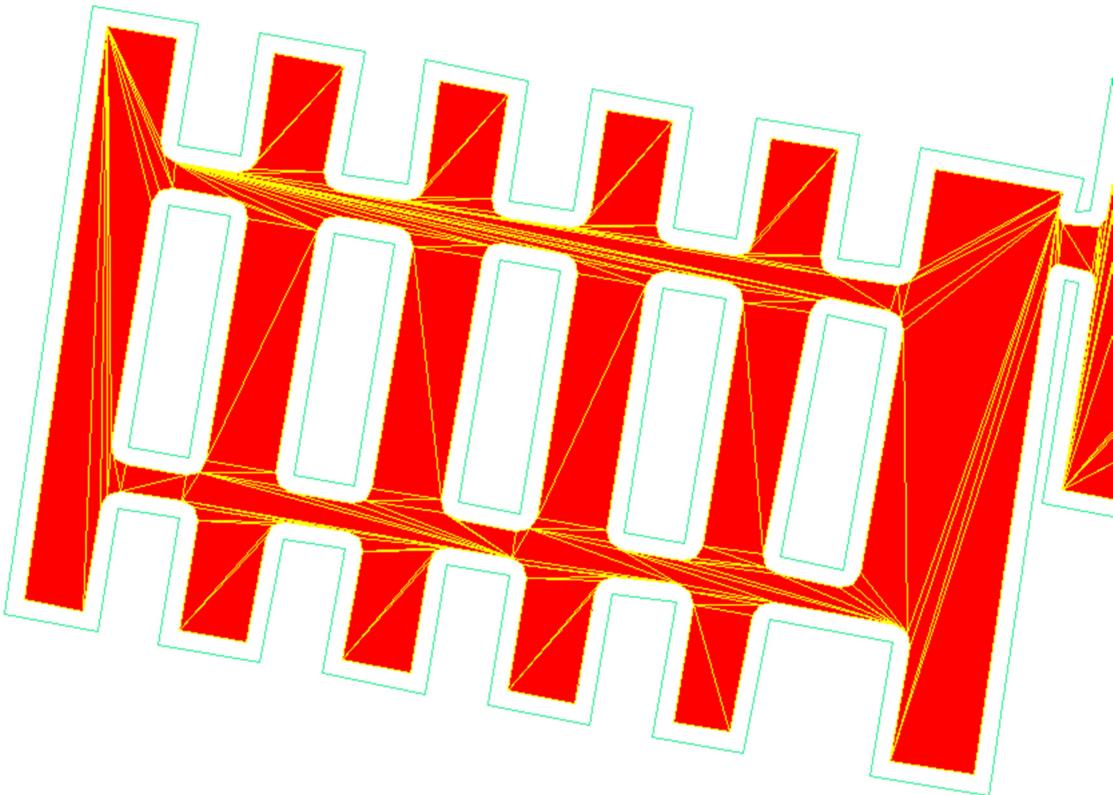
**13. ábra A „fülező” módszer: egy-egy megfelelő háromszöget leválasztva a síkidomból néhány lépés alatt a síkidom megmaradó része egy háromszög lesz, a leválasztott háromszögek a megmaradt háromszöggel együtt a síkidom egy lehetséges felháromszögelésének háromszögei**

Ha a határoló síkidomokat nem dolgozza egybe az algoritmus, akkor ellenőrizni kell, hogy az új háromszög minden pontja a síkidomon belül van-e. Kiszögellés esetén nem a belső, hanem a külső teret fedné le. Egyéb esetben felmerülhet az is, hogy a szakasz elmetszi a síkidomnak egy másik szakaszát. Ez azt jelenti, hogy átlépi a falat valamilyen módon, ami szintén nem helyes. Ezen kívül viszont nem találtam problémát a megoldással.

Másik megoldást alkottam meg. Ebben a megoldásban először a határoló síkidomok oldalain iteráltam végig. Mindegyik oldalhoz kerestem valamelyik másik síkidom egy olyan csúcsát, amellyel alkalmas háromszöget képes alkotni. Ekkor ezen háromszög azon két szakaszát eltároltam, amelyik nem a síkidom oldalát képezi. Ezt követően csak nagyon ritka esetben fordult elő olyan, hogy nem minden oldal esetében hozott létre egy háromszöget. Ezt követően azon háromszögeket hozza létre, amit csak egy síkidom csúcsaival lehet létrehozni. Ez a konkáv síkidomokra jellemző.

Ezt követően azon hátramaradó háromszögeket hozom létre, amelyek egyike sem szomszédos a másikkal a határoló síkidomok között. Itt az volt segítségemre, hogy a létrehozott háromszögek egyes oldalai mindenkor kétszer szerepelnek az adatok között.

Mert minden háromszög oldala egy másik háromszögével közös, vagy egy határoló síkidom oldala. Mind a kettő esetben kétszer kell szerepelnie az összes szakasznak. Amely nem szerepel kétszer, az biztos, hogy lefedetlen területeket határol.



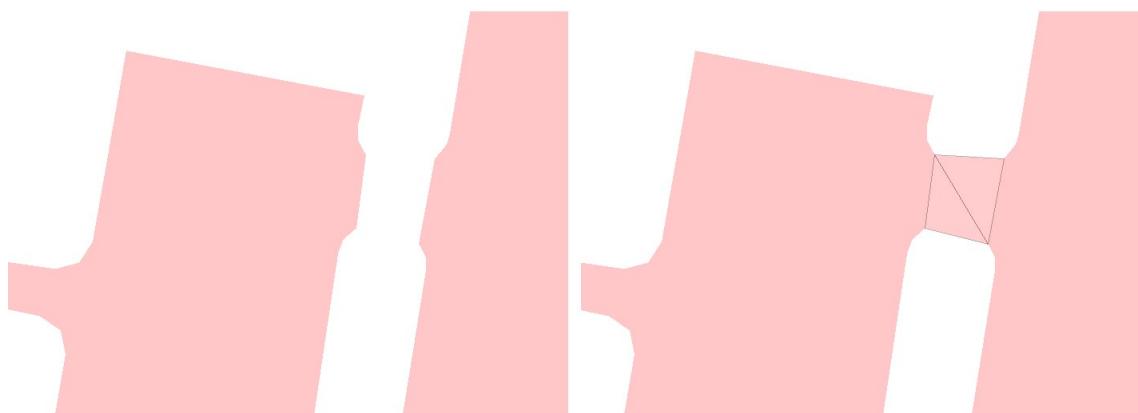
14. ábra Az IB413-as terem navigációs hálója

A számítások valamennyire pontatlannak voltak, olyan  $10^{-4}$  nagyságrendű számítási pontatlansággal kellett néha dolgoznom. Ezt epsilonnak megválasztottam, és ez volt az a maximális eltérés, amikor még két értéket egyenlőnek véltem. A pontatlanság komoly problémát okozott, főképp közel, és közel párhuzamos szakaszok esetében. Ekkor ugyanis tévesen metszőnek vélte a szakaszokat. Sajnos ezt kiküszöbölni nem tudtam, ezért még egy utolsó háromszögelést végzek. A megmaradó nem lefedett területeken már nem háromszögeket hozok létre, hanem csak néhány átlót adok eredményül, amelyek nem metszik egymást, és több átlót nem is lehet behúzni ezeken kívül. Ekkor minden a síkot – egy módon – lefedő háromszög szakaszát ismerem. Ezen szakaszok halmaza elégsges ahhoz, hogy létrehozzam a háromszögeket, vagy csak az útvonalkereséshez szükséges navigációs hálót. Külön el kell tárolni a határoló síkidomok oldalait is, ami a kiindulási adata volt ennek a résznek, így azzal nincs külön probléma.

A megoldás hatékonysága abban rejlik, hogy szobánként végzem a háromszögelést. Ezzel a teljes emelet hálóját fel tudtam darabolni kisebb részekre. Kisebb részekre darabolva egyszerre kevesebb csúcs között kellett háromszögeket alkotnom, ami a bemeneti paramétere lényegében az algoritmusnak és az időkomplexitás függvényének paramétere is így. Ezért ezt csökkentve az algoritmus futásidejét csökkentettem, ezzel nem az algoritmus, hanem az algoritmus használatának hatékonyságát növelte meg.

### 3.1.3 A szobák emeletté alakítása

Fontos, hogy az útvonalkeresés során ne a teljes emeletet adjam meg a bejárható térnek az algoritmusnak, hanem lehetőleg szobákra bontva legyen ez megoldva. A szobák kapcsolati rendszerén keresztül nem kell foglalkozni más helyiségek háromszöghálójával. Egy szobának van egy alaprajza, egy azonosítója, az alaprajzához hozzárendelt néhány kijárata, és hogy azok melyik másik helyiségekbe vezetnek. Ezen kívül a szobának van egy kapacitása, amely nem egy szám, hanem a szobában a diákoknak és az oktatóknak van külön helyük, ami a kapacitását adja a szobának. Továbbá a helyiségnek van egy tulajdonsága, ami azt mondja meg, hogy átjárásra használható-e. A szobákban a helyeknek van egy pozíciójuk és egy igaz/hamis értékük, hogy foglalt-e.



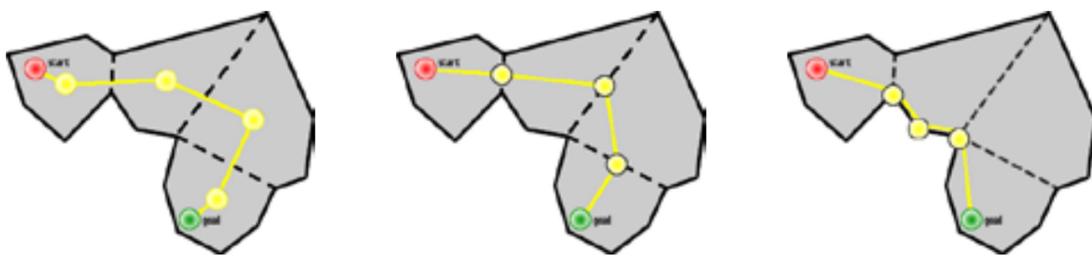
15. ábra Szobák összekötése két háromszöggel

Ezt követően a szobákat össze kell kötnie a program ezen részének. A szobák kijáratait egymáshoz rendeli az emelet. Az emelet a szobákat a helyükre illeszti, ehhez megfelelően elforgatja és eltolja őket, majd a szobák közti kijáratokat összeköti. Ezek az összeköttetések falak, ajtófélák például. Az összeköttetések közötti területet két

háromszöggel lefedő, mint bejárható tér. (15. ábra) Ezen háromszögeket hozzáadja a navigációs hálóhoz.

### 3.1.4 Az útvonal tervezése

Az útvonal tervezéséhez a navigációs háló háromszögeit az A\* algoritmusnak feldolgozható módon kell átadni. Ennek megoldására három lehetőséget találtam. (15. ábra)

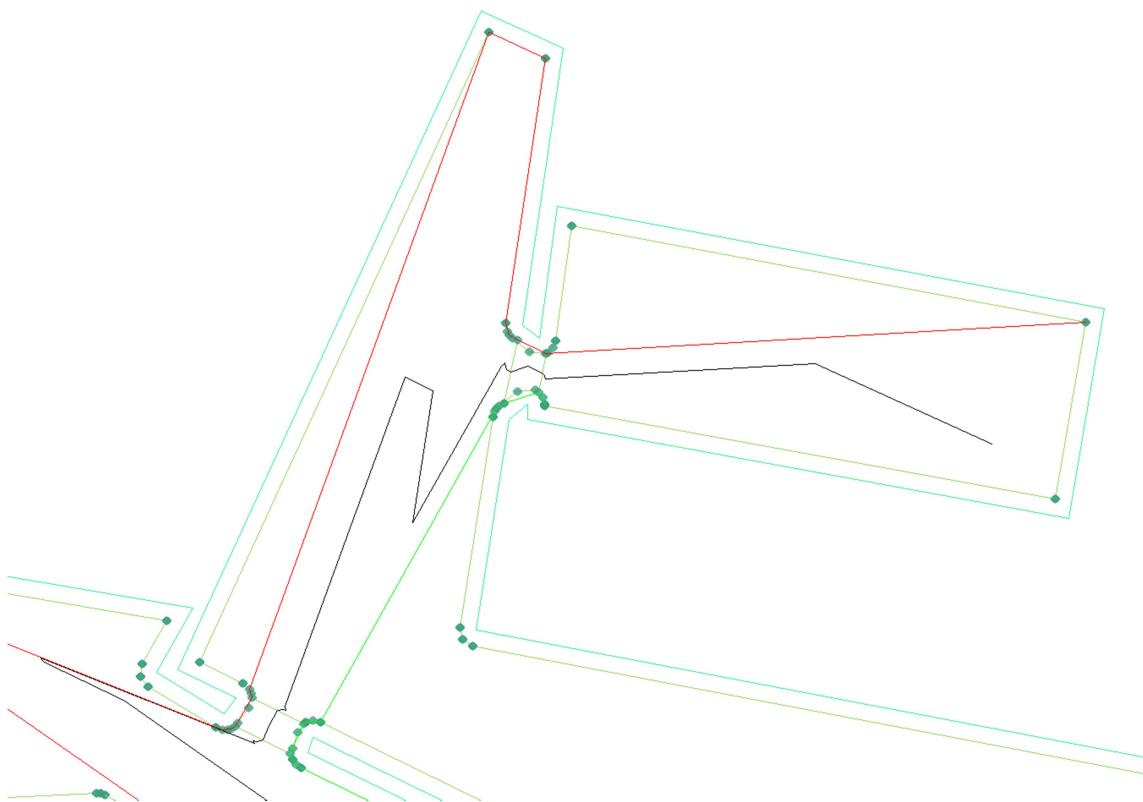


**16. ábra A navigációs háló három lehetséges gráf leképezése, ami feldolgozható az A\* algoritmussal: a háromszögek súlypontjaiból képzett gráf, a háromszögek oldalfelező pontjaiból képzett gráf, és a háromszögek csúcsaiból képzett gráf ( a képen háromszögek helyett konvex sokszögek vannak, háromszögekre hasonlóan teljesülnek a képen látottak)**

Első megoldás során a háromszögek súlypontjait összekötő gráfot adom meg az A\* algoritmusnak. Ezt a megoldást azért nem választottam, mert a gráf élei kilóghatnak a síkidomon kívülre. Ezzel a gráf a valós távolságokat torzíthatja. Az ábrán a harmadik megoldást, a csúcspontok mentén haladást azért nem választottam, mert túl szögletes utat ad, amivel dolgozni kellene, ami a csúcsszámot is megnövelheti. Ezért választottam a második megoldást, mert itt a háromszögek oldalainak felezőpontjai nem lógnak kis a síkidomon kívülre és több esetben egyenesebb útvonalat ad vissza kezdetben, mint a harmadik megoldás, így a kapott útvonal hossza nem nagyon torzul a valóságtól. Ezzel a megoldással a legbiztosabb, hogy az algoritmus futtatását követően a legrövidebb útvonalat találja meg.

Ekkor az útvonalat ki kell egyenesíteni, hogy ne egy tört vonalon történjen a járókelő mozgása, miközben végezhetné akár egyenesen is. Ennek megoldására a „tolcsérező” (angol névén funneling) algoritmust használtam. Az adott háromszög láncot végigkövetve az algoritmus visszaadja a határoló síkidomokra simuló útvonalat. Ennek feltétele, hogy az útvonal szomszédos háromszögekből álljon, amit az előző megoldás biztosít.

Az algoritmus azért „tölcsérező”, mert a háromszögek láncolatán lépelve egy tölcsér alakot képez a futása közben. Első lépésben a kiindulási pont és a kiindulási háromszög azon szakaszának két végpontja által képzett két szakasz alkotja az első tölcsért, amelyeknek oldalfelező pontja az útvonal részét képzi. Ezt követően a háromszögek láncolatán végig halad. A háromszögekbe belép, majd a másik két oldalfelező pontján keresztül kilép az algoritmus. minden háromszögből történő kilépés során csak az egyik csúcs képzi a kiléző oldal szakaszának egyik végpontját, az a csúcs, amelyik közös a háromszög belépő és a kiléző oldalának. Ekkor a másik csúcsot a kiléző szakasz másik végpontjának választja, ha ez nincs takarásban az utolsó közös ponthoz képest, akkor a tölcsér csúcsát módosítjuk csak. Ellenkező esetben a takarás okát - egy kiszögellés csúcsát – eltárolja, és innentől ettől a ponthoz képest vizsgálja a takarást. Ezek az eltárolt csúcsok adják majd az útvonalat.



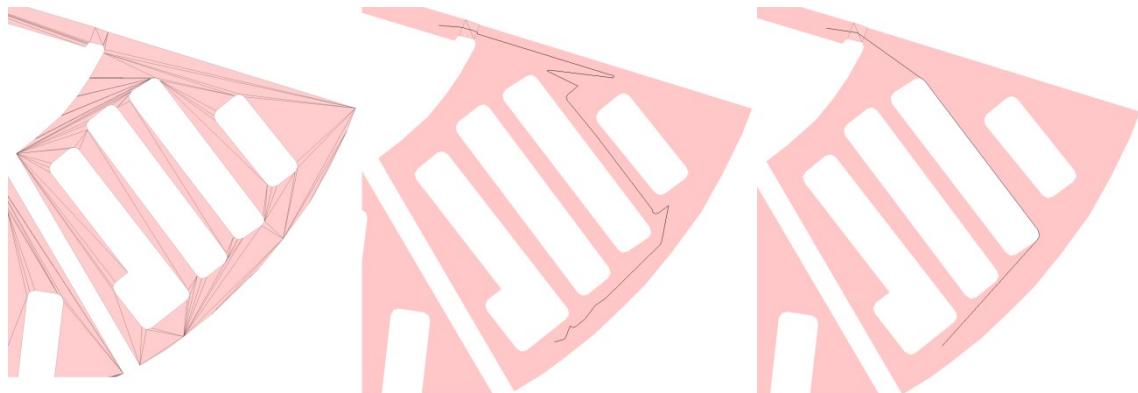
**17. ábra „tölcsérező” algoritmus működése: az útvonalat két oldalról közrefogja, és háromszögről háromszögre haladva kiegynenesíti az útvonalat**

Az algoritmus addig fut, amíg az összes háromszög kiléző oldalát meg nem vizsgálja. Ekkor a tölcsér két oldalának elmentett csúcsait megvizsgálja, és a kezdő pontot, a két oldal elmentett közös csúcsait és a végpontot összefűzi, és kész van a kiegynenesített útvonal. Az algoritmus csak az érintett háromszögeken iterál végig és

csak egyszer, így ez a része a lehető leghatékonyabb. Az iterálás közben vizsgálni kell a kiszögelléseket, ez sem komplex feladat. A tölcsér oldalainak elmentett csúcsaiból könnyen eldönthető, hogy a tölcsér jobb oldala mikor cserélne helyet a bal oldalával vagy fordítva, vagy ha az oldal nem szűkíti a tölcsért, akkor takarásban van a vizsgált csúcs és ekkor eltárolja az új közös csúcsot, ami majd az útvonalat képzi. Ezen második rész a tölcsér megfelelő oldalának és az új csúcsba vezető él közbezárt szögének vizsgálata nagyvonalakban, így az algoritmus egészre kimondható, hogy hatékony.

### 3.1.5 Szobákban generált útvonalak összekötése

A szobánként kiszámított útvonalat össze kell fűzni. Ekkor a szobahatárokon törés lehetséges az útvonalban. Ezt a törést az útvonal összeillesztésekor megvizsgálja a program, hogy lehet-e egyenesíteni, és lehetőség szerint egyenesít is rajta, ha kell. Ez költséges, de csak a törésnél kell megvizsgálni és csak egy szakasznál, ezért elhanyagolható számítási időt igényel. Programomban ezt nem valósítottam meg.



18. ábra Útvonal kiszámításának folyamata: a program a tér navigációs hálóját elkészítette, a háló háromszögeinek oldalfelező pontjaiból képzett gráfon kiszámítja az A\* algoritmus a legrövidebbnek becsült utat, végül a „tolcserező” algoritmus kiegynесíti az útvonalat

## 4 Járókelők mozgásának szimulációja

A bejárható tér létrehozása kész. A szobák emeletté szervezése az ajtóik mentén megoldott. Az útvonal kiszámítása hatékony. A járókelő mozgásának szimulációjához minden adott, csak a mozgás megvalósítása szükséges. Szükséges kitérnem a járókelők paramétereinek megválasztására, mozgásuk megvalósítására és a mozgás megvalósításának sikerességére.

### 4.1.1 Járókelők sebességének megválasztása

A sebességük a kutatómunka eredményének megfelelően maximálisan 2 m/sec, és legalább 1.6 m/sec. Ez az a sebesség, amivel egy folyosón halad, ha nincs semmi gátló tényező. Ha sarkoknál lelassít vagy más emberek a mozgását zavarják, akkor a járókelő sebessége csökken.

A sebességét a környező emberek mennyisége alapján módosítom az egyénnek. A tömeg közepén lévő járókelő sűrű közegben mozog, így lassabbnak veszem a sebességet, míg a tömeg elején haladók mozgását kevésbé lassítom, ahogyan a kutatási eredmények mutatják. Fontos, hogy a haladási iránynak megfelelően vizsgáljam az emberek sűrűségét, mert hátrafele nem számít annyit az embertőmeg a sebességen, mint előre fele. Míg előrefele 100%-ban számít, addig a járókelő háta mögött lévő járókelők csak 15%-ban számítanak a tömeg sűrűségének kiszámításában egyes embereknél.

Ezen kívül a mozgás során az éles kanyarokban, például sarkoknál lelassít az ember. A legnagyobb törés 90 foknál nem lehet nagyobb. Ebben az esetben egy teljes falat megkerül a járókelő és ekkor a fal végét három csúcs alakítja ki, amely két kisebb és egy nagyobb törést eredményez. Ekkor a törés akár 90 fokos is lehet (16. ábra), ezért ekkora törésre készítem fel a mozgás esetén a sarkoknál történő lassítást. A lassítás nem lineáris a törés szögével, hanem szögfüggvény adja a tényleges lassítás értékét.



**19. ábra Határoló síkidom akár 90 fokban is megtörhet: az elvékonyodó fal megkerülése közel 180 fokos fordulatot igényel három részre osztva, ezek közül a középső törés elérheti a 90 fokot is**

A törés szögének függvényében a sebességét lelassítom, de a sebessége emiatt nem csökkenhet maximális sebesség 66%-a alá. A haladási iránya az útvonalának következő útpontja, emiatt, ha elkanyarodik az útvonala, akkor a haladási irány is megváltozik, így ha a sebessége nem csökken, akkor kisodródik a kanyarokban. A kisodródás elkerülése érdekében lassítom le a járókelő sebességét. Azért nem alacsonyabbra, hogy valósághűen a sarkot megközelítse egy biztos sebességgel. A fordulás során az ember más irányba kezd haladni, így gyorsulni fog a kívánt irányba, ami dominálja majd mozgását és a sodródás ellen dolgozik.

Ezt a lassítást a saruktól csak bizonyos távolságra kezdi alkalmazni a járókelőn, és egyre erősebben.

#### 4.1.2 Normák betartása

A járókelők egymástól távolságot tartanak, ha lehetséges. Ezt a többi járókelőtől egy taszító erővel írom le. Ennek az erőssége a közelséggel négyzetesen arányos, de nem gyakorolhat hatást ez egy adott erőnél erősebben. Az ajtóknál a kifele haladást végző emberek taszító erőt fejtenek ki a befelé haladni tervező emberekre. Ez az erő domináns a többi között, ugyanis ez az erő mindenkihez viszonylag egységesen hat. Nem csak a terem előtti tömeg első sorát képző emberek a zavaróak, hanem az összes. Ezért nem elegendő csak a távolság szerint a közelű emberekre hatni, mert akár 40 diák között is utat kell találnia a kifele áramló tömegnek. A közelű emberekre nagyobb hatást gyakorolnak külön, de csak 0-25%-kal.

### **4.1.3 Útvonal követése**

A járókelők az útvonalat követik, útponttól útpontig haladnak. Viszont probléma az, hogy mi történik, ha nem tud az úpontra lépni pontosan. Nem lehetséges erővezérelt mozgás során pontosan egy adott pontra lépni beavatkozás nélkül. Ezért szükséges valami terület, vagy határ, amit elegendő elérni, és onnantól feladata a járókelőnek a következő útpontot hasonlóan megközelítenie.

Ha a járókelő egy adott távolságra megközelíti az elérni kívánt útpontot, akkor továbbhaladhat a következőhöz. Ez a megoldás helyesen működik, ha nincsen tömeg. Tömeg esetén egy sarkon befordulni nem lehetséges mindenki számára, lehet, hogy a külső ívét képzi az embertömegnek a járókelő és a saroktól távol jut tovább az útján. Továbbá a haladási irány az adott pont felé folyton vonzaná az embereket, így szükséges, hogy egy folyosón párhuzamosan is képesek legyenek haladni. Erre megoldás, hogy a pontokat lehetőleg közelítse meg a járókelő, de a tömeg tasztóereje nagyobb lehessen adott távolság esetében. Ekkor a járókelő törekszik az útvonalának elérésére, de figyel a környezetére is. A párhuzamos haladás meg lett oldva így, viszont az útpont elérése nem. Megoldása ennek az, hogy az útpont elérése helyett túl is lehet haladni az útponton. Amikor az aktuális útponttól a távolsága már nő, míg a következő útponttól mért távolsága már csökken, akkor haladt túl az útponton. Ekkor a külső íven haladók is a kanyarban nem törekszenek visszafele haladni a tömegben. Ezzel folyamatos lesz a tömeg mozgása, és önmagát feleslegesen nem gátolja.

### **4.1.4 A járókelők napirendje**

A járókelőknek kell adni egy napirendet. Ez az egyetemen egy órarendnek feleltethető meg lényegében. A járókelőnek kell tudnia, hogy mikorra érkezzen meg az órájára, és hogy mikor hagyja azt el.

A szimulációban fontos, hogy néhány diák szembe haladjon a forgalommal, mert úgy természetes és a teljesség kedvéért is. Így az órára érkező diákok találkozhatnak az óráról távozó diákokkal. Ezt úgy oldottam meg, hogy a diákok az órarendjüköt egy bizonyos mértékben követik csak. Igazodnak hozzá, de kisebb eltérésekkel csak. Valaki korábban érkezik, valaki később, így a járókelők célja keveredik, amiből közlekedési szituációk keletkeznek, ezzel valóságosabb képet ad a szimuláció.

Napirendben egy elem a terem nevét, az óra kezdetét és az óra végének időpontját tartalmazza. Ezen elemek láncolata képzi a diákok napirendjét. A terembe érkezve a terem egyik helyét kiválasztja és lefoglalja magának. Távozásakor a helyet felszabadítja a következő csoport számára.

#### **4.1.5 Járókelők létrehozása**

A járókelőket a program egy fájlból olvassa be, amiben a járókelők napirendje és paraméterei találhatóak. A fájl szerkesztésére létrehoztam egy segéd programot, ami legenerál adott termekhez egy órarendeket és azokhoz az órákhoz egy létszámot. Majd a járókelőkhöz rendeli és a járókelőket maximális sebességgel és szélességgel látja le. Ez segíti a tesztelést a kész programnak.

Ezen program minimálisan, de elégsges szinten paraméterezhető. Meg lehet adni az egyszerre tartott órák számának maximumát, a szimuláció kezdetének és a végének az idejét, hogy hány órát lehet addig megtartani a terembe. A termeknek külön meg lehet adni a nevét és az átlagos kihasználtságát és a maximális kapacitását. Szélső eseteiben képes tömeget is létrehozni megfelelő paraméterek esetén, ahogy szinte üres termeket is.

#### **4.1.6 A járókelők életciklusa és az emelet létrehozása**

A járókelők a lépcsőházban „jönnek létre”, ott indulnak el útjukra. A napirendjük végeztével ide térnek vissza, és itt megszűnnek létezni. Ez a belépési terület az emeletez van eltárolva.

Az emelet adatait fájlból olvassa be a program, de lehetőség van beolvasás nélkül is az I épület 4. emeletének használatára is. Ebben a fájlban a betöltendő szobák, az emelet belépési területe és a szobák szomszédságai vannak eltárolva. Az emelet ezek eltárolását követően képes betölteni a szobákat, azokat szomszédolni, majd a járókelőket létrehozni az adott időpillanatban a belépési terület valamelyik pontján. Ezen járókelőknek biztosítja az emelet navigációs hálójának használatát.

## 5 Összefoglalás, eredmények értékelése

A program az elvártak szerint helyesen működik. Helyes használat esetén a program az alapvető követelményeknek megfelel. Nem áll le egy hiba miatt, folyamatos képet ad és a betöltést követően reagál a felhasználói bemenetekre.

A program tetszőleges szobák és a szobák berendezéseinek körvonalaiból - mint egy, a berendezési tárgyakat is feltüntető alaprajzból - képes navigációs hálót képezni. Képes a navigációs háló tetszőleges pontjára járókelőket lehelyezni. Képes a járókelőknek az emelet egy tetszőleges másik pontjához útvonalat generálni. Ezekre a program mind képes, legfeljebb elhanyagolható hibákkal.

A program nagyobb egységeit a készítésének folyamatában és a kész állapotában is többször futtattam, hogy sebességét vizsgáljam. A futási időtartamokat milliszekundumban mértem, mert nagyságrendileg ebben a mértékegységben kifejezhetők a kapott időtartamok. Egyes algoritmusok komplexitását kiszámítani pontosan nem tudtam, így csak a bemeneti paraméterek módosításával voltam képes becslést végezni hatékonyságukra. Két számítógépen is futtattam a programot a tesztelések során, hogy két különböző processzorral végezve két időtartamot is kapjak. A két kapott időtartamot összehasonlítottam, hogy lineáris-e a különbség a két számítógép között. Az egyik számítógép a sajátom, a másik számítógépnek a kari felhőben megtalálható Windows 10 20H2 CB+GIT+SDL virtuális gép sablonnal létrehozott gépet használtam.

### 5.1 A szobák betöltése és emeletté alakítása

A szobák betöltése és emeletté alakítása elhanyagolható futásidéjű probléma. Összesen 4 és 23 ms (milliszekundum) volt szükséges a betöltésükhez és kialakításukhoz az I épület 4. emeletének esetében. Ez a tetszőlegesen sok szoba esetén, példaként 100 szoba betöltése során sem változik számottevően. A szobák betöltése fájlműveettel jár, a hatékonysága nem lehet semmiképp sem  $O(1)$  (ordó), azaz konstans futásidéjű, de a futásideje elhanyagolható.

## 5.2 Az emelet navigációs hálójának létrehozása

A navigációs háló létrehozásának két része van. Az első részének feladata a terület háromszögekre bontása, a második részének feladata az előző rész háromszögeiből az útvonalkereséshez használható háló leképzése. Ezért ezt a két rész külön vizsgálom.

A navigációs hálót 100 szoba esetében a virtuális gépen  $\sim 16.000$  ms-be telt, míg a saját számítógépemen csak  $3.100$  ms-be. Ez az időtartam 10 szoba esetén, a két eszközön rendre  $\sim 1.600$  és  $316$  ms időt igényelt. A két-két értékből az algoritmus időkomplexitására lineáris futásidőre lehet következtetni a bemeneti paraméter méretéhez képest. Ennek az az oka, hogy szobákra bontottam a háromszögek generálását. Így adott számú szoba esetén adott számú többszöröse is lesz a futásideje az algoritmusnak.

A virtuális gépen a futási időt megmérni az algoritmusoknak sok időt igényel, mert változik a használható számítási kapacitása a felhőnek, így többször futtattam minden teszesetet a felhőben futó számítógépen. A mért leghosszabb időtartam az algoritmusoknál a legrövidebb időtartamhoz képest átlagosan 28%-kal volt nagyobb.

A navigációs háló leképzésének második része, amikor az algoritmus által használható hálót készíti el. A vizsgált 10-, 50-, és 100-szobás esetekben a saját számítógépemen 196 ms,  $\sim 4000$  ms és  $\sim 15.400$  ms alatt futottak le a tesztek, amíg a felhőben  $\sim 1.020$  ms,  $\sim 19.200$  ms és  $\sim 81.500$  ms alatt. Itt nem lineáris a bemeneti paraméter méretéhez képest az algoritmus futásideje. Az algoritmus több ciklusból áll, amelyek előkészítik az azt követő ciklushoz az adatokat. A háromszögek létrehozása igényli ezt a sok időt.

A háromszögek oldalfelező pontjait fejtem ki sorban és azok szomszédjait felhasználva képzem a háromszögeket. A háromszögek egyediségét minden egyes új háromszögnél ellenőrzöm a meglévő háromszögek között. Ennek megoldása lehetne a háromszögeket set-ben eltárolni, aminek felépítése nem lenne ilyen költséges, viszont a háromszögek egyediségéért felel. Ekkor a háromszög sorszámát el kellene tárolnom a háromszögben, mert a set nem őrzi meg a sorrendet, hiszen a csúcsok alapján sorba rendezzi. A háromszögek sorszámát, vagy azonosítóját az útvonalkereséshez felhasználom, így fontos, hogy legyen.

### **5.2.1 Az algoritmus hatékonyságának növelése**

Az algoritmus a futása során háromszögeket generál. A generált háromszögek egyediségét meg kívánom őrizni, ezért egy azonosítót adok neki. Eddig nem létező háromszöget nem hoz létre az algoritmus, mert a csúcsok és a csúcsok szomszédságai miatt kötöttek, ezért a háromszögek súlypontjai egyértelműen azonosítják a háromszöget. A háromszögek egyediségének ellenőrzését egy set-ben történő kereséssel sikerült úgy megvalósítanom, hogy az algoritmust szinte egyáltalán nem változtattam meg. A létrehozott háromszögekhez eltároltam a súlypontjukat, és egy vektoron kívül egy set-ben is eltároltam őket. A vektorban eltárolt háromszögeket felhasználom a programomban továbbra is, a set-ben eltárolt háromszögekre az egyediséget tudom ellenőrizni hatékonyan, és az algoritmus lefutását követően nem is használom.

Ezt követően a saját számítógépemen 28 ms, 150 ms és 310 ms alatt futottak le a 10, 50, és a 100 szobás tesztek. A felhőben futó számítógépen ezen értékek rendre 160 ms, 1.050 ms és 2.020 ms voltak. Ezekből az értékekből látható, hogy a futásidő a bemenet méretével egyenesen arányos, így a hatékonyságát sikerült növelnem az algoritmusnak.

## **5.3 Az útvonalkeresés**

Az útvonalkeresést szobákra bontottam. Ha a kezdőpont és a végpont egy szobán belül található, akkor a szobán belül végzi az algoritmus az útvonalkeresést. Ha több szobán is keresztülvezet az út, akkor a köztes szobákban az ajtók között számítja ki az útvonalat, illetve a kezdőpontot, és a végpontot összeköti a szobák ajtajával. Ezen útvonalak láncolata adja meg végül az útvonalat. Ezért az útvonal kiszámítása a szobák komplexitásával és a szobák számával nő, viszont csak lineárisan.

Futásidőről külön csak két számadatot gyűjtöttem ki. Amikor nem osztottam szobákra az útvonalkeresést, akkor ~1.080 ms alatt sikerült az I épület 4. emeletének 6 terméből összesen 181 diáknak az útvonalát kiszámítani a lépcsőházhöz. Ez az időtartam a szobákra osztás esetén 220 ms-ra csökkent, ami 5-szörös gyorsítást eredményezett.

## 5.4 Összefoglalás

Nem találtam információt arról, hogy a navigációs háló felépítéséhez mi a leghatékonyabb algoritmus. Ennek okán a programom algoritmusait csak önmagukban tudom értékelni. A bemenet méretével egyenesen arányos futási idő a hatékonyság szempontjából jó. Tökéletes a konstans futásidőjű algoritmus lenne, a kiválóról a logaritmikus vagy négyzetgyökös algoritmusok esetében beszélnek, viszont mivel az előzőek egyike sem, és nem is exponenciális, ezért a munkámmal összességében elégedett vagyok, mert az útvonalkeresés egésze jó. Minimális levágásokat végez csak a bejárható téren kívülre, de azok elhanyagolhatóak.

## 6 Irodalomjegyzék

- [1] „Thunderhead Engineering Pathfinder,” Thunderhead Engineering, 09 12 2022. [Online]. Available: <https://www.thunderheadeng.com/pathfinder>.
- [2] „Top 10 programming languages in 2022,” IEEE Spectrum, [Online]. Available: <https://spectrum.ieee.org/top-programming-languages-2022>. [Hozzáférés dátuma: 05 12 2022].
- [3] „C++ reference,” [Online]. Available: <https://en.cppreference.com/w/>. [Hozzáférés dátuma: 05 12 2022].
- [4] „Simple DirectMedia Layer Főoldal,” SDL, [Online]. Available: <https://www.libsdl.org/>. [Hozzáférés dátuma: 05 12 2022].
- [5] „PAC-MAN Főoldal,” Bandai Namco Entertainment Inc., [Online]. Available: <https://www.pacman.com/en/>. [Hozzáférés dátuma: 05 12 2022].
- [6] M. Kartika, „Dijkstra’s Algorithm Application on the Pac-Man Game,” Bandung, 2010.
- [7] „Toward More Realistic Pathfinding,” Game Developer, [Online]. Available: <https://www.gamedeveloper.com/programming/toward-more-realistic-pathfinding>. [Hozzáférés dátuma: 05 12 2022].
- [8] M. Karlsson, „A Navigation Mesh-Based Pathfinding Implementation in CET Designer - An Alternative to a Waypoint Graph-Based Solution,” Linköping University | Department of Computer and Information Science, Linköping, 2021.
- [9] „Navigation and Pathfinding,” Unity, [Online]. Available: <https://docs.unity3d.com/Manual/Navigation.html>. [Hozzáférés dátuma: 05 12 2022].
- [10] „What’s an Average Shoulder Width?,” healthline, [Online]. Available: <https://www.healthline.com/health/average-shoulder-width>. [Hozzáférés dátuma: 05 12 2022].

- [11] J. Y. Y. W. S. L. Z. Fang, „Survey of pedestrian movement and development,” Wuhan University, Wuhan, 2006.
- [12] J. L. Z. Fang, „On the relationship between crowd density,” Wuhan University, Wuhan, 2001.
- [13] W. D. W. J. L. B. Philip J. DiNenno, „Behavioral Response,” in *SFPE Handbook of Fire Protection Engineering - Third Edition*, Quincy, Massachusetts 02269, One Batterymarch Park, National Fire Protection Association, 2002, p. 3–315.
- [14] K. Nahtkasztlija, „Az idegen szavak toldalékolása,” június 2009. [Online]. Available: <http://www.pcguru.hu/blog/kredenc/az-idegen-szavak-toldalekolasa/5062>.
- [15] P. Koopman, „How to Write an Abstract,” október 1997. [Online]. Available: <https://users.ece.cmu.edu/~koopman/essays/abstract.html>. [Hozzáférés dátuma: 20 október 2015].
- [16] W3C, „HTML, The Web’s Core Language,” [Online]. Available: <http://www.w3.org/html/>. [Hozzáférés dátuma: 20 október 2015].

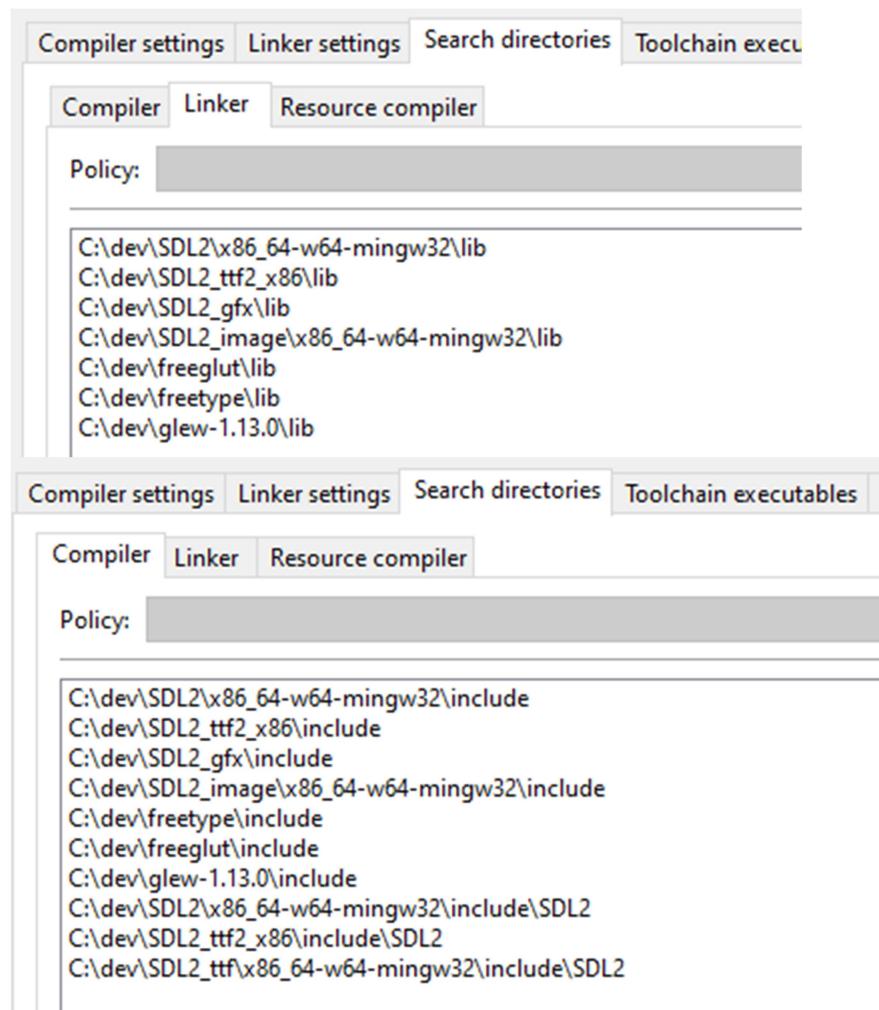
## 7 Függelék

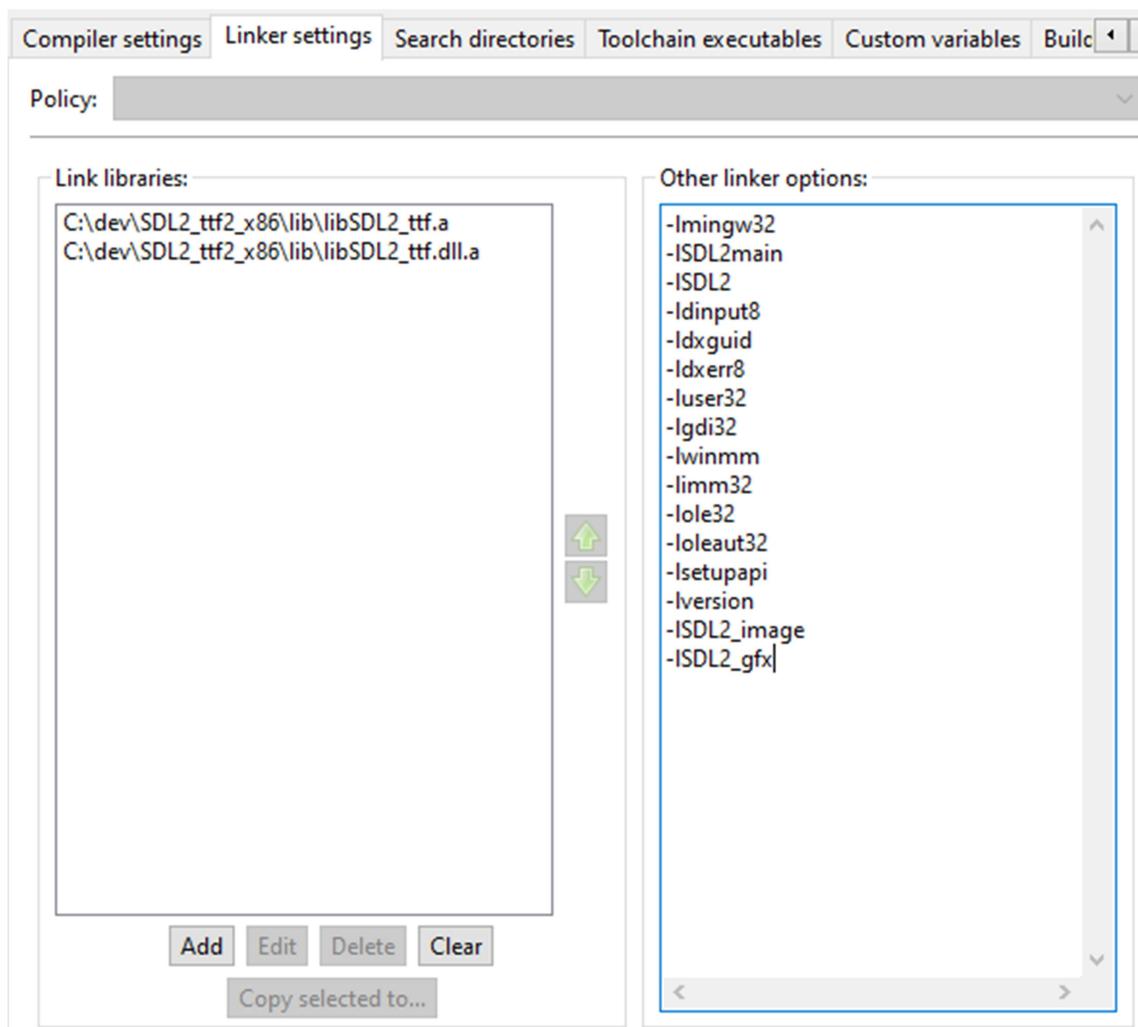
A programomat egy verziókezelő rendszerben (GitHub) vezettem a fejlesztés során. A programot az alábbi linken elérhetik:

<https://github.com/Blextor/SzakdolgozatPathFinding>

A feltöltött forrásfájlok fordításához és futtatásához én a Code Blocks MinGW 20.03-as verziójú fejlesztői környezetet használtam. Az SDL2 könyvtárát kell a projekthez include-olni és a fordítás lehetséges. Én a linken elérhető dev mappában található dev.zip fájlba töltöttem fel az include-olt könyvtárakat. Csatolom a „további linkelési opciókat” a másolhatóság kedvéért. Megtalálható az említett dev mappában tlo.txt néven.

Az include-olást követően hasonló állapotot kell látni a projekt beállításaiban:





Ezt követően a futtatás és a fordítás már működik.

A programok használati leírását a dev mappában megtalálható hasznalati.txt néven.