KERESÉSI ALGORITMUSOK DINAMIKUS KÖRNYEZETBEN

BME-VIK MÉRÉSTECHNIKA ÉS INFORMÁCIÓS RENDSZEREK TANSZÉK
MESTERSÉGES INTELLIGENCIA (VIMIAC00)
SÁNDOR MÁTYÁS HÁZI FELADATA

A FELADAT ÁTTEKINTÉSE

A házi feladat célja különböző keresési algoritmusok működésének vizsgálata dinamikus környezetben. A vizsgálathoz először is egy megfelelő környezetet kellett teremteni, majd az algoritmusokat a környezethez igazítva, egy közös interfészt megvalósítva implementálni.

A környezet feladata, hogy a megvalósított algoritmusok teljesítményének mérésére eszközöket szolgáltasson, valamint hogy egy grafikus interfész segítségével biztosítsa az interaktív kommunikációt a felhasználóval.

A leírás első felében programot fogom bemutatni, a második felében pedig a megvalósítás részleteinek ismertetése után az elvégzett méréseket.

A PROGRAMRÓL

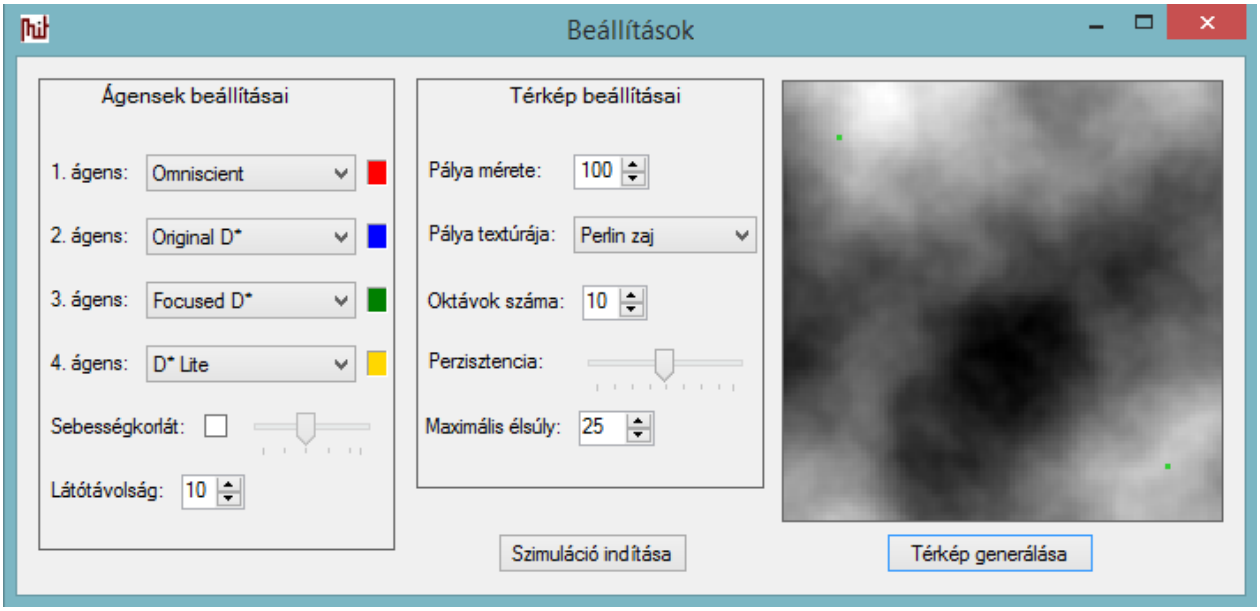
A programot C# nyelven írtam, a .NET 4.5-ös keretrendszert és a WinForms API-t felhasználva. Bár teljesítmény szempontjából nem ez volt a legelőnyösebb választás, a feladat céljának tökéletesen megfelel.

Az alkalmazás grafikus felülete két ablakból áll: egy főablakból és szimulációs ablakból. A főablakban az algoritmusokat és a szimuláció futtatásának környezetét tudjuk testre szabni különböző szempontok alapján, a szimulációs ablakban pedig az algoritmusok futásának vizuális nyomon követése mellett a pillanatnyi teljesítményadatokhoz is hozzáférhetünk.

Az alkalmazás futtatásához egy többmagos processzorral rendelkező számítógép javasolt, ugyanis minden ágens külön háttérszálat hoz létre magának. Így, ha minden létrehozott ágensre jut egy processzormag, az algoritmusok felhasznált processzorideje szinte megegyezik a tényleges futási idővel, ami nagyban megkönnyíti az intuitív értékelést.

A használt memória jellemzően 1-200 MB körül mozog egy közepes méretű térképnél, a .NET keretrendszer gondoskodik a tárhely hatékony kihasználásáról.

BEÁLLÍTÁSOK ABLAK



A *bellítások* ablak bal oldalon találjuk az ágensekre vonatkozó beállításokat: az alkalmazás egyidejűleg négy ágens futtatását teszi lehetővé, amelyeknek típusán kívül a színét is kiválaszthatjuk.

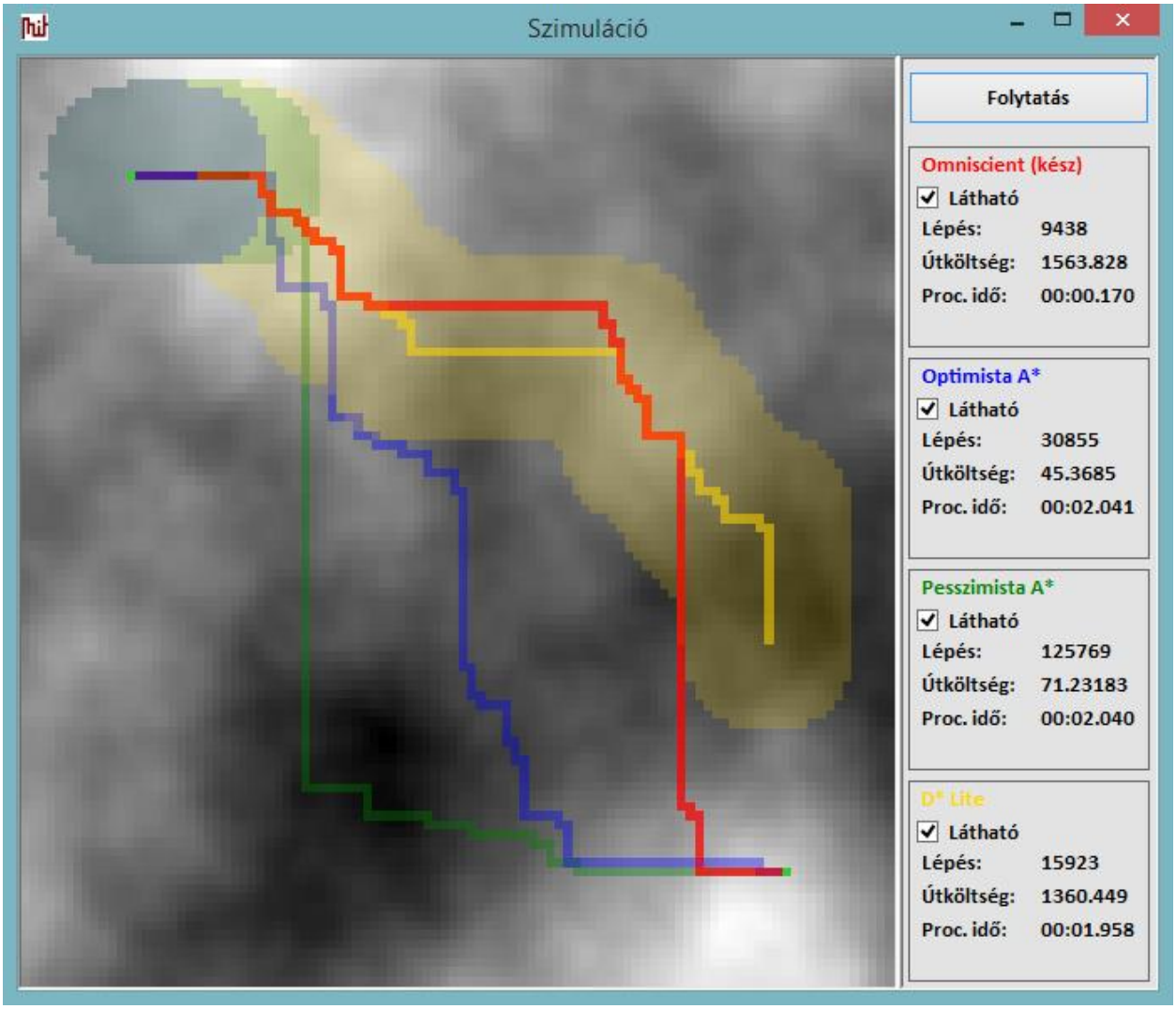
További lehetőségként az ágensek látótávolságát is módosíthatjuk és korlátozhatjuk a mozgási sebességüket is, ezzel érzékeltetve, hogy valós körülmények között előfordulhat, hogy az ágens lassabban mozog, mint ahogy a vezérlő algoritmus kiszámolja az aktuális tudásához mérten optimális útvonalat. A mozgási sebesség megfelelő kiválasztásával a két erőforrás kihasználtságát tudjuk kiegyenlíteni.

Az ablak közepén a térképre vonatkozó beállításokat láthatunk: ez lesz a környezet, amiben az ágenseket létrehozzuk. A pálya textúrája alapján két fő típust különböztethetünk meg: a folytonos és a bináris pályákat. Folytonos esetben (*Perlin zaj* és *Márványszerű*) a térkép pontjainak súlya 1 és a beállított maximális élsúly közötti értékeket vesz fel, a beállított paraméterekkel generált kiválasztott Perlin zaj-implementáció szerint.

Bináris esetben a hasonló nevű folytonos textúrák értékeit a két szélsőérték közül a közelebbire „kerekíti”, így a pálya a szürke árnyalatai helyett csak fekete és fehér régiókból áll. Fontos azonban, hogy a fekete mezők *nem* jelentenek áthatolhatatlan akadályt! A két típus megkülönböztetésének indokoltsága a későbbiekben igazolódik majd be, ugyanis a kereső algoritmusok különböző csoportjai nagyon eltérően viselkednek a két esetben.

A „térkép generálása” gombbal a megadott paraméterek alapján egy véletlenszerű pályát generálhatunk, ami a jobb oldalon látható előnézeti ablakban meg is jelenik (minél világosabb egy mező, annál kisebb az áthaladás költsége). A két zöld pont az ágensek kezdőpontja és célja, a bal felsőből a jobb alsóba fognak majd utat keresni. Az alkalmazás megjegyzi a legutoljára generált térképet, így lehetőség van ugyanazon a pályán több szimulációt is futtatni (ha esetleg a négy párhuzamos ágens nem lenne elegendő).

SZIMULÁCIÓ ABLAK



A *szimuláció* ablak nagyobb részét az ágensek állapotának vizuális reprezentációja tölti ki. Ágensenként legfeljebb három mezőhalmazt különböztethetünk meg:

1. Az ágens által bejárt utat a kezdőponttól (legsötétebb)
2. Az ágens által aktuálisan optimálisnak vélt utat a célig
3. Az ágens által ismert mezőket (legvilágosabb)

Bizonyos algoritmusok sajátosságaiból adódóan ezek a halmazok akár hiányozhatnak is, például a *D* Lite* algoritmus során nem kell az ágensnek előre kiszámolnia a célhoz vezető utat (hatékonysága éppen ebből fakad), az *Omniscient* ágens reprezentációja pedig egyszerűen a célhoz vezető *lehető legrövidebb* út.

Az ablak jobb oldalán a szünet gomb és az egyes ágensekhez tartozó panelek találhatók. A *Lépés* címke után szereplő érték a szűk keresztmetszetet jelentő *open* listában való minimumkeresés száma. A *Proc. idő* címke után az ágens útvonaltervezéssel töltött ideje látható. Ha a sebességkorlát opció engedélyezve van, a panel a processzorhasználati időn kívül a „valódi” keresési időt is kiírja.

A MEGVALÓSÍTÁS ÁTTEKINTÉSE

A *dinamikus környezet* a feladat címében arra utal, hogy az ágensek hozzáférése a környezethez hiányos, nem feltétlenül ismerik az összes csomópont költségét. Ez a hagyományos keresési algoritmusokhoz képest egy nagyságrendbeli komplexitásnövekedést jelent, ami azonban bizonyos speciális algoritmusok felhasználásával jelentősen lecsökkenthető.

A keresési algoritmusok teljesítményének szűk keresztmetszete az általam használt implementációk esetében az *open* lista minimumának megkeresése, ami egy lineáris minimumkeresés *O(n)* műveletben (*n* a csomópontok száma). Az *open* listát kupac (binary heap) adatszerkezettel megvalósítva akár *O(log n)*-re is le lehetne csökkenteni az útvonalkeresés idejét, ez azonban már nem fért bele az időmbe.

A dinamikus (D*) algoritmusok alapvető célja az, hogy e számításigényes művelet használatának gyakoriságát minimalizálják. Ehhez szinte mindig valamilyen segédstruktúrákat használnak, amikben az egyes csomópontokra vonatkozó információt a keresés során eltárolják, hogy a következő kereséseket felgyorsítsák. Közös tulajdonságuk, hogy minden algoritmus definiál egy *kulcs* struktúrát, ami az *open* lista rendezésének kulcsa.

Az ágensek csoportosításának fontos szempontja, hogy az általuk még fel nem fedezett csomópontokat hogyan kezelik. Ebből a szempontból két csoportra bontottam az algoritmusokat, *optimistákra* és *pesszimistákra*. Az *optmista* algoritmus az ismeretlen pontok élsúlyát 1-nek feltételezi, a pesszimista pedig a pálya maximális élsúlyának. A két csoportba tartozó algoritmusok bizonyos helyzetekben nagyon jellegzetes viselkedésmintákat mutatnak, ahogy majd a továbbiakban ki is fejtem. Elöljáróban annyit fűznék hozzá, hogy a bemutatott kísérletekben használt valószerű környezetben a *pesszimista* algoritmusok kiszámíthatóbb viselkedést és konzisztensebb teljesítményt mutattak, ezért a programban az A* kivételével egyik algoritmusnak sincsen *optimista* változata.

A mérések során a függő változók a processzoridő, az *O(n)* műveletek száma és a megtett út hossza, a független változók pedig az algoritmus típusa, valamint a pálya mérete és jellege közül kerülnek ki. A mérések során nem használt független változók értéke az alkalmazásban szereplő alapértelmezett értékkel egyenlő.

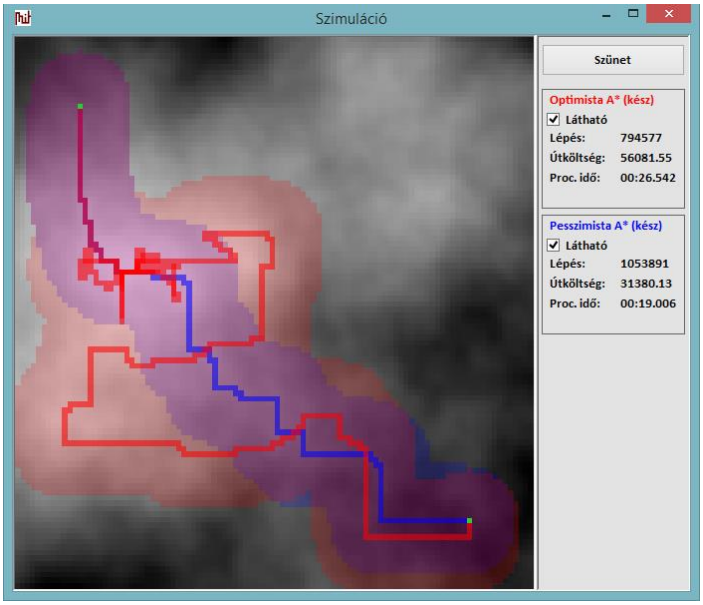
MÉRÉSEK, VIZSGÁLATOK

1. OPTIMIZMUS KÖRNYEZETTŐL FÜGGŐ HATÁSA AZ ALGORITMUS MŰKÖDÉSÉRE

A következő mérésekkel azt mutatom be, hogy hogyan befolyásolja az algoritmus működését a felfedezetlen csomópontokról alkotott hiedelme és hogy milyen speciális környezetekben melyik megvalósítás hatékonyabb. A mérésekhez A* algoritmusokat fogok használni, de a levont következtetés a dinamikus algoritmusokra is igaz lesz.

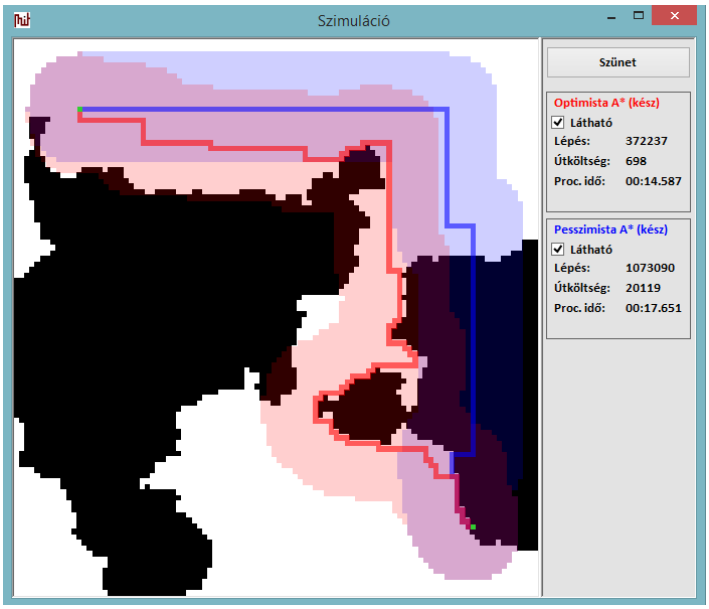
Az *optimista* algoritmusokra jellemző, hogy nehezen veszik rá magukat arra, hogy egy lehetségesen nem optimális, de ismert utat válasszanak az ismeretlen, de adott esetben nagy kerülővel járó út helyett. Így elképzelhető, hogy oda-vissza ingáznak két, közel optimális útvonal között csak azért, mert az egyiken haladva rájönnek, hogy az éppen felfedezett él súlya miatt a másik útvonal elméletileg jobb lenne, majd mire a másikon eljutnak egy új pont felfedezéséig, visszatérnek az első útvonalra.

Ez tipikusan akkor szokott előfordulni, ha a heurisztika (Manhattan-távolság) szerinti költség és a tényleges költség különbsége nagy és a cél felé vezető úton növekvő tendenciát mutat – vagyis minél magasabb a pálya maximális élsúlya, annál gyakrabban fordul elő. Ennek egy példáját láthatjuk itt (a piros az *optimista* ágens által bejárt út, az élsúlyok maximuma a heurisztika 500-szorosa):



Bár az elvégzett útvonaltervezési lépések tekintetében az *optimista* algoritmus egy kicsivel jobban teljesített, de processzoridő szempontjából a *pesszimista* lett a hatékonyabb. Bizonyos körülmények között fontos lehet még a megtett út költsége is, ebből a szempontból pedig az *optimista* algoritmus által nyújtott megoldás akár elfogadhatatlanul rossz is lehet.

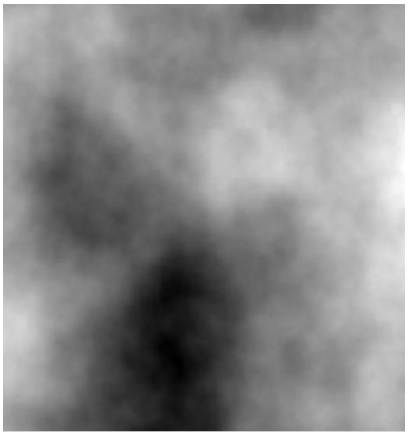
Ha azonban a pálya jellege olyan, hogy létezik egy olyan út a kezdő- és a végpont között, ahol az élsúlyokat jól közelíti a heurisztika és az út költsége közel minimális, az *optimista* algoritmus akár minden szempontból jobban teljesíthet. Erre egy példa itt látható:



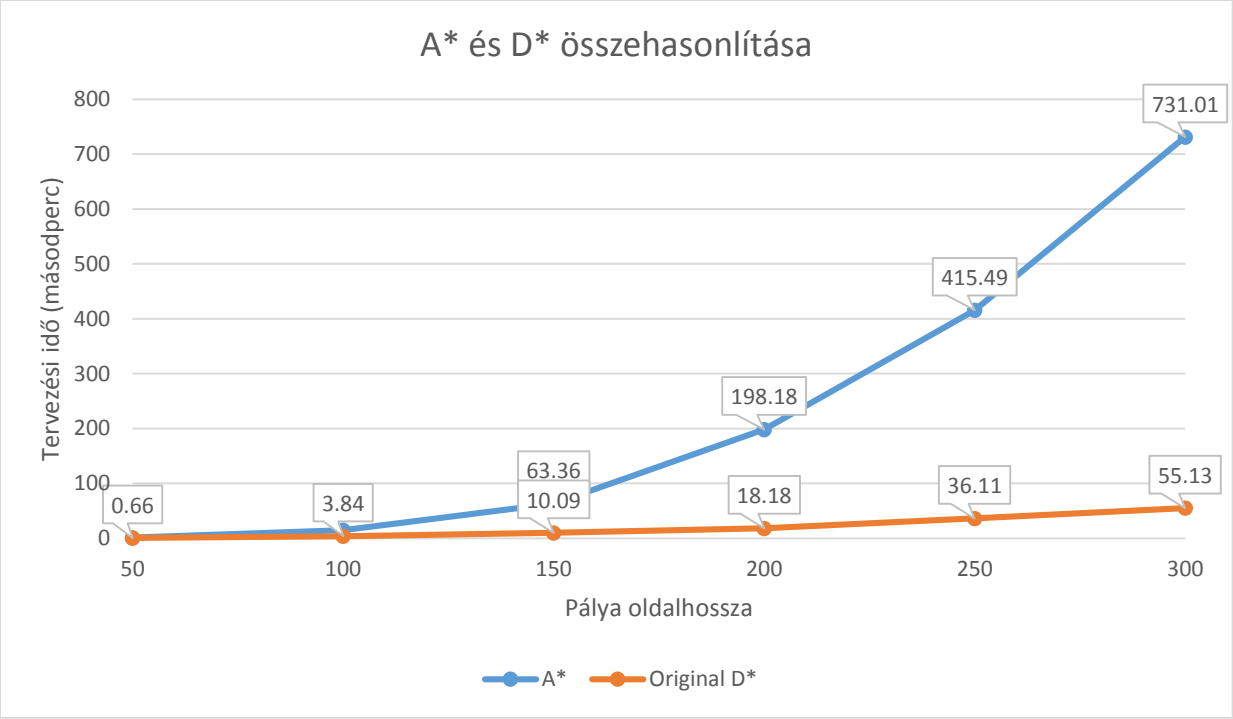
2. ALGORITMUSOK TELJESÍTMÉNYE A PÁLYA MÉRETÉNEK FÜGGVÉNYÉBEN

Ez a vizsgálat képzi a feladat gerincét: a következő mérések célja annak megmutatása, hogy mennyivel hatékonyabbak dinamikus környezetben a dinamikus algoritmusok a sima A*-nál és milyen sorrendet lehet felállítani az egyes D*-implementációk között. A mérések során a pálya méretét 50 és 500 között 50-es lépésekben növeltem és minden mérethez 3 különböző pályán hasonlítottam össze az algoritmusokat.

Érdemes továbbá megjegyezni, hogy a csomópontok száma már önmagában négyzetesen függ a pálya oldalhosszától, de az alábbi grafikonok ettől függetlenül megfelelően szemléltetik a nagyságrendbeli különbséget.

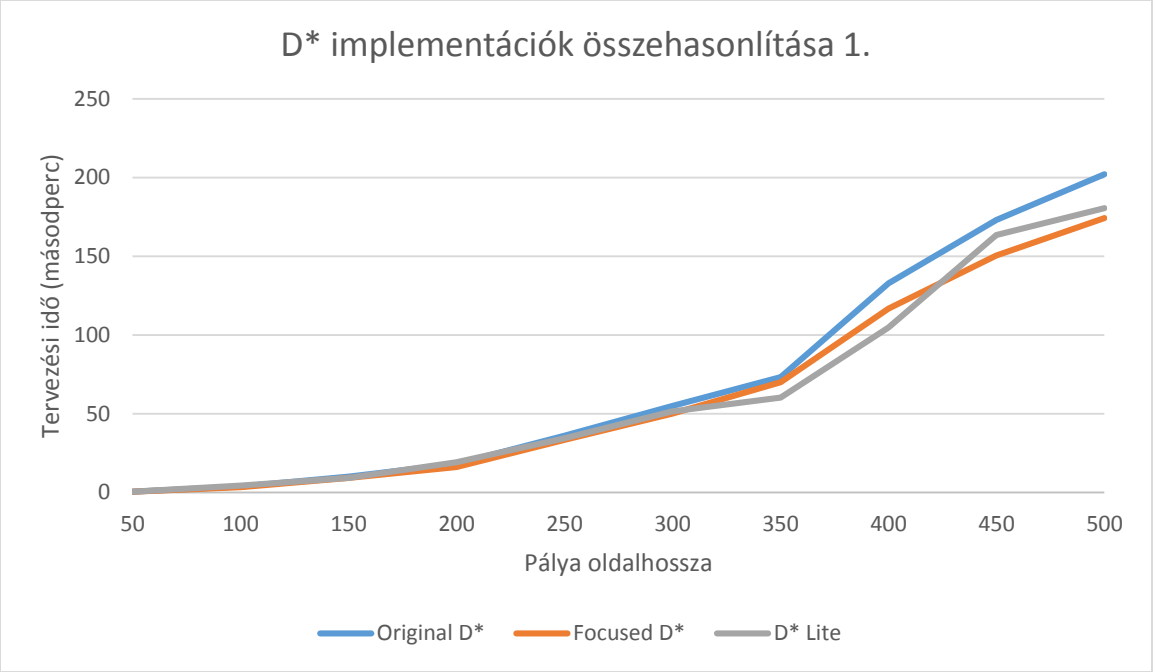


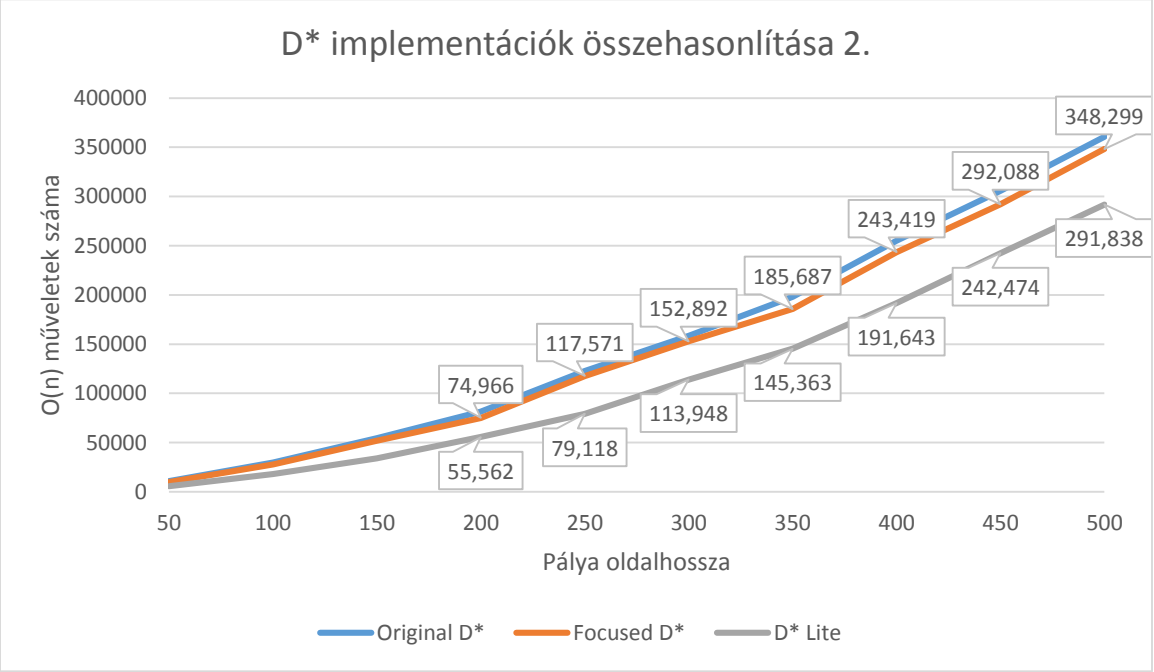
A MÉRÉSEKHEZ HASZNÁLT TIPIKUS PÁLYA



Az A* algoritmus futási idejének gyors exponenciális növekedése miatt 300 x 300-as pályaméretnél megálltam, a program így is több, mint 10 percig futott.

A grafikon szerintem magáért beszél, nagyon szépen látszik a nagyságrendbeli különbség a két algoritmus között.





A két diagramból látszik, hogy ilyen pályaméreteknél nincs nagy különbség a D* algoritmusok között, de a trendvonalat előrevetítve felállítható egy sorrend, ami egybevág a megalkotásuk idejével (Original D*: 1994, Focused D*: 1995, D* Lite: 2002).

Külön érdekesség az első diagramon, hogy míg az Original és a Focused D* algoritmusok futási ideje igen szoros korrelációban áll egymással, addig a D* Lite-nál mástól függ az algoritmus teljesítménye, ugyanis az implementációja más elveken alapul.

FELHASZNÁLT IRODALOM

http://en.wikipedia.org/wiki/D*

http://www.frc.ri.cmu.edu/~axs/doc/icra94.pdf (Original D*)

http://www.frc.ri.cmu.edu/~axs/doc/ijcai95.pdf (Focused D*)

http://idm-lab.org/bib/abstracts/papers/aaai02b.pdf (D* Lite)

http://www.cs.cmu.edu/~./motionplanning/lecture/AppH-astar-dstar_howie.pdf