

Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Kovács Boldizsár

Járókelők mozgásának szimulációja

Konzulens

Dr. Goldschmidt Balázs

BUDAPEST, 2022

Tartalomjegyzék

[Összefoglaló 6](#_Toc433184091)

[Abstract 7](#_Toc433184092)

[1 Bevezetés 8](#_Toc433184093)

[1.1 Frissítsd a dokumentumot 8](#_Toc433184094)

[1.2 Szakdolgozat, vagy diplomaterv 8](#_Toc433184095)

[1.3 Témaválasztás 8](#_Toc433184096)

[2 A dolgozat szerkezete 10](#_Toc433184097)

[2.1 Fejezetek 10](#_Toc433184098)

[2.1.1 Feladatkiírás 10](#_Toc433184099)

[2.1.2 Címoldal 10](#_Toc433184100)

[2.1.3 Tartalomjegyzék 10](#_Toc433184101)

[2.1.4 Nyilatkozat 11](#_Toc433184102)

[2.1.5 Tartalmi összefoglaló 11](#_Toc433184103)

[2.1.6 Bevezetés 11](#_Toc433184104)

[2.1.7 Irodalomkutatás, technológiák, hasonló alkotások bemutatása 12](#_Toc433184105)

[2.1.8 A feladatkiírás pontosítása és részletes elemzése 12](#_Toc433184106)

[2.1.9 Önálló munka bemutatása 12](#_Toc433184107)

[2.1.10 Önálló munka értékelése, mérések, eredmények bemutatása 13](#_Toc433184108)

[2.1.11 Összefoglaló 14](#_Toc433184109)

[2.1.12 Köszönetnyilvánítások 14](#_Toc433184110)

[2.1.13 Részletes és pontos irodalomjegyzék 14](#_Toc433184111)

[2.1.14 Ábrajegyzék, táblázatjegyzék 14](#_Toc433184112)

[2.1.15 Függelék 14](#_Toc433184113)

[2.2 Egyéb tartalmi elemek 15](#_Toc433184114)

[2.2.1 Stílus 15](#_Toc433184115)

[2.2.2 E/1 15](#_Toc433184116)

[2.2.3 Rövidítések 15](#_Toc433184117)

[2.2.4 Technológia megválasztása 16](#_Toc433184118)

[3 Formázási tudnivalók 17](#_Toc433184119)

[3.1 Általános tudnivalók 17](#_Toc433184120)

[3.2 Stílusok 17](#_Toc433184121)

[3.3 Címsorok 18](#_Toc433184122)

[3.4 Másolás, beillesztés 18](#_Toc433184123)

[3.5 Mezőfrissítés 19](#_Toc433184124)

[3.6 Helyesírás 19](#_Toc433184125)

[3.6.1 Elgépelések 19](#_Toc433184126)

[3.6.2 Egyeztetés hiánya 19](#_Toc433184127)

[3.6.3 Külföldi szavak, kifejezések 19](#_Toc433184128)

[3.6.4 Stb 20](#_Toc433184129)

[3.6.5 Helyesírás ellenőrző 20](#_Toc433184130)

[3.7 Képek 21](#_Toc433184131)

[3.7.1 Beszúrás, formázás 21](#_Toc433184132)

[3.7.2 Képminőség 21](#_Toc433184133)

[3.8 Kereszthivatkozások 22](#_Toc433184134)

[3.9 Irodalomhivatkozások 23](#_Toc433184135)

[3.9.1 Pozícionálás 23](#_Toc433184136)

[3.9.2 Mikor kell hivatkoznom? 24](#_Toc433184137)

[3.10 Word tippek és trükkök 24](#_Toc433184138)

[3.10.1 Navigációs ablak 24](#_Toc433184139)

[3.10.2 Megjegyzések 24](#_Toc433184140)

[3.10.3 Korrektúra 25](#_Toc433184141)

[3.10.4 Gyorsbillentyűk 25](#_Toc433184142)

[3.11 Kódrészletek 25](#_Toc433184143)

[3.11.1 Formázás 26](#_Toc433184144)

[3.11.2 Irodalomjegyzék 27](#_Toc433184145)

[3.12 Utolsó simítások 27](#_Toc433184146)

[4 Irodalomjegyzék 28](#_Toc433184147)

[Függelék 29](#_Toc433184148)

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

Összefoglaló

A szakdolgozat, vagy diplomaterv elkészítése minden egyetemi hallgató életében egy fontos mérföldkő. Lehetőséget ad arra, hogy az egyetemi évei során megtanultakat kamatoztassa és eredményeit szélesebb közönség előtt bemutassa, s mérnöki rátermettségét bizonyítsa. Fontos azonban, hogy a dolgozat elkészítésének folyamata számos csapdát is rejt magában. Rossz időgazdálkodás, hiányos szövegszerkesztési ismeretek, illetve a dolgozat készítéséhez nélkülözhetetlen „műfaji” szabályok ismeretének hiánya könnyen oda vezethetnek, hogy egy egyébként jelentős időbefektetéssel készült kiemelkedő szoftver is csak gyengébb minősítést kapjon a gyenge minőségű dolgozat miatt.

E dokumentum – amellett, hogy egy általános szerkesztési keretet ad a dolgozatodnak – összefoglalja a szakdolgozat/diplomaterv írás írott és íratlan szabályait. Összeszedjük a Word kezelésének legfontosabb részeit (címsorok, ábrák, irodalomjegyzék stb.), a dolgozat felépítésének általános tartalmi és szerkezeti irányelveit. Bár mindenkire igazítható sablon természetesen nem létezik, megadjuk azokat az általános arányokat, oldalszámokat, amelyek betartásával jó eséllyel készíthetsz egy színvonalas dolgozatot. A részletes és pontokba szedett elvárás-lista nem csupán a dolgozat írásakor, de akár más dolgozatok értékelésekor is kiváló támpontként szolgálhat.

Az itt átadott ismeretek és szemléletmód nem csupán az aktuális feladatod leküzdésében segíthet, de hosszútávon is számos praktikus fogással bővítheti a szövegszerkesztési és dokumentumkészítési eszköztáradat.

Abstract

Ide jön a ½-1 oldalas angol nyelvű összefoglaló, amelynek szövege a Diplomaterv Portálra külön is feltöltésre kerül. Ez a magyar nyelvű összefoglaló angolra fordított változata.

# 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 kimondva, együtt kell dolgozni velük. Ezek nélkül az együtt élés lehetetlen lenne. Remek példa erre a közlekedés.

Fontos, hogy az emberek legyenek egymásra tekintettel, ezekben segítenek a normák. Fontos ugyan úgy, hogy a helyzetfelismerő képességük is helyes legyen, ehhez figyelniük kell, és alapvető ismeretekre van szükségük. Ahhoz, hogy a helyzetfelimserése 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 terepobjektum milyen lehetőségekre korlátozza a továbbhaladását.

Megfelelőnek kell 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 elérjék a buszt. Ezen esetekben mind 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, mert fal vagy más eltakarja. 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 ezen problémák megmaradnak komplex feladatoknak. Számos tényt és okot figyelembe kell venni ahhoz, hogy a valósághoz hasonlót sikerüljön lemodellezni az emberek mozgásával kapcsolatban. Hogy az emberek mozgása végül természetesnek hasson, és a valóságot minél jobban megközelítse temérdeknyi kutatás és adat szükséges, hogy akár csak kezdetleges megoldást, modellt készítsünk, ami leírja az emberek - a közlekedésben a járókelők - mozgását.

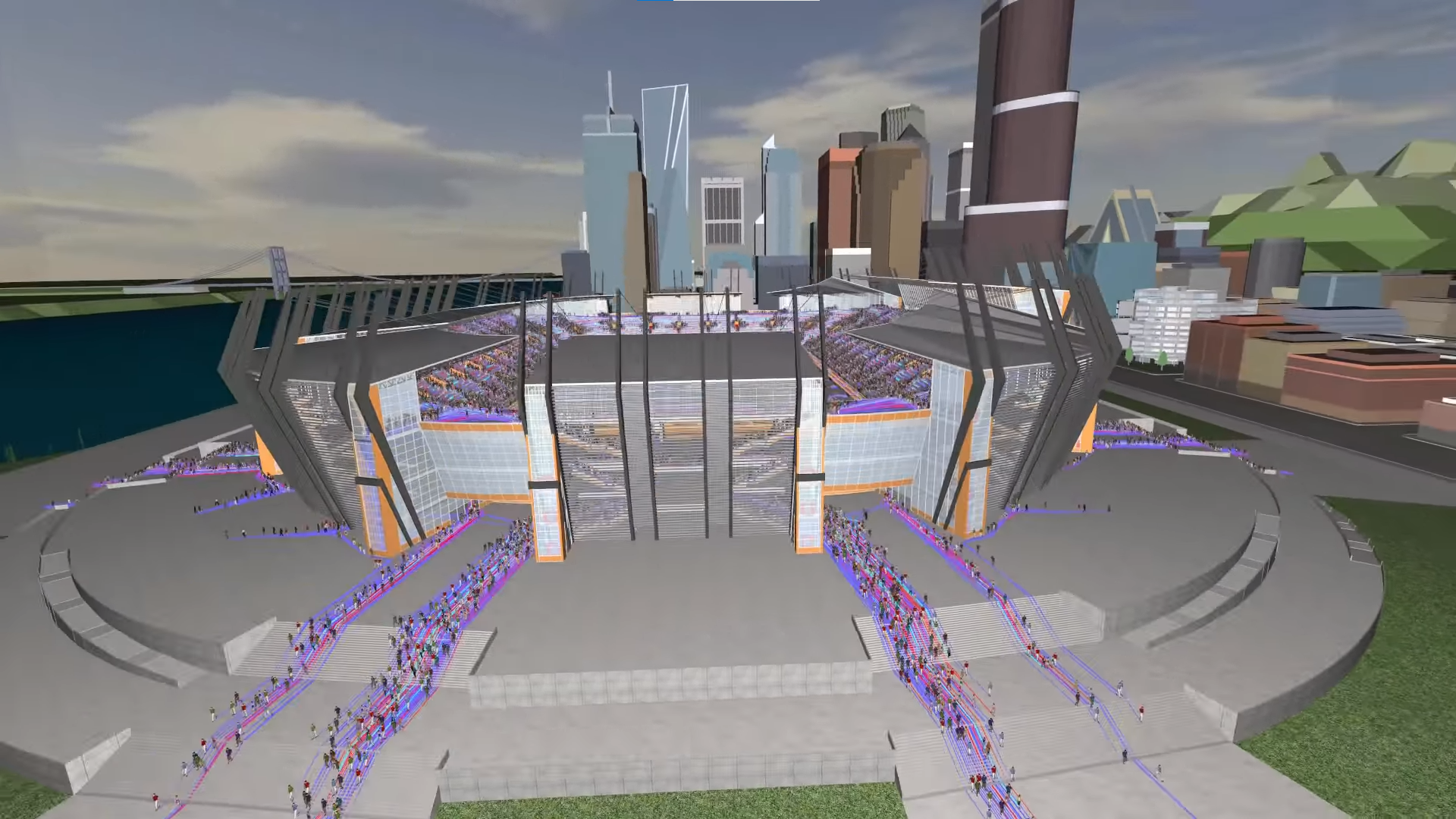
## Frissítsd a dokumentumot

Amikor az emberek számára egy pillantás is elegendő, hogy felmérjék a közlekedési szituációt, miért szükséges bevonni az informatikát, amelyben ezen probléma megoldása nehezebb. Ennek oka, hogy gyakran nincs lehetősége az embereknek egy pillantást vetni a környezetre, esetenként nincs még meg a környezet sem, vagy automatizálásra van szükség valaminek az ellenőrzéséhez. Azért fontos foglalkozni azzal, hogy helyesen lehessen szimulálni a járókelők mozgását, hogy a szimulációk futtatása során kiderüljön az, hogy ha a tervekben valami nem a legalkalmasabb az emberek mozgásra nézve, akkor javítva azokat az emberek élete kellemesebb, és komfortosabb lehet.

Rengeteg tervezés előz meg egy épület megépítését, egy villamos összeszerelését, vagy egy kereszteződés átalakítását. Azért szükséges a tervezéshez sok idő, hogy a lehető legjobb lehessen a végeredmény, és minél kevesebb problémája legyen a kész állapotban. Egy épület esetében nem csak az a fontos, hogy minél magasabb vagy épp impozánsabb legyen, hanem többek között az is, hogy a legapróbbnak tűnő dolgok 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. Hogy a tervezés végére minden a rendeltetésének megfelelő méretű legyen, ahogy legyen ergonómikus elrendezésű is és az embereknek természetesnek ható és komfortos.

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

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, amely közül az egyik, hogy tűz esetén várhatóan mindenkinek sikerüljön elhagynia biztonságban az épületet. Ezt kisebb épületeknél, nagyjából 500 főig tűzriadó próbával is meg lehet vizsgálni, ezen létszám fölött jellemző, hogy szükséges a terveknél is ezt már leszimulálni. Erre a Thunderhead Engineering Pathfinder\* programja használt az iparban Magyarországon, mert ezen programnak a terveket megadva, és az emberek paramétereit meg nem változtatva elfogadott a szimulációban elért eredménye a tűzriadónak. Egész stadionok tesztelésére is alkalmas.[ábra]



. ábra

Kell számolni azzal is, hogy a tervezett objektum kiknek készül, például betegeknek, idősebbeknek, óvodásoknak vagy szélesebb embereknek. A lépcsők az idősebbek számára fontos kérdés, míg a szélesebb emberek számára meg a tömegközlekedésen a székek szélessége fontos például. Kórházak és egyéb közintézmények esetén az akadálymentes közlekedést kötelező megoldani.

Ezzel mind lehet, és egyes esetekben kell is a tervezés során foglalkozni.

Egy olyan programon dolgoztam, ami a járókelők mozgásának szimulációjára alkalmas, főképp, de nem szükségesen csak zárt terekben. Arra használhatón, amivel egy adott épületnek egy adott emeletéről szerezhetünk információkat. Például a tervezett maximális kapacitás esetén hogy zajlana le egy tűzriadó, vagy csak a hétköznapokban hol okoz fennakadást egy szűkület, mely ajtóknál vagy sarkoknál.

Ehhez szükségem volt az emberekre általánosságban elmondható tulajdonságok összegyűjtésére, a különböző paramétereiknek az átlagos értékeiknek meghatározására, továbbá a közöttük lévő kapcsolatok megismerésére. Gondolok paraméterek alatt a sebességükre, a korukra, a magasságukra, a szélességükre, a privátszféra kiterjedtségükre 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.

## A feladat

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

Kezdetben a feladatom, hogy a problémafelvetéssel és az irodalomkutatással kezdjem. A kettőt szükséges együtt végezni, ugyanis a már meglévő, mások által írt munkákban a problémakör nehézségére gyakran kitérnek. Ennek okán olyan feladatokat és problémákat volt feladatom találni a témán belül, amikkel a problémaköröm kibővíthetem. Ezt követően folytassam az irodalomkutatást, hogy a kibővített problémakörre találjak már létező megoldásokat, vagy csak segítséget ahhoz, hogy ki tudjak indulni valamiből. Mások munkáit olvasva a program egyes részproblémáinak megoldására próbáljak egyre több féle megközelítést találni, és ezeket lehetőleg hasonlítsam majd össze, és jussak valami következtetésre.

Az így keletkezett megoldásokat fel tudjam használni majd a programomban és végül egy bizonyos szempontból egy optimális megoldással tudjak előállni. A szerzett ismeretek alapján ezt követően 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.

Tervezés közben teszteljem a megoldásomat gyakran, és ha lehet, a problémát osszam többrészre, hogy elkülönítve, a tesztelés csak egy-egy egységre történjen egyszere. Ezzel segítsem azt, hogy a program felépítése legyen egységekre bontva, és megfelelően szervezve. Ezt mind a működőképességért, az áttekinthetőségéért, és a jövőbeli továbbfejlesztési lehetőség fenntartása érdekében tegyem meg.

