Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar

Speciális útkereső algoritmusok és alkalmazásaik modern játékprogramokban

Konzulens: Dr. Horváth Gábor, MIT

Készítette:

Tódor Balázs balash@freemail.hu

és Csorba Kristóf kristof@impulzus.sch.bme.hu

Tartalomjegyzék

Előszó

Ebben a dolgozatban három olyan speciális útkereső eljárást fogunk bemutatni, melyek jelentősen eltérnek a klasszikus megoldásoktól, és igen hasznosak lehetnek többek között a modern játékprogramokban felmerülő különböző problémák megoldására. A „gráf nélküli” keresés és a hangya alapú keresés futása közben térképezi fel a keresési teret, majd a „terepviszonyoknak” megfelelően, a lehetőségekhez mérten széles utakat jelöl ki. Céljuk tehát nem egy konkrét út megtalálása, hanem egy olyan (két vagy akár több dimenziós) sáv kijelölése, melyben a célpont felé navigáló objektumok már – csak a lokális környezetet vizsgáló algoritmussal is – komolyabb nehézségek nélkül haladhatnak előre még akkor is, ha közben ki kell kerülniük néhány kisebb akadályt (például társakat). A harmadik bemutatásra kerülő eljárás a lehetséges cselekvések terében történő keresés, mely egyrészt hatékony párost alkothat a gráf nélküli kereséssel, másrészt önállóan is megállja a helyét az olyan helyzetekben, amikor az a tér, melyben keresünk, már túl összetett (például túl magas dimenziójú) ahhoz, hogy a klasszikus A* algoritmus megbirkózzon vele.

Útkeresés „gráf nélkül”

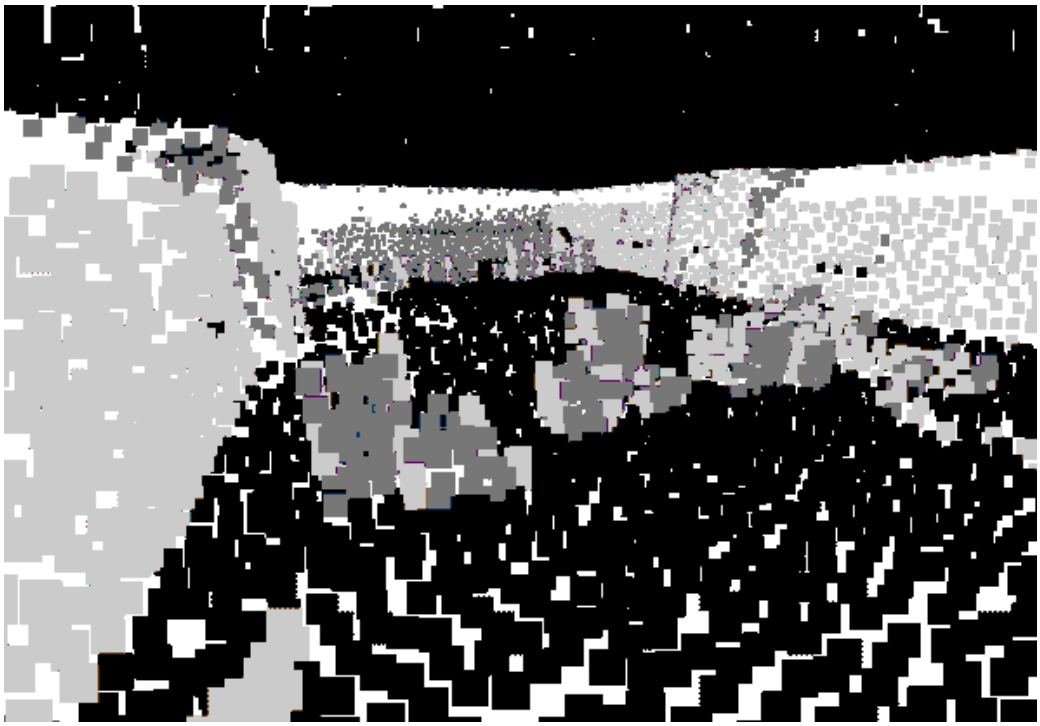
A világmodell

A mesterséges intelligencia egyik legrégibb területe az útkereséshez és útvonaltervezéshez használható algoritmusokkal [Russell] foglalkozik. Ezek az eljárások a világot gráfokkal modellezik. Ezt a leképezést általában emberek végzik – a tervezési idő szempontjából gyakran ez jelenti a legszűkebb keresztmetszetet, hiszen már elég sokféle-fajta (újrahasználható vagy univerzális) keresőeljárást ismerünk, azonban a gráfokat minden egyes problémánál külön meg kell adni. Ez azt jelenti, hogy egy modern játékprogram fejlesztése során, a tervező feladata, hogy a játékbeli környezetekhez hatékonyan használható gráfokat tervezzen. Ez egy egyszerű, de meglehetősen időigényes feladat. (Nem ritka, hogy egy játék akár negyven-hatvan, vagy még több különböző környezetet tartalmaz.) Ha elvonatkoztatunk a játékprogramozástól, akkor könnyen találhatunk olyan problémakört is, amikor egyszerűen nem alkalmazhatunk embereket a megoldáshoz, például, ha a feladat egy ismeretlen környezetben navigáló robot tervezése.

Mi a fenti két probléma megoldását próbáltuk ötvözni egy szimulált világban mozgó ágens tervezésével. A világ adatait a Sierra Half-Life [HL] nevű játékprogramjából vettük át, amely egy háromdimenziós „first person shooter” – magyarul egy játékbeli karakter bőrébe bújva kell az ellenségeket lelőni. A szimulált környezet egy viszonylag nagy, zártnak tekinthető terület, amelyben különböző akadályok, illetve falakkal elválasztott helyiségek találhatók. Az ágens feladata, hogy két, koordinátáival megadott hely között utat találjon. Ez a klasszikus útkeresési probléma, csak ezúttal a gráfot is egy program állítja elő.

A világmodell folytonos, de nem „teljesen” háromdimenziós – ugyanis akkor lenne valóban az, ha az akadályok mindhárom irányban szabadon helyezkedhetnének el, illetve az ágens mozgása sem lenne korlátozott (vagyis a gravitáció hatása az útkeresés szempontjából elhanyagolható lenne). Ehelyett legtöbbször „2,5D-s” világokkal találkozhatunk, amelyekben az ágensek valamilyen felületen (talajon, padlón) mozognak, és csak kevés lehetőségük van ettől eltérni (pl. ugrás) – ezért egy felülnézeti 2D-s modell alkalmazásával általában nem követünk el nagy hibát. Természetesen, ha a cél pl. egy repülőgép, helikopter, vagy tengeralattjáró útvonalának tervezése lenne, akkor ezt az egyszerűsítést nem tehetnénk meg.

Az **1. ábrán** a szimulátor ablaka látható, a világosszürke négyzetek a falpontokat jelölik, a talaj és a plafon színe pedig fekete (ezeket a színeket a normálvektorokból nyerjük).



1. ábra - Amit az ágens a világból "lát"

Térfelosztó algoritmusok

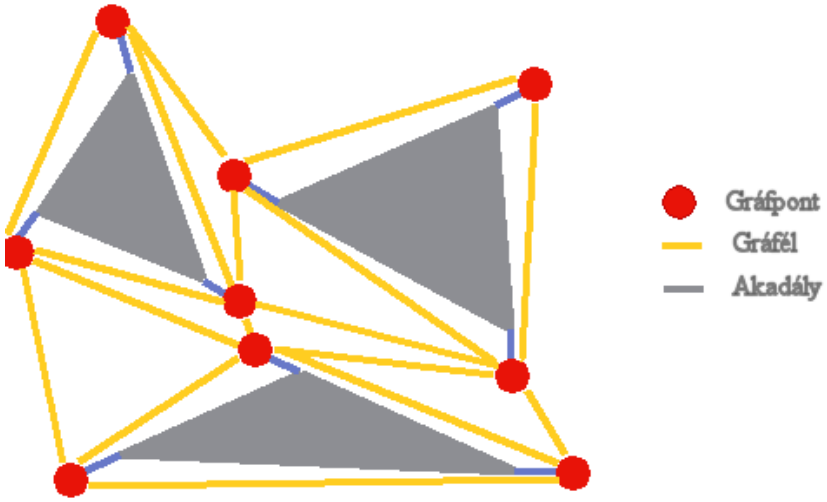
A „gráf nélküli” keresés elemzése előtt tekintsünk át a probléma megoldására már kitalált, működő módszerek közül néhányat! (Természetesen, a mi eljárásunkhoz is szükség van egy gráfra, azonban ezt a keresést végző program emberi beavatkozás nélkül állítja elő.)

Látható csomópontok

A legegyszerűbb eljárás a látható csomópontok (Points of Visibility, PoV) [GPG2, Stout99, UR], amely arra épít, hogy az útkeresés szempontjából csak az akadályok „sarkai” a lényegesek. Ezért olyan keresési gráfot készítünk, amelynek a pontjai az akadályok sarkaitól

adott távolságra vannak, éppen annyira, hogy a csomóponton áthaladó ágens ne ütközzön bele az akadályba. A gráf éleit azon pontok között húzzuk be, amelyek „látják” egymást.

Az eljárás előnye, hogy amíg a cél közel van a falakhoz, addig elég rövid útvonalakat talál, hiszen az akadályok kikerülése során a lehető legközelebb megy hozzájuk. Azonban ehhez viszonylag sok gráfpontra van szükség, a **2. ábrán** látható egyszerű, háromszög alaprajzú testek esetén akadályonként háromra. A valóságban ennél sokkal bonyolultabb falak szoktak előfordulni, amelyek igen gyakran nem is síklapokkal határoltak – ekkor természetesen háromnál jóval több csomópontra van szükségünk, ami jelentősen lassítja a keresést.

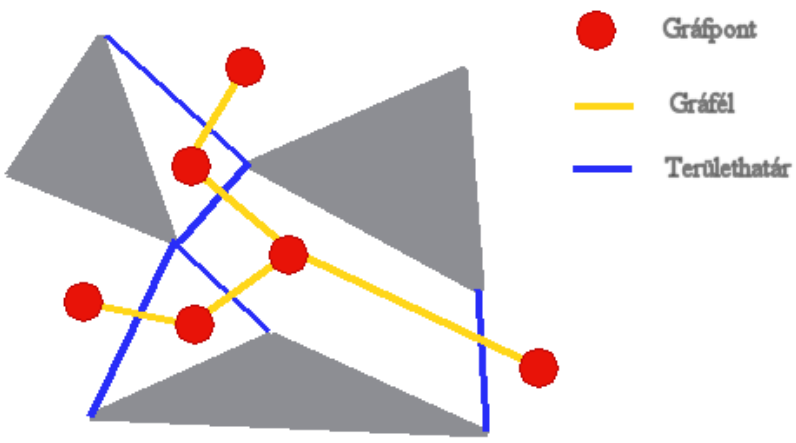


2. ábra - A "Points of Visibility" algoritmus eredménye

Konvex cellák

Egy másik lehetőség a konvex cellák (C-Cells) [Stout99, Kwang92] eljárás használata. Ez annyiban különbözik a PoV-tól, hogy egyrészt az akadályok sarkait csak az onnan látható legközelebbi akadálysarkokkal köti össze, másrészt viszont az így kialakuló szakaszok nem élek lesznek, hanem területhatárok. A keresési gráf felépítéséhez minden területhez egy gráfpontot rendelünk, míg az éleket a szomszédossági viszonyok alapján húzzuk be.

Amint az a **3. ábrán** is látható, az eljárás viszonylag kevés gráfpontot „termel”, cserébe viszont a megtalált útvonalak közel sem lesznek olyan rövidek, mint a PoV esetén. Természetesen előfordulhat olyan eset, amikor messziről el akarjuk kerülni az akadályokat – ekkor a C-Cells jó megoldást jelenthet.



3. ábra - A "C-Cells" eljárás eredménye

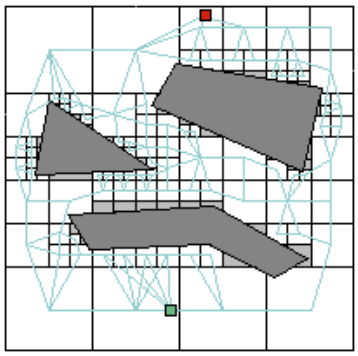
Maximális területekre bontás

A C-Cellshez hasonló módszer a maximális területekre bontás (Maximum-Area Decomposition) [Greulich, Stout99], amely nagyjából azonos méretű darabokra vágja a síkot. Ez az akadályok konvex sarkait vetíti a legközelebbi akadályra, majd az így kapott vágási vonalat összeveti a C-Cells eredményével, és a két szakasz közül a rövidebbet tartja meg. Az így kapott területekhez gráfpontokat rendelünk, amelyeket a szomszédossági viszonyok alapján kötünk össze élekkel. Ennek az eljárásnak (a módszer hasonlósága miatt) ugyanazok a tulajdonságai, mint a C-Cellsnek, azonban annyi előnye van az előbbihez képest, hogy az akadályok közötti területek közel azonos méretű darabokra lesznek felvágva.

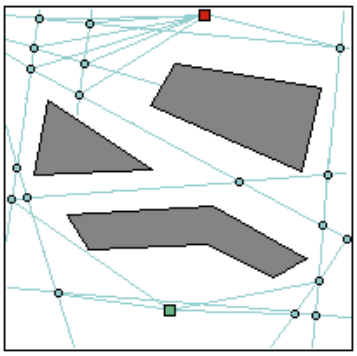
Kvadratikus fa, oktális fa

A hierarchikus útkeresés [Pinter01] szempontjából előnyösebb módszerek közül a legegyszerűbb a kvadratikus fa (quadtree, forrás: [Stout99]), amelynek során négyzetekre bontjuk a síkot. Ha az adott felbontás nem elég pontos, akkor egy-egy négyzetet negyedakkora területű négyzetekre vágunk. Ezt a lépést addig ismételjük, míg a hiba kellően kicsi nem lesz. Ez a módszer közel sem tökéletes, hiszen a „ferde” vonalakat nem tudjuk nulla hibával követni. Cserébe viszont a quadtree-s eljárás térben is alkalmazható, tehát valódi 3D-s problémáknál is működik – csak ilyenkor oktális fának (octtree-nek) hívják, mivel a kockát

nyolc egybevágó kockára vágja. A **4. ábrán** [Stout99] a quadtree működésének eredménye látható.



4. ábra – Kvadratikus fa



5. ábra – Általános hengerek

Általános hengerek

Néha problémát okozhat, hogy a kiszámított útvonal túl közel halad az akadályokhoz (pl. a PoV egyik hibája ez). Ezen segít az „általános hengerek” (generalized cylinders) [Stout99] algoritmus, amelynek a lényege, hogy a két akadály közötti síkra egy olyan hengert fektet, amelynek az akadályok egymás felőli szélei a palást síkra vetített képeit adják. Magyarul olyan általános hengert (vagy csonka kúpot) keresünk, amit a síkunkra fektetve éppen elfér a két, kiválasztott akadály között. Ebből kiszámítjuk a henger hossztengelyét (ez gyakorlatilag egy olyan szakasz, ami mindkét akadálytól egyforma távolságra van), ez lesz a keresési gráf egy éle. A gráfpontokat az „élek” metszéspontjaiba helyezzük. Ez a módszer viszonylag számításigényes, cserébe viszont csak relatíve kevés gráfpontot „termel”, ami az útkeresést gyorsíthatja. Az **5. ábrán** (forrás: [Stout99]) látható, hogy ez az eljárás is területekhez rendel csomópontokat, azonban ezeknek más az elhelyezkedése, mint a konvex cellák esetében – ráadásul a gráfpontok elhelyezésének módszere miatt előfordulhatnak olyan területek is, ahol feleslegesen sok pont lesz.

Erőtér

Érdekes eredményt ad, ha a térfelosztást másként közelítjük meg. A cél az útkeresés során az akadályok elkerülése és a célpont elérése. Ha tehát taszító erőket rendelünk az

akadályokhoz és vonzókat a célponthoz, akkor a gradiens mentén végighaladva elvileg eljutunk a célba. A folytonos térben való gradiensalapú keresés könnyen beragadhat a lokális minimumokba, azonban az alábbi módszerek kiküszöbölik ezt a problémát. A vonzó és taszító erőkön alapszik az erőterek módszere (potential fields) [Baert00, Kwang92].

Hullámfront-terjesztés

Tároljuk le a síkot egy tömbben, mégpedig úgy, hogy a térképet kvantáljuk kis négyzetekre (egyébként ez az egyik legegyszerűbb térfelosztó módszer, az ant search is ezt használja), és mindegyikhez hozzárendeljük a középpontban mérhető térerőt. Az akadályokhoz ezúttal nem rendelünk taszító erőt, csak a célhoz vonzót. Elindulunk a céltól, nulla értéket rendelünk hozzá, majd az összes szomszédos cellának 1-et adunk. Az ezekhez szomszédos mezőknek kettőt, és így tovább, amíg csak van üres elem a tömbben (**6. ábra**).

	2	2	2	2	2	
	2	1	1	1	2	
	2	1		1	2	
	2	1	1	1	2	
	2	2	2	2	2	

6. ábra - Hullámfront-terjesztés

Ezzel tulajdonképpen egy lineáris „inverz” mezőt tároltunk le, amelynek értéke a cél felé haladva csökken – így keresés helyett egyszerűen a kezdőpontból kiindulva mindig a legkisebb értéket tartalmazó cellába lépünk, rekurzívan. Ezt az eljárást hívják wavefront expansion-nek (hullámfront-terjesztés) [Baert00] – ez azonban gyakran nem a legjobb utakat adja meg. Ennek az egyik oka a térerősség-számoló eljárásban keresendő: nem valódi távolságmérést használunk a számításhoz, azaz egy pont körül az összes (nyolc) szomszédot azonos távolságúnak vesszük, pedig a valóságban az átlósan elhelyezkedő mezők 1,4-szer (gyök kettőször) akkora távolságlépést jelentenek, mint a többi (**7. ábra**). Ez azt jelenti, hogy bizonyos irányok a valóságban előnyösebbek, mint ahogy a tömb mutatja. Mivel a

keresőalgoritmus a tömb alapján dolgozik, ezért nagy valószínűséggel nem az optimális utat fogja visszaadni.

Ezen a problémán segíthetünk, ha a tömböt távolságmérést használva töltjük ki. Ennek az eredménye az alábbi ábrán látható. Ezzel azonban a számító eljárást lassítottuk.

	2,8	2,4	2	2,4	2,8	
	2,4	1,4	1	1,4	2,4	
	2	1		1	2	
	2,4	1,4	1	1,4	2,4	
	2,8	2,4	2	2,4	2,8	

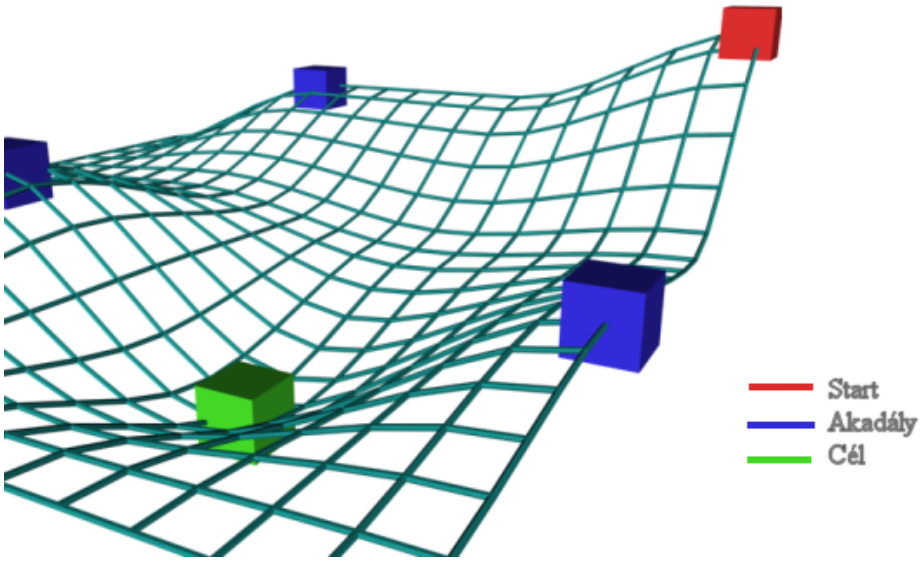
7. ábra - Javított hullámfront-terjesztés

Javíthatunk a megoldáson, ha a hullámterjesztést nem a célpontból végezzük, hanem az akadályokból: mindegyik objektum saját „azonosítójú" erőteret terjeszt. Amikor egy cellába több különböző tér értéke kerülne, akkor azt a cellát megjelöljük (és ezekből nem is lépünk tovább). Az eljárás végén egy olyan térképet kapunk, amin az objektumoktól azonos távolságra lévő útvonalak vannak megjelölve (az eredmény a Voronoi-diagramra [Aurenh] hasonlít). Ezután alkalmazzuk a célpontból a hullámfront-terjesztést a megjelölt cellákra, de úgy, hogy csak az útvonalakon lépkedünk. Így az egységek mozgás közben az összes akadályt messziről elkerülik (kivéve persze, ha az akadályok olyan közel vannak egymáshoz, mint a **8. ábrán**), valamint az útvonal sem lesz olyan szögletes, mint a korábbi algoritmusoknál. Ráadásul így a célpont változásakor sem kell az egész tömböt újraszámolni, (csak akkor, ha az akadályok elmozdulnak) – elég ugyanis a megjelölt cellákra elvégezni az új célpontból a hullámfront-terjesztést.

		1	2	3	4,3	
	1	1	2	3,2	2	
	2	2	2,2	3,1	1	
	3	3,3	3,2	1		
	4,4	3	2	1	1	

8. ábra - Hullámfront-terjesztés az akadályokból

Természetesen a módszer valódi 3D-s esetben is alkalmazható, csak ekkor a memóriaigény miatt valószínűleg nem fogjuk tudni alkalmazni a hullámfront-terjesztést (túl nagy tárterület kellene a tömbhöz), ezért valódi gradiens-módszereket kell használni. Ezek azonban legtöbbször nem alkalmasak real-time használatra, ráadásul nem is mindig találnak jó útvonalat (hajlamosak a lokális minimumokba való „beragadásra").



9. ábra - Erőtér gradiensalapú kereséshez

Gráfépítés keresés közben

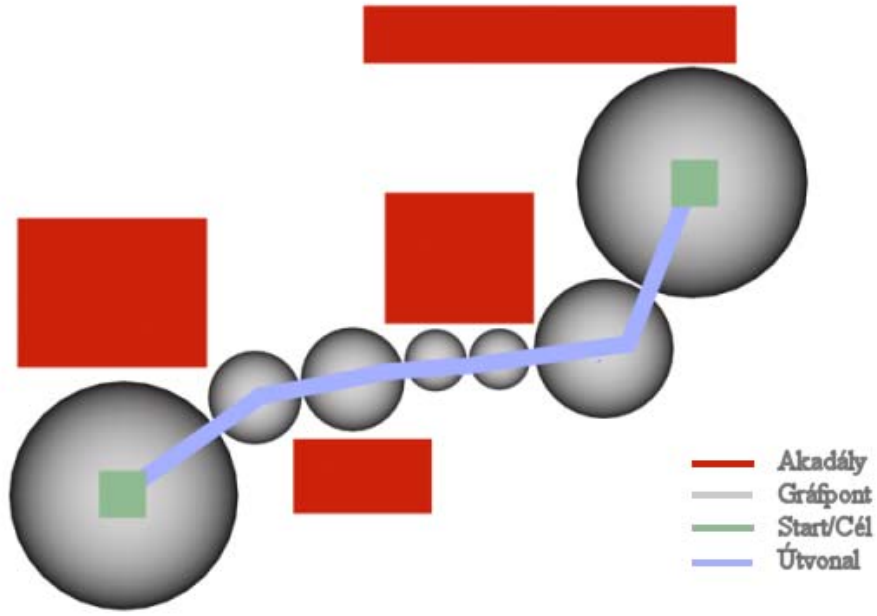
A fenti módszereknek van egy nagy (közös) hátránya is: mindegyikhez szükség van a teljes világ ismeretére. Ezért a problémát máshogy közelítjük meg. Olyan eljárásra lenne szükség, amely csak a keresés által bejárt területeket alakítja diszkrétté – sőt, még jobb lenne, ha keresés közben végezné el az átalakítást. Mivel a klasszikus keresési módszerek már eléggé kiforrottak, ezért célszerű ezeket használni. Ehhez azonban szükség van egy gráfra is, amelyet ezúttal a keresés közben építünk fel. Ennek során a gráf pontjait véges kiterjedésű testekkel helyettesítjük. A méret megválasztásánál azonban vigyázni kell, mert ha túl kicsik a „pontok", akkor túl sok lesz belőlük, ami a keresést lassítja, míg ha túl nagyok, akkor veszítünk a pontosságból, és esetleg az útvonal belevezetheti az ágenst az akadályokba. Mi a pontokhoz gömböket rendeltünk.

Bár a gömbökkel nem tudjuk a teret hézagmentesen „lefedni", de ez az átmérő megfelelő megválasztása esetén elhanyagolható hibát okoz, és valamelyest még csökkenthető is, ha megengedjük, hogy a gráfpontok kismértékben átlapolódjanak. Ennek ellenére előfordulhat olyan eset, amikor még ilyen kis hibát sem engedhetünk meg – ekkor használhatunk a gömbök helyett más testeket is (pl. téglatesteket, kockákat). Természetesen az algoritmus bonyolultabb lesz, mint a gömbök esetén.

A pontméret megválasztása

Azért, hogy csak olyan útvonalakat találjunk meg, amelyeken az ágens valóban el tudna haladni, célszerű a gömbméretet az egység méretéhez igazítani. Ezeket mi a játékprogramból kaptuk, amely befoglaló téglatesteket használ, 36x36x72 mérettel. Ha a karakter „emberszerű", akkor képes leguggolni is, ekkor a mérete 36x36x36-ra csökken.

Természetesen lehet dinamikus átmérőt is alkalmazni, pl. minél közelebb vagyunk egy akadályhoz, annál kisebbeket – így a felbontás csak ott lesz nagy, ahol ezt a terepviszonyok valóban megkövetelik. A gráfpontok méretének meghatározásához általában elég csak a legközelebbi akadályt figyelembe venni. Ez azt jelenti, hogy amikor egy új gráfpont méretét meghatározzuk, akkor elég csak a legközelebbi falponttól való távolságot lemérni. A gömb mérete ennél kisebb kell legyen, mert különben a következő pont esetleg átkerülhetne a fal túlsó oldalára.



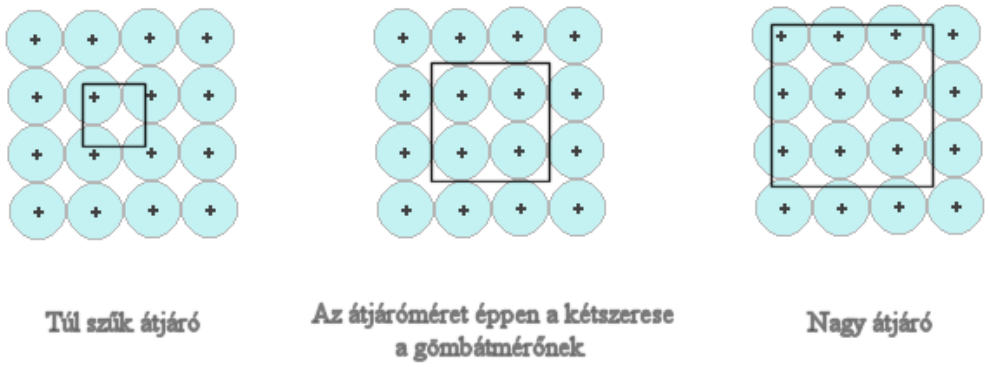
10. ábra - A keresés közbeni gráfépítés eredménye

Az akadályok és falak tárolása

A folytonos világ akadályait véletlenszerű mintavételezéssel írjuk le, mivel az algoritmus működéséhez ugyanis általában elég csak akadályonként néhány pont ismerete. Ha ezekhez a pontokhoz is (alkalmasan megválasztott) méretet rendelünk, akkor viszonylag kevés ponttal is jól leírhatjuk a tereptárgyakat. Természetesen, minél nagyobb a pontméret, annál nagyobb lesz az a minimális átjáró, amit még „észrevesz” az eljárás, ezért ennek a paraméternek a megválasztásához szükség van valamennyi előismeretre.

Ha semmit nem tudunk a környezetről, akkor célszerű az egység ún. orientáció-független méretének (a legkisebb befoglaló gömb átmérője) legfeljebb felét használni. Az első mintavételi törvény értelmében ekkor a legalább kétszer ekkora lyukakat biztosan tudjuk detektálni. (Pontosabban: az ennél nagyobbakat. Azonban a lebegőpontos számok használatával az egyenlőség esélye kicsi.) Egy átjáróban akkor fér el az ágens, ha a keresztmetszetben legalább négy gömb elfér.

Ez tehát egy ésszerű felső korlát. Az alsó határt a szükséges futási sebesség (minél kisebbek a pontok, annál több kell belőlük egy útvonal lefedéséhez), illetve az ágens pozíciójának pontossága (ld. lentebb) adja.



11. ábra - Különböző méretű átjárók detektálása (a kék körök falpontokat jelölnek)

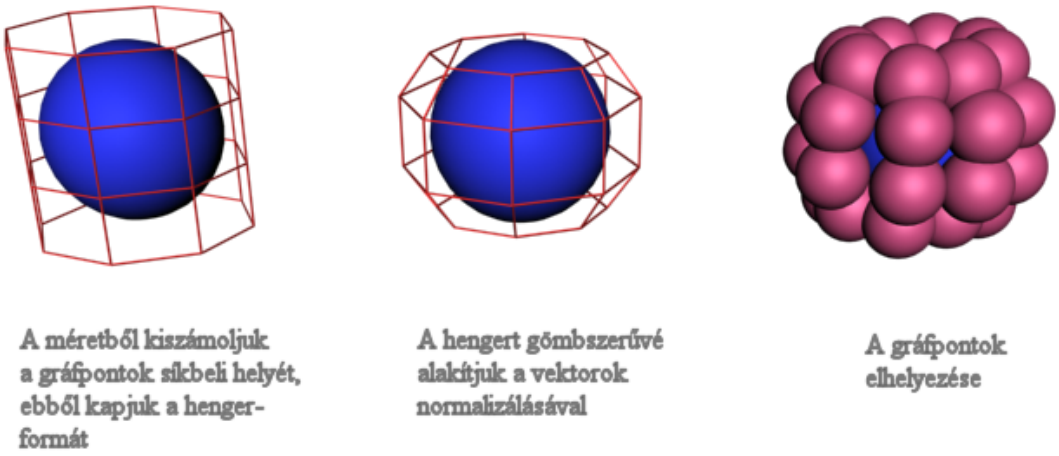
Ezeken kívül még egy fontos tényező is befolyásolhatja a pontsűrűséget: az ágensre veszélyes gödrök, csapdák. Ha ezek kisebbek, mint az orientációfüggetlen méret, akkor a sűrűséget növelni kell, különben semmi nem garantálja, hogy a mintavételezés során észrevesszük őket.

Mi a pontinformációt a játékprogramból kapjuk. Ehhez csak egy egyszerű távolságmérést használunk, ami az ágens felépítését egyszerűvé teszi. A valóságban persze ennél kicsit bonyolultabb a helyzet, ugyanis a távolságmérőnek forgathatónak kell lennie, és az orientációját is ismernünk kell. Ráadásul emellett szükség van egy abszolút pozíció ismeretére, amit a mi esetünkben szintén a játék ad. Az eljárás működési elve miatt a helymeghatározásnak nem szükséges túl pontosnak lennie, elég, ha a hibája a választott pontméret kb. fele alatt marad.

A keresés menete

A keresés kezdetekor a kezdőpontra rakjuk le az első gráfpontot. A szomszédos pontok meghatározásához kikötjük, hogy érinteniük kell a kezdőpontot. Az egyszerűség kedvéért minden szomszédot azonos méretűnek veszünk. A „2,5D”-s problémakörnek megfelelően a szomszédok elhelyezésekor igyekeztünk a talajjal párhuzamos irányokban több pontot elhelyezni, a felfelé és a lefelé találhatók rovására. A szükséges méretből kiszámítjuk, hogy a

gömb sugarával megegyező sugarú kör körül milyen távolságra kellene elhelyezni a pontokat. A szomszédos középpontokat összekötjük, majd az így kapott sokszöget felfelé és lefelé is eltoljuk, úgy, hogy a szomszédos másolatok között éppen egy átmérőnyi távolság legyen. Így egy hengerszerű testet kapunk. Ennek a sokszögeit úgy méretezzük át, hogy a kezdőpont középpontjából a csúcsokba mutató vektorok hosszát egységesen átállítjuk az eredeti sugár és a szomszédos pontok sugarának összegére. Ezzel gyakorlatilag behúztuk a szomszédokat a gömb felületére.



12. ábra - A szomszédok elhelyezése