A program elkészítését követően vizsgáljam meg a működését, és teszteljem helyességét. A szimuláció paramétereit a valóságnak megfelelően választva vessem össze, hogy az eddigi ismereteinkkel összeegyeztethető-e az eredmény. Ha nem az elvárt eredmény kapom, vizsgáljam át esetleges hibák után kutatva a programom, vagy indokoljam meg, hogy mi okozhatja a eltérést.

Program készítése során feladatom már létező algoritmusok részletesebb megismerése, és számomra specifikus algoritmusok kifejlesztése és létrehozása. Ezen algoritmusok ne csak gyorsak legyenek, hanem lehetőleg hatékonyak is, és lehetőleg hasonlítsak össze több megoldást is. Ezen 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.

## Irodalomkutatás és a feladatkör kiterjesztése

### Programozási környezet megválasztása

Kiindulásként a programozási nyelvet kívántam megválasztani. Olya nyelvet, ami a feladathoz megfelelő és az iparban elterjedt a mai nap is. Egy széleskörben elterjedt, az számos iparágban\* jelenleg is használatos nyelvet választottam így, amiben lehetséges objektum orientált szemléletet követni. Az algoritmusok hatékony és gyors működése érdekében szükségesnek tartottam egy gyors nyelvet, hogy ezen tényező biztosan ne legyen hátráltató tényező. Ezért döntöttem a C++\* nyelv mellett.

A megjelenítéshez kellett választanom egy grafikus könyvtárat, a Simple DirectMedia Layer 2-t (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 bőven 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.

### Útvonal keresés

Ezt követően, előnyben részesítve a választott nyelvet, elkezdtem keresni a témában munkákat, cikkeket, dolgozatokat, hogy a program tervezésénél minél több problémára tudjak figyelni és szerzett ismeretek alapján 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 nagyon régi példát említve már több, mint negyven éve alkották meg az első PAC-MAN\*-t, amelyben a játékost a játéktéren szellemek próbálták elkapni. Ebben a példában a szellemekből négy volt és mindegyiknek saját logikája volt.\* A szellemek útvonalkeresését Dijkstra algoritmus\* segítségével is valósíthatták meg.\* Az algoritmus alapját a későbbiekben felhasználtam, így teljesség kedvéért kifejtem.

Az algoritmust 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 példánkban nincs erre precedens, így alkalmas megoldás az útvonalkeresésre, továbbá összefüggő is a gráfunk így kevesebb problémát kell megvizsgálnia az algoritmusnak.

Ugyanis a játéktér tekinthető egy négyzetrács hálónak, és ha jobban megnézzük, akkor ezen háló szabad mezőiből tudunk csoportokat alkotni, amelyeket élekként és csúcsokként gráfba tudunk szervezni. Az algoritmus mindig a legrövidebb útvonalat adja vissza a gráfban, és futásideje is kedvező. Az algoritmus egy gráfot használ keresési tereként, és két a 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, és az ő szomszédjainak az értékét úgy adja meg, hogy nem csak a csúcshoz vezető él súlyát veszi figyelembe, hanem a azt is, hogy magának a csúcsnak mekkora az értéke. Azaz a csúcshoz vezető út éleinek súlyainak mennyi az összege a kezdőpontból. [ábra]

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



Fontos, hogy ha annak a csúcsnak, melynek értéket ad, már van értéke, és az kisebb vagy egyenlő, akkor nem bírálja felül, és az értékét nem változtatja meg. Ellenkező esetben felülírja a kisebb 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 fejtené ki. Ekkor ugyanis a végpont csúcsának értéke adja meg, a legrövidebb utat 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 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 tudnánk meghatározni, csak az út hosszát. Ezért fontos - például - a felfedezett csúcsokhoz eltárolni 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 ezen út így előáll a visszafele haladva megvizsgált csúcsokból és a legrövidebb is lesz a két csúcs között.

Ezen algoritmus nagyon közel áll egy jelenleg is elterjedt, a játékiparban standardnak tekinthető, A\* (kiejtve „a csillag”) algoritmushoz. Ennek az algoritmusnak több átdolgozott verzióját is használtam a programomban végül. Az Dijkstra algoritmus publikálását követően tizenkettő évvel később publikálták az A\*-ot először, mint az Dijkstra algoritmus kiegészítése. Az algoritmus abban volt több, és a gyakorlatban a későbbiekben sokkal hatékonyabb, hogy a csúcsokhoz rendelt értéket új heurisztikával adta meg. Az A\*-ot akkor lehet használni, ha létezik olyan költség becslést végző függvény, amely a végpont csúcsáig tetszőleg csúcsból képes olyan távolságot megbecsülni, amely sosem lehet több, mint a tényleges út hossza, abból a csúcsból. Ekkor ugyanis a heurisztika a következő. A csúcs értékének ne az odáig vezető út éleinek súlyát adjuk meg, hanem külön adjuk hozzá ennek 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, hanem az előző csúcs érétkének a becslést végző függvény értékének hozzáadását megelőző érték, az összekötő él súlya és a becslő függvény értéke az új csúcsra adja majd.

Az algoritmus továbbra is egy csúcs kifejtése során minden szomszédos csúcsot kifejt, viszont akkor áll le, amikor először felfedezi a végpont csúcsát. Ehhez viszont 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 távolságát a térben. Erre lehetőség van olyan esetekben, amikor a gráf csúcsai valamilyen teret írnak le. Ilyen tér például lehet egy olyan gráf, ahol a csúcsok városok, az élek meg a városokat összekötő út hossza. Ekkor a távolságbecslő függvénynek alkalmas megválasztani a városok távolságát légtávolságban.

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 lévő átjáró a játéktér egyik oldalát összeköti a másik oldalával. Ekkor ami a képernyőn egy egység távolságnak tűnik, az igazából az átjáró okán lehet nulla is. Ennek okán 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 különbségként az hozható fel, hogy a legrövidebb út a térképnél az egyenes, addig 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.

Azért fejtettem ki ezen két algoritmust részletesen, mert a kutatás során fontos volt a megismerésük, és a programomban az A\* algoritmust számos alkalommal használom fel. Azért elterjed az iparban ez az algoritmus, mert ha lehet helyes költség becslő függvény találni, akkor az algoritmus hatékonysága kiugróan magas. A legrövidebb út megoldására alkalmas algoritmusok időbeli komplexitás a csúcsok és az élek számával függenek össze. Az a A\* algoritmus komplexitása viszont a heurisztikától nagyban függ, így csak konkrét eseteknél lehet ezt meghatározni. Az útvonalkeresés elengedhetetlen része manapság az A\* algoritmus.

### Helyszín átalakítása gráffá

Ezt követően fontosnak tartottam, hogy legyen is olyan gráfom, amin lehet az algoritmust futtatni. Ahogy a példákkal mutattam, nem minden esetben lehet kihasználni az A\* heurisztikáját, viszont az ember mozgásánál ezzel nincsen probléma. Ahogy a valóságban, úgy a programomban is a legrövidebb út az egyenes lesz. A különböző komplexitású problémákra különböző gráfokat sikerült találnom.

Egyik példának a sakkot találtam, ha a királlyal lépek. Ekkor a játéktér véges, és a király helyzete csak diszkrét és véges sok lehet. Összesen hatvannégy mezőn állhat, és az A\* algoritmus is helyesen alkalmazható, mert a király lépésére elmondható, hogy legrövidebb út, az egyenes. Kifejezetten nem mondható ez el a király helyett a lóról 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 éllel 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. Ezen megoldást annak ellenére megvizsgáltam, hogy nem írja le a valóságot, mert egy ember tetszőlegesen sok helyen állhat egy szobában, nem csak pontosan hatvannégy helyen.

Izgalmasabb problémának tűnik az, amikor a gráf egy épület emeletének, vagy csak egy szobának a bejárhatóságát írja le. 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.\*

Az útpont gráf azt a problémát kívánta megoldani, hogy ne végtelen csúcsa legyen a gráfnak. Ugyanis a végtelen csúcs közül csak keveset tart meg, és a megmaradt csúcsok viszont elnagyolt, de jó képet adnak arról, hogy miként lehet bejárni a teret. Lényegében ezen pontokat követve el lehet jutni az emelet esetében tetszőleges szobából, egy másik szobába. Figyelemmel van arra, hogy a gráf élei bejárhatóak legyenek, azokon végigsétálva falba, vagy más terepobjektumba ne ütközzünk. A csúcsai arra alkalmasak, hogy közülük kiválasszuk a legközelebbit hozzánk, és kiválasszuk végcélunkhoz is a gráf legközelebbi csúcsát, majd a közöttük kiszámított legrövidebb útvonalat, a gráf adott éleit kövessük.

Ezen 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 életszerű a szimuláció. Ugyanis ha mindenki az éleken haladva menne végig a csúcsokon, akkor az emberek mozgása a hangyákéra hasonlítana. Egymást követve vagy szemben lényegében egy sorban haladnának. Az emberek viszont kihasználják a teret. Továbbá a nem megfelelő gráf megalkotása esetén az útvonal lehet nagyon szögletes, nem természetes, és se nem a legrövidebb, se nem az ember szokásos útvonala. Ennek okán komoly feladat az, hogy egy tökéletes ilyen gráfot megalkosson az ember.

Sajnos van egy problémája is. Véges csúcs okán 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 azon végighalad, akkor az utolsó csúcstól el kell találnia valahogyan a saját céljáig. Ez viszont azt vonja magával, hogy két útvonalkereső algoritmust is kell használnia a programnak. Természetesen ezen megoldás hibáinak a nagy részét az képzi, hogy automatikusan generálni ilyen hálót komplex probléma. Viszont ha sikerül, akkor kifejezetten hatékony, de az emberek természetes mozgatása továbbra is kérdése benne.

A másik létező megoldás, a navigációs háló, angol nevén Navigation Mesh (NavMesh). \*\* Ez az eredeti problémának a végtelen csúcs problémáját úgy orvosolja, 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. Ekkor a helyszín minden bejárható pontja valamelyik háromszög élére vagy belsejébe esik. A háromszögek nem lapolódnak át, így könnyű meghatározni, hogy a háromszögek melyikébe esik az adott pont.

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

Ennek a megoldásnak szintén nehézsége, hogy a hálót létre kell hozni. Rengeteg megoldás van erre a problémára. Lehet például mindig 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 behúztuk az új szakaszt, elmentjük a többi behúzott szakasszal, és azt az alaprajz részének tekintjük onnan, és az előbb említett két élt kitöröljük. Mivel minden egyes él behúzása után kettőt törlünk, ezért lépésenként csökkennek a módosított alaprajz éleinek száma. Addig kell behúzni így szakaszokat, míg egy háromszög marad csak. Akkor az algoritmus leáll, és az elmentett behúzott éleket felvesszük a gráfba. Ekkor a gráf éleiből háromszögeket képzünk és kész is van a navigációs háló.

Ezen a háló azért jól használható, mert szomszédos konvex sokszögekből épül fel. Nem szükséges háromszögekkel dolgozni, lehet téglalapokkal vagy 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 síkidomban a kezdő és végpontok, akkor a síkidom láncon keresztül kell az útvonalat kialakítani.

Komoly probléma az útvonal kiegyenesítése a legrövidebb út meghatározásában mind a két hálóban. Itt bonyolult számítást igényelhet nem csak a háromszögek oldalain, mint gráfon lefuttatni az A\*-ot, hanem ki is egyenesíteni az útvonalat. Az útpont gráfban a gráf pontjaival vagyunk csak képesek dolgozni, ami csak véges lehetőséget biztosít, így ritkán adja vissza a legrövidebb utat.

A programomban a navigációs háló megvalósítása, és használata mellett döntöttem. Ugyanis 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\* a gép által vezérelt entitásokat ennek segítségével navigálja.

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

### Emberek mozgása

Most, hogy tudatában vagyok annak, hogy milyen módon tudok útvonalat generálni, fontos lenne ezt egy embernek végig is sétálni. Ennek okán az emberek mozgásával kapcsolatos információk után 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.

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\* 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 ezek alapján majd 30 cm és 50 cm közötti vállszélességű emberekkel fogok dolgozni. Az más ország adatait ezen határok közé feltételezem. Egyetemi körülményeket tervezek szimulálni, ahol az emberek viszonylag nyugodtak, mozgásuk során nem futnak, ha nem 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.

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.\* Ennek okán, ha egyedül mozog az illető, haladhat akár 2 m/sec-mal de akár csak 1.6 m/sec-mal is. Ahogy a tömegben szintén eltérő eredményeket kaphatunk különböző mérések során. Az ember tömeg 1 ember/m^2 sűrűség esetén az egyén sebességét 0.7 m/sec-tól egészen 1.3 m/sec-ig várhatjuk. Ez az intervallum 3 ember/m^2 esetén már csak 0.2 m/sec-tól 0.8 m/sec-ig tart. És egészen 7-8 ember/m^2 -ig lehet számolni azzal, hogy még a tömeg lényeges mozgást végez.

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 jelenlétük. Ennek okán, 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 épülettűzből levont tapasztalat alapján ki tudjuk kijelenteni.\* 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.

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 sebességük le is csökken. Ennek megoldása komplex feladat, és csak próbálkozást teszek a programomban ennek figyelembe vételére.

### És amit már nem említek meg

Sok munkát olvastam át, amelyekkel nem értettem egyet és nem találtam hasznosnak. Vagy a problémakörbe nem ásták bele kellően magukat, vagy megelégedtek részeredménnyel, ennek okán az eredményük helytelen, vagy pontatlan lett, amit nem kívántam felhasználni. Ennek okán rengeteg kérdés és tervezési feladat maradt meg számomra. Ennek ellenére a problémakört tudtam szélesíteni, és az ismereteimet bővíteni.

# Saját munka bemutatása

Ahhoz, hogy egy programot létrehozzak érdemes megtervezni azt előtte. Ezzel elősegítem 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 lefejleszteni. Ennek első lépése, hogy specifikációt készítsek. Azaz írjam le, mit kell tudnia majd a programomnak, mit kell megvalósítanom a tervezés során.

## 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. Nem szabad gyakran váratlanul leállnia, vagy a külső beavatkozásra 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 könnyen feldolgozhatóan ábrázolja. Lehessen különböző paraméterű, például különböző szélességű és sebességű embereket használni, de elég valós helyzetekre felkészíteni a szimulációt. Nem kell dolgoznia 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.

## Emberek ütközésének elkerülése

A valósághű szimuláció megalkotása nehéz feladat. Hatékony megoldást találni rá még bonyolultabb. Hogy a feladatot jobban átlássam saját magam is, nem csak papíron olvasva, 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 algoritmus csak a legrövidebb útvonalat biztosítja. Nem számol azzal, hogy az adott útvonalon hányan kívánnak szintén végighaladni. Esetleg szembe fogalommal is dolgoznia kellene a programnak, továbbá ezt milyen szinten kell figyelembe venni.

Fontos, hogy inkonzisztens állapotba ne lépjen a szimuláció sosem. Ne legyen egyik ember a másikban, és nem mozoghat át falakon. Ez elengedhetetlen ahhoz, a valóságot minél jobban megközelítse a megoldásom. Ezért minden pillanatban bármi is történik, például torlódás alakul ki ajtóknál, nem szabad, hogy az emberek egymásba csússzanak.

## Első program megvalósítása

Az első programomat ennek a megoldására alkottam meg. Korlátozásokkal alkottam meg a szabályrendszerét, hogy fókuszálni tudjak a járókelők egymás kerülgetésére. Az embereknek kiterjedésük volt ebben a megoldásomban már, viszont mindenki csak egységnyi széles volt. A helyszínt lehetett tetszőlegesen paraméterezni, statikus és mozgó falakat is le lehetett helyezni benne. Az embereknek csak egyszerű tehendőjük volt, el kellett jutniuk A pontból B pontba. Viszont a helyszín leegyszerűsített volt annak okán, 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 az kifejlesztett algoritmusok olyan útvonalakat adtak vissza, amelyek kezdő és végpontok listájából álltak, amelyekhez egy időpillanatot is társítottak. 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 legyünk. 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 csoport, 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, hogy dolgozhasson. Tudhat-e mindenről, láthat-e a jövőbe, vagy csak a közvetlen környezetéből megszerezhető információval dolgozhasson. Ebben a megoldásomban nem csak a közvetlen környezetéből levonható információkból engedem dolgozni az algoritmusoknak, és törekszem arra, hogy minden információt megadjak az útvonalkereséshez.

Minden információ tudatában csak egyszer kell az útvonalat kiszámítani, és onnantól el van rendeltetve a kimenetele az eseményeknek. Ehhez szükséges az, hogy valóban tudjon mindenről minden résztvevő, és a jövőről is. Ennek okán 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.

### Első program megvalósítása

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 ellen.

Az embereknek aktuális pozíciójuk van, de fontosabb, hogy van útpont láncolatuk is. A járókelők létrehozásánál kiszámítom az útvonalukat, és a kezdeti pozícióból kiindulva ezen láncolaton halad 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 optimálisabbra dolgozhattam ki. Ekkor a négyzethálót a térnek pontos lenyomataként használva felhasználható az, hogy egy élszomszédos négyzetre történő átjutás egy embernek egy időegységbe telik, és csak egy közös csúccsal rendelkező mezőre ez az érték időegység, az átlós mozgás okán.

Az időegység minél kisebbnek való megválasztásával a szimuláció részletességét 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 időegységnyit, viszont tovább is állt már. Ezzel való számolást felesleges komplexitásnak véltem, mert megfelelő időegység megválasztásával ez mind kiküszöbölhető. Három megoldást fontoltam meg ebben a helyzetben.

### 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 pontjára lehet eljutni. Az átlós és a szomszédos lépés arányának függvényében más-más alakzatot kapunk bejárható térként. Ha ez az arány 1 és 2 között található, akkor egy nyolcszöget kapunk eredményül. Ezen nyolcszög csúcsai a következők. A négy főégtájban száz időegységnyit halad élszomszédos mezőkön egyenesen, továbbá a fő mellékégtájak irányában száz időegységnyit halad átlósan szintén egyenesen, ekkor 8 mezőre jut el a kezdeti mezőről az ember, és ezen nyolc mező mint csúcspontok alkotják a nyolcszöget. Az arányából kiszámítható, hogy ezen mezőket, mint csúcsokat összekötő szakaszoknak mekkora meredekségük. Ez a nyolcszög alak azzal magyarázható, hogy a nyolc irányon kívül nem lehet más pontját a körnek elérni egyenes vonalú mozgással. Törtvonalat leíró mozgással az ember több utat tesz meg, mint légtávolságban nézve az út végére, ezzel a kör alakja torzul és egy nyolcszöggé alakul.

Ezen okból kifolyólag tetszőleges pontossággal megközelíthettem a –t, mint az élszomszédos és sarokszomszédos mezőkre történő áthaladáshoz szükséges idő arányát, viszont nem értem volna el vele különösebb javulást, mindig csak nyolcszöget kapnék bejárható területnek adott időn belül. Így 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 számítási igényű lenne a program, de szemmel játható 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 át kívánnák haladni a cél érdekében. Ezért egy kisebb, 10x10-es négyzethálón keresztbe áthaladni, még ha csak kis forgalom is van már, akkor is 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, vagy egy statikus falra a szimuláció végéig. Ekkor a többi embertől vagy faltól tudnánk meg azt, hogy ők mikor terveznek tovább haladni. 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 lehetőségéig 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.

Ez nagyságrendekkel kevesebb számításigényt követel már meg magának, de még ezt is gyorsítani kívántam mert egy ügyeskedő megoldással kívántam felváltani az emberek kérdezgetését a tovább haladással kapcsolatban. Ekkor véglegesítettem, hogy a –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 teszi, és négyzetté alakítaná át, amit eredetileg körnek szántam, így elvetettem ezen egyszerűbb lehetőségeket. Az, hogy egy 10x10-es négyzethálón az átlós áthaladás időpillanat igényét 1.4 millióról, 70-en keresztül 30-ra csökkentettem, utat engedett annak a megoldásnak, hogy minden időpillanatot leképezhessek külön.

### 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 időegységnyit 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 is foglalkozom az emberekkel, így már csak a megjelenítésnél használtam fel újra az útvonalat, amelynél segítség volt ezen számítást nem megejteni.

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 keresztbe 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/hamis) típusú érték, ami 27 kB (kilóbájt) memóriát foglal le, és ez csak a nyers adat. Sajnos a pálya méretével, és a szimuláció hosszával a memóriaszükséglet 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 vettem 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 vettem, így maximum 100 egység széles vagy magas pályákkal dolgoztam legfeljebb. Ezen pályákat már önmagukban nem szimuláltam, mert temérdeknyi embert igényelt volna ahhoz, hogy komolyabb kerülőutakat kelljen kiszámolnia az útvonalkereső algoritmusnak. Ezért a négyzetalak megtörése érdekében falakat helyeztem le a játéktérbe.

### Falak a játéktérben

A falak olyanok, mint az emberek, tudnak akár mozogni is, például ha egy forgóajtóról van szó. Nem ütközhetnek 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 eszközölnek, ugyanis az élszomszédos mezők feléjük irányuló két átlós mozgását is letiltják külön, minden időpillanatra. Ezt lehet úgy tekinteni, mintha egy lenyomatot hagyna a fal a játéktérnek a pillanatképein. A dinamikus lekerekített falaknak, és az embereknek ezen lenyomatai viszont nem triviálisak. Ezen tiltó lenyomatai összefüggenek az emberek szélességével. Ha túl hatalmasak lennének, akkor nem lehetne átlósan ütközés nélkül egymás mellett elhaladni. Akkorára választottam a szélességüket, hogy az átlós elhaladás lehetséges legyen, és annál valamivel kisebbre.

### Ütközések elkerülésének megvalósítása

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

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

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

[ábra]

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.

### Útvonalkereső algoritmus megalkotás

Az útvonal kereső 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 mindig 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 kapjuk 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 fáradság 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.

### Megmaradt problémák orvoslása

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

### Első program eredménye

A kezdeti kikötések mind abban segítettek, hogy koncentráltan az ütközések elkerülésével tudjak 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 nagyságrenddel 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 megoldása komplex probléma globálisan, előre kiszámolni, ezért a következő programban időpillanatról időpillanatra, lokálisan tervezem megoldani a problémakört.

## Második program

Ezen programomat három jól elkülöníthető részre lehet osztani. A navigációs háló csúcsainak, azaz a teret határoló síkidomok létrehozásáért felelős az első rész, amely az alaprajzot figyelembevéve képes megalkotni ezen határló síkidomokat. A második része az, ami a határoló síkidomok által alkotott teret háromszögekre bontja. A harmadik rész az, ami keretet ad ezen háromszöghálónak, helyiségeket köt össze és tárol el róluk különböző adatokat. Továbbá van a fő feladatot megvalósító rész, amely a járókelőket, és az ők mozgatásáért felelős. Ez az utolsó kettő szorosabban összetartozik, és komplexitásában megegyezik az első két résszel.

### Navigációs hálót határoló síkidomok létrehozása

A határoló síkidomok azon sokszögek lesznek a helyín síkjára vetítve, amelyeken a járókelők középpontja tartózkodhat. Ennek okán a járókelők szélességével ez szorosan összefügg. Ugyanis az alaprajz csak a falakat tartalmazza, a járókelők szélességével nem számol. Ezen létrehozott határolósíkidomok a különböző szélességű járókelőkre más és más rendre. A létrehozott síkidomokat a falakat leíró síkidomoktól pontosan egy megadott távolságra lévő pontok halmaza képzi. Egy kör alakú fal, mint például egy oszlop esetében az létrehozott síkidom egy nagyobb sugarú kör lenne. Egy téglalap alakú fal esetében egy téglalapot ad vissza, viszont szélesebet, és hosszabbat, de minden toldását a sarkoknál lekerekíti. 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 iránya folyamatnak, és kézzel kellett volna megalkotnom különböző szélességű járókelőkre minden egyes ilyen csúcspont halmazt.

Második megoldásom során ezt automatizáltam. 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 második részének a programnak, ahol ezen síkidomok közötti rész háromszögekre feldarabolja.

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. A másik szabály az az, hogy kiszögelléseknél a síkidomnak körívet kellene tartalmaznia. Ezt én három csúccsal cseréltem fel, a csúcsszám 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. [ábra]

Ezen szabályok alapján egy emelet alaprajzát tökéletesen feldolgozza a program. Nagyon széles emberek esetében lehetne ezzel csak probléma. Ugyanis akkor ezen rész fel kéne készítenem olyan esetekre is, amikor egy szoba elérhetetlen az illető számára, mert az ajtón nem fér be. Ekkor a létrehozott síkidomok metszenék egymást, és megfelelő vágást kéne eszközölnöm. Erre nem tértem ki a munkám során, mert a 30 és 50 cm széles emberek esetében erre nem volt szükségem. Egy emelt 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 szögellnek ki a falból. Ekkor a kiszögelléseknél lévő első vagy az utolsó csúcs lehet, hogy közelebb kerül a falhoz, mint az ember távolsága. Ezt a problémát ennek külön ellenőrzésével oldottam meg, azaz nem csak a két vizsgált oldal távolságát vettem alapul, hanem az azt megelőzőt és az azt követőt is. Ezzel a vizsgálattal megoldottam minden felmerült problémát az egyetem I épület 4. emeletének lemodellezésénél.

### A bejárható tér háromszögekre bontása

Az irodalom kutatás során kifejtettem, hogy miként lehet ezt a problémát megoldani könnyen. Azt abban a részben nem fejtettem ki, hogy nem annyira könnyű mint a leírtak. A „fülező” módszer - amit leírtam, de itt röviden újra leírom - csak leírva egyszerű elgondolás. a módszer során két szomszédos oldalt kívánok mindig 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 hozzáadjuk a határoló síkidomokhoz a két kiválasztott szakasz helyére illesztve. Így minden behúzott szakasszal kettőt kitörlünk, azaz lépésenként egyre kevesebb él lesz, így véges számú kezdőél esetén véges futás idejű az algoritmus. Ez az algoritmus akkor nem jó, ha a határolósíkidomokból több is van, ekkor ugyanis össze kell őket olvasztani eggyé. Ha nem tenné meg az algoritmus az elején ezt a lépést, akkor nem találna akár egyetlen egy megfelelő behúzható szakaszt. Példa erre egy négyzet alakú ajtó nélküli szoba, amelyből egy négyzet ki van vágva. ekkor semelyik két szomszédos élből nem alkotható olyan háromszög, ami azt a teret fedi le, és csak is azt, amit kívánnánk lefedni. [ábra]

Ha az összeolvasztása meg is történt a síkidomoknak, akkor is nem elegendő minden szomszédos szakaszt háromszögelni. 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. Az összeolvasztásnak a megoldása, és az egész helyett egy másik megoldást hoztam létre.

Itt először a határolósíkidomok oldalain iteráltam végig. Mindegyik oldalhoz kerestem valamelyik másik síkidom 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 alkotott egy háromszöget. Ezt követően azon háromszögeket hoztam létre amit csak egy síkidom csúcsaival lehet létrehozni. C alakú síkidomok ilyenek tudnak lenni. [ábra]

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 mindig kétszer szerepelnek az adatok között. Mert minden háromszög oldala egy másik háromszögével közös, vagy egy határolósíkidom oldal. Mind a kettő esetben kétszer kell szerepelnie az összes szakasznak. Amely nem szerepel kétszer, az biztos, hogy lefedetlen területeket határol.

Sajnos a számítások valamennyire pontatlanok, olyan nagyságrendű számítási pontatlansággal kellett néha dolgoznom. Ezt epszilonnak megválasztottam, és ez volt az a maximális eltérés, amikor még két értéket egyenlőnek véltem. A pontatlanság komoly problémát okozott, főképp közeli, és közel párhuzamos szakaszok esetében. Ekkor ugyanis tévesen metszőnek vélte a szakaszokat néha. Sajnos ezt kiküszöbölni nem tudtam, ezért plusz 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 rajtuk kívül. Ekkor minden a síkot – egy módon – lefedő háromszög szakaszát ismerem. Ezen szakaszok halmaza elégséges ahhoz, hogy létrehozzam a háromszögeket, vagy csak az útvonalkereséshez szükséges navigációs hálót. Ahhoz külön el kell tárolni a határoló síkidomok oldalait is, ami a kiindulási adata volt ezen résznek, így azzal nincs külön probléma.

### Az előző két rész szobákra bontása és 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 csak egy szám, hanem a szobában a diákoknak és az oktatóknak van külön helyük is. Továbbá a helyiségnek van egy tulajdonsága, ami azt mondja meg, hogy átjárásra használható-e. A szobákban a helyeknek vagy egy relatív pozíciójuk és az, hogy foglalt-e. (és egy értékük is van, hogy melyik helyet töltsék fel a diákok első sorban)

Ezt követően a szobákat össze kell kötni. A szobák kijáratait egymáshoz rendeli az emelet. Az emelet a szobákat a helyükre illeszti, ehhez alkalmasan elforgatja és eltolja őket, majd a szobák közti kijáratokat összeköti. Ezen összeköttetések falak, ajtófélfák, és a köztes területet két háromszöggel lefedi, mint bejárható tér. Ezen háromszögeket hozzáadja a navigációs hálóhoz.

### Nyilatkozat

a

### Tartalmi összefoglaló

a

### Bevezetés

a

### Irodalomkutatás, technológiák, hasonló alkotások bemutatása

a

### A feladatkiírás pontosítása és részletes elemzése

a

### Önálló munka bemutatása

a

### Önálló munka értékelése, mérések, eredmények bemutatása

a

### Összefoglaló

a

### Köszönetnyilvánítások

a

### Részletes és pontos irodalomjegyzék

a

### Ábrajegyzék, táblázatjegyzék

a

### Függelék

a

## Egyéb tartalmi elemek

### Stílus

a

### E/1

a

### Rövidítések

a

### Technológia megválasztása

# Formázási tudnivalók

Ez a fejezet összeszedi azokat a nélkülözhetetlen elemeket, amelyeket a dolgozat készítése során használnod kell a Word eszköztárából. Feltétlenül olvasd egyszer végig, hátha akad köztük újdonság.

## Általános tudnivalók

A diplomaterv szabványos méretű A4-es lapokra kerüljön. Az oldalak tükörmargóval készüljenek (mindenhol 2.5cm, baloldalon 1cm-es kötéssel). Az alapértelmezett betűkészlet a 12 pontos Times New Roman, másfeles sorközzel.

Minden oldalon - az első négy szerkezeti elem kivételével - szerepelnie kell az oldalszámnak.

A fejezeteket decimális beosztással kell ellátni. Az ábrákat a megfelelő helyre be kell illeszteni, fejezetenként decimális számmal és kifejező címmel kell ellátni.

Az irodalomjegyzék szövegközi hivatkozása történhet a Harvard-rendszerben (a szerző és az évszám megadásával) vagy sorszámozva. A teljes lista névsor szerinti sorrendben a szöveg végén szerepeljen (sorszámozott irodalmi hivatkozások esetén hivatkozási sorrendben). A szakirodalmi források címeit azonban mindig az eredeti nyelven kell megadni, esetleg zárójelben a fordítással. A listában szereplő valamennyi publikációra hivatkozni kell a szövegben.

Ez a sablon úgy van kialakítva, hogy a fenti előírásokat eleve kikényszerítse, így általában külön nem kell foglalkoznod velük.

## Stílusok

A Word szövegek egységességét stílusok segítéségével lehet a legegyszerűbben garantálni. Azt javaslom, rögzítsd ki a stílus eszköztárt jobb oldalra, így folyamatosan nyomon követheted, hogy milyen stílussal dolgozol, illetve a stílusok közti váltást is sokszor kényelmesebb itt kezelni, mint a fenti eszköztáron.



. ábra: Stílus eszköztár rögzítése

## Címsorok

A fejezetcímek esetén a **Címsor 1-4** (Heading 1-4) stílusokat használjuk. Címsor 4-nél mélyebb fejezetszintre egy ilyen terjedelmű munkában ritkán van szükség, ha ez mégis felmerülne, sokszor inkább a fejezetszerkezetet érdemes átgondolni újból

Tipp: A fejezetcímek aktuális szintjét gyorsan módosíthatod az alt+shift+[bal/jobb] billentyűkombinációkkal.

## Másolás, beillesztés

A copy-paste a szép formázás legnagyobb gyilkosa. Amennyiben a dolgozatodon belül helyezel át részleteket, és a stílusokat szépen következetesen használtad, nem nagyon ronthatsz el semmit. Ha viszont külső forrásból másolsz be szöveget, érdemes minden esetben a beillesztést formázások nélkül végezni. Ezt az opciót a Word mindig felkínálja, a beillesztést követően.



. ábra: Beillesztés formázás nélkül

Amennyiben olyan szövegrészt mozgatsz kivágás/beillesztés párossal, melyben automatikus sorszámozás volt (irodalomhivatkozás, ábra, táblázat), akkor figyelj rá, hogy a sorszámok nem frissülnek automatikusan.

## Mezőfrissítés

A dolgozatodban számos automatikusan kalkulált mező szerepel (ábra sorszámok, irodalomhivatkozások, tartalomjegyzék stb. Ezeket a Word nem tartja automatikusan karban. A mezők értékét érdemes rendszeresen (de a beadás/nyomtatás/pdf generálás előtt feltétlenül) frissíteni. Ehhez jelöld ki a teljes dokumentumot (ctrl+A), majd az előugró menüből válaszd a „mezőfrissítés” opciót.

## Helyesírás

A rossz helyesírásra nincs mentség ☹. E fejezetben összeszedem a leggyakrabban látott hibákat, amiknek elkerülésére érdemes odafigyelni. Ettől függetlenül melegen ajánlom, hogy a kész dolgozatod olvastasd át egy barátoddal/családtagoddal, hogy az apróbb, megbúvó hibákat is kiszűrd.

### Elgépelések

Ez mindenkivel megesik. Egy jó helyesírás ellenőrző az elgépelések nagy részét megfogja.

### Egyeztetés hiánya

Az elírások egyik leggyakoribb formája az egyes szám/többes szám egyeztetésének hiánya mondatrészek között, mint például itt: „Petike és a barátnője elment**ek** a boltba és hoz**ott** egy kiló kenyeret”. Ezek a mondatok főleg az utólagos átfogalmazások, belejavítások során keletkeznek, legjobb védelem ellenük az utólagos átolvasás.

### Külföldi szavak, kifejezések

Az idegen szavakkal csak a baj van, nehéz őket leírni, ragozni, kiolvasni, vagyis úgy általában használni. Az általános jó tanácsom, hogy amennyiben csak lehetséges, akkor **magyar, vagy magyarosított írásmódú szakkifejezéseket használj**, és a könnyebb olvashatóság érdekében mindig **kerüld az idegen szavak ragozását**.

Ez sokszor nem intuitív szabály, hiszen a beszélt informatikai szlengben előszeretettel használunk olyan fordulatokat, mint „property-vel”, „compile-olom” stb., ezek helyett a „tulajdonsággal”, illetve a „lefordítom” százszor jobban néznek ki papíron, könnyítik az olvasást és csökkentik a hibázás esélyét. Néha (például a tulajdonneveknél) nehezen elkerülhető a ragozás, de kis leleményességgel, ügyes fogalmazással itt is megoldható a probléma. Pl. „Apache-csal” helyett írjuk azt, hogy „Apache webszerverrel”.

Néhány gyorstipp a szoftverfejlesztőknek: property🡺tulajdonság, event🡺esemény, method 🡺metódus/függvény, debug🡺hibakeresés, file🡺fájl.

Ha mégis belevágsz az angol kifejezések használatába, lelked rajta, de kérlek, legalább ezt a minimális szabályhalmazt olvasd át előtte [3].

### Stb

Az „stb.” rövidítés azt jelenti „és a többi”, mivel „és” előtt felsorolásoknál nem teszünk veszőt, az „stb.” elé sem teszünk soha. Pl.: asztal, szék, lámpa stb.

### Helyesírás ellenőrző

Személyes ízlés kérdése, hogy milyen eszközzel (Latex/Word), illetve ezen belül milyen nyelvű változattal dolgozol, ugyanakkor azt **meg kell oldanod, hogy legyen mellé magyar nyelvű helyesírás ellenőrződ**. E nélkül dokumentumot szerkeszteni olyan, mint papíron programozni….

Figyelj rá, hogy a Word megengedi, hogy többféle nyelv is legyen egy dokumentumban, illetve a kívülről beillesztett szövegek nyelvét néha a forrásoldal/forrásdokumentum határozza meg. Érdemes ezekben az esetekben kikényszeríteni, hogy a teljes szakasz/dokumentum nyelve magyar legyen. Érdemes néha tesztelni a helyesírás ellenőrzőt. Ha pl. beírod, hogy  és nem húzza alá pirossal, akkor valószínűleg valami nem stimmel a beállításaiddal.



. ábra: ellenőrzés nyelvének a megadása

## Képek

A dolgozatodban valószínűleg számos ábrára lesz szükséged, ezek használatánál azonban érdemes pár dologra odafigyelned.

### Beszúrás, formázás

A képhez használd a **Kép** stílust.

Képaláírást a képen jobb gombbal kattintva a Képaláírás beszúrása… opcióval adhatod hozzá, így az automatikusan **Képaláírás** (Caption) stílusú lesz és kezeli a sorszámozást is.



. ábra: Példa képaláírásra

Az ábra sorszáma mellé mindig érdemes rövid magyarázatot is fűzni. Érdemes tudni, hogy Wordben számos beállítási móddal lehet képet beszúrni. Tapasztalni fogod, hogy számodra a legpraktikusabb a „szöveggel egy sorba” elrendezés lesz, mivel így szépen együtt marad az ábra a szöveggel. Oldaltörésekkor az ábrák méretének módosításával tudod megoldani, hogy ne maradjanak nagy üres felületek a dolgozatodban.

### Képminőség

A raszteres (tehát nem vektorgrafikus) képek használata különös körültekintést igényel. Ezek kiválóan néznek ki a 90 dpi-s monitorodon, ám, a 600/1200 dpi-s nyomtatókon kinyomtatva rendkívül bénák lesznek a szép, pixelmentes szövegek és vektorgrafikus ábrák mellett. Hogy lásd, mire számíthatsz, nézzük meg az előző ábrát 5-szörös nagyításban.



. ábra: a raszteres képek nyomtatásban csúúúúnyák lesznek

Ha tehát lehetséges, használjunk vektorgrafikus ábrákat, vagyis a diagramokat, forráskódot stb. ne képenyőképeken keresztül, hanem közvetlen copy-paste megoldással másoljuk át a dokumentumunkba.

Ha elkerülhetetlen a raszteres képek használata, akkor próbáljunk meg minél magasabb felbontású képet berakni. Bár a képernyőn nem feltétlenül látszik majd a különbség, de a nyomtató ki tudja majd használni a magasabb felbontású képet. A 6. ábra például igen magas felbontású, amit láthatsz is, ha felnagyítod.



7. ábra: ez az ábra 460\*750 pixel felbontású

Figyelj rá, hogy a PDF-be mentéskor a Word (2013) alapértelmezésben jelentősen lerontja a képek minőségét, sőt, a vektorgrafikus ábrákból is rasztereset csinál. Ha a nyomtatást PDF alapján akarod csinálni, érdemes külső szoftvert használni a Word🡺Pdf konverzióra, vagy alaposan eljátszani a Word beállításaival.

## Kereszthivatkozások

Amennyiben szeretnél egy ábrára, vagy korábbi fejezetre hivatkozni, használd a Word „Hivatkozás/Kereszthivatkozás” menüpontját. Kerüld az így beszúrt szövegek közvetlen ragozását, hiszen az könnyen változhat, amennyiben a forrás változik. Fejezetekre, ábrára sose a teljes szövegükkel, csupán a sorszámukkal („Csak címke és szám”) hivatkozz. Ezt szemlélteti a 7. ábra.



8. ábra: ábrahivatkozás beszúrása

Tipp: érdemes elkerülni az „előző oldalon”, „következő oldalon” fordulatokat, ugyanis az ábrák végső helyzete a tördelés során még megváltozhat.

## Irodalomhivatkozások

Az irodalomhivatkozások kezelésére a Word egy kényelmes és jól használható funkciót kínál. Amikor dolgozatodban egy külső műre, weboldalra, könyvre, előadásra stb. szeretnél hivatkozni, használd a „hivatkozás/források kezelése” menüpontot. Itt felveheted az egyes műveket, illetve a hozzájuk kapcsolódó szerzői adatokat. Folyóirat cikkeknél a szerzők mellett szerepeljen a pontos cím, a folyóirat címe, évfolyam, szám, oldalszám tól-ig. A folyóirat címeket csak akkor rövidítsük, ha azok nagyon közismertek vagy nagyon hosszúak. Internet hivatkozások megadásakor fontos, hogy az elérési út előtt megadjuk az oldal szerzőjét és címét (mivel a link egy idő után akár elérhetetlenné is válhat), valamint az elérés időpontját.

Az egyes forrásmunkákra a dolgozat szövegében lehet hivatkozni a „hivatkozás/idézet beszúrása” menüponttal. Figyelj rá, hogy az IEEE stílust használd.



. ábra: idézetek és irodalomjegyzék

### Pozícionálás

Az irodalomhivatkozások a szövegtörzsben, ábrák szövegében és táblázatokban is előfordulhatnak, de **fejezetcímekben soha**. Amennyiben egy adott forrás egy egész bekezdésre/fejezetre vonatkozik, akkor is elég, ha az első mondat/első bekezdés után megemlítjük. Szó szerint idézett szöveg esetén az idézet végén, külső forrásból átvett ábra esetén a képaláírásban hivatkozzunk a forrásra.

### Mikor kell hivatkoznom?

Minden külső forrásból átvett képnél, szövegrésznél, olyan szövegeknél, melyek megfogalmazásához külső forrásokat használtál, illetve olyan állítások/technológiák/algoritmusok megemlítésénél, melyek nem feltétlenül egyértelműek egy átlagos műveltségű olvasó számára. Nagyságrendileg egy szakdolgozatban átlagosan 10-20, egy diplomatervben átlagosan 20-30 külső forrást illik megemlíteni. A források megválasztásánál, ha lehet, törekedjél a nyomtatott források keresésére, ezek lehetnek folyóiratcikkek, könyvek, könyvfejezetek, útmutatók, egyetemi jegyzetek stb., s csak akkor hivatkozz weboldalakra, vagy Wikipediára, ha ilyent nem találtál.

## Word tippek és trükkök

### Navigációs ablak

Mivel a fejezeteket nagy valószínűséggel nem sorrendben fogod tartalommal feltölteni, érdemes bekapcsolni a navigációs ablakot (nézet/navigációs ablak). Ez gyors és egyszerű keresési és ugrálási lehetőséget biztosít a fejezetek közt



. ábra: Navigációs ablak

### Megjegyzések

A véleményezés/megjegyzés menüponttal megjegyzéseket fűzhetsz a szöveg tetszőleges részéhez. Ezt konzulensed előszeretettel fogja majd használni az észrevételei megfogalmazásához, de neked is jól jöhet, hogy megjelöld azokat a részeket, ahova még vissza kell térned. Fontos, hogy nyomtatás előtt az összes megjegyzést töröld, vagy külön állítsd be, hogy a megjegyzések ne jelenjenek meg a nyomtatásban.

### Korrektúra

Konzulensed, vagy mások, akiket a dolgozat átolvasására megkérsz legegyszerűbben a korrektúra funkció használatával tudják úgy javítani a dolgozatod, hogy a javításokat te is lásd. Az átolvasáshoz tehát érdemes bekapcsolni a „változtatások követése” funkciót és „Minden korrektúra” mutatására beállítani a rendszert.

Amikor korrektúrázott változatot kapsz vissza valakitől, használd a „Véleményezés ablak”-ot, így egyetlen javítás sem kerülheti el a figyelmedet, s így mindegyiket elfogadhatod, vagy elvetheted, igény szerint.



. ábra: ellenőrizd, hogy minden bekarikázott funkciót ismersz-e!

### Gyorsbillentyűk

A Word lehetőséget ad tetszőleges funkció gyorsbillentyűre kötésére. Használd a „billentyűparancsok testreszabása” funkciót, hogy gyorsbillentyűt rendelj a leggyakrabban használt gombokhoz.

Érdemes továbbá gyorsbillentyűt rendelni az alapvető stílusokhoz is. Bemelegítésként hozzárendeltük az alt+K kombinációt a „Kép” stílushoz.

## Kódrészletek

Érdekesebb és bonyolultabb programozási megoldásainkat bátran illusztrálhatjuk kódrészletek beszúrásával. Fontos, hogy a beillesztett kódrészlet mérete álljon arányban annak a fontosságával, tehát ritkán érdemes egy „bekezdésnyi” kódnál többet beszúrni egyszerre. Amennyiben egy bonyolultabb (akár több oldalas) algoritmust szeretnénk bemutatni, annak a kódját érdemesebb függelékbe rakni.

### Formázás

A kódrészletek formázásánál kerüljük a helypazarlást, illetve próbáljuk megelőzni az olvashatóságot rontó sortördelést, akár a forráskód módosításának árán is. Két praktikus tipp Visual Studio-hoz:

Másolás előtt érdemes a behúzások mértékét 4-ről 2 karakterre csökkenteni, majd a másolandó kódrészletet újraformázni (ctrl+K, ctrl +D), alternatívaként persze utólag is kitörölgethetjük a felesleges szóközöket a kódból.



. ábra: a behúzások mértékének csökkentése Visual Studioban

A másolandó kódrészlet kijelölésekor tartsuk lenyomva az ALT-ot. Így a valóban releváns részekre korlátozhatjuk a kijelölést.



. ábra: kódblokk kijelölése ALT+egérkurzor segítségével

Kódrészletek beillesztésére semmiképpen ne képeket használjunk, hanem a közvetlenül beszúrt kódrészletekkel dolgozzunk. A Visual Studio-ból kimásolt kódok alapvetően jól néznek ki, további szépítésre használjuk a **Kód** stílust.

static void Main(string[] args)

{

var ci = new CultureInfo("en-us");

ci.NumberFormat.CurrencySymbol = "";

Thread.CurrentThread.CurrentCulture = ci;

Console.WriteLine(ci);

Console.WriteLine("{0:c}", 5.66);

}

### Irodalomjegyzék

Az Irodalomjegyzékben szereplő hivatkozásokat **Irodalomjegyzék sor** stílussal formázzuk, a címüket pedig **Irodalomjegyzék forrás** stílussal emeljük ki.

A szövegbe a hivatkozásokat a Kereszthivatkozás beszúrása (Insert cross-reference) funkcióval helyezzük el (példa egy így beszúrt hivatkozásra: **Hiba! A hivatkozási forrás nem található.**), így azok automatikusan frissülnek a hivatkozások átrendezésekor.

## Utolsó simítások

Miután elkészültünk a dokumentációval, ne felejtsük el a következő lépéseket:

* Kereszthivatkozások frissítése: miután kijelöltük a teljes szöveget (Ctrl+A), nyomjuk meg az F9 billentyűt, és a Word frissíti az összes kereszthivatkozást. Ilyenkor ellenőrizzük, hogy nem jelent-e meg valahol a "Hiba! A könyvjelző nem létezik." szöveg.
* Dokumentum tulajdonságok megadása: a dokumentumhoz tartozó meta adatok kitöltése (szerző, cím, kulcsszavak stb.). Erre való a Dokumentum tulajdonságai panel, mely a Fájl / Információ / Tulajdonságok / Dokumentumpanel megjelenítése úton érhető el.
* Kinézet ellenőrzése PDF-ben: a legjobb teszt a végén, ha PDF-et készítünk a dokumentumból, és azt leellenőrizzük.

# Irodalomjegyzék

|  |  |
| --- | --- |
| [1] | 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]. |
| [2] | W3C, „HTML, The Web’s Core Language,” [Online]. Available: http://www.w3.org/html/. [Hozzáférés dátuma: 20 október 2015]. |
| [3] | K. Nahtkasztlija, „Az idegen szavak toldalékolása,” június 2009. [Online]. Available: http://www.pcguru.hu/blog/kredenc/az-idegen-szavak-toldalekolasa/5062. |

Függelék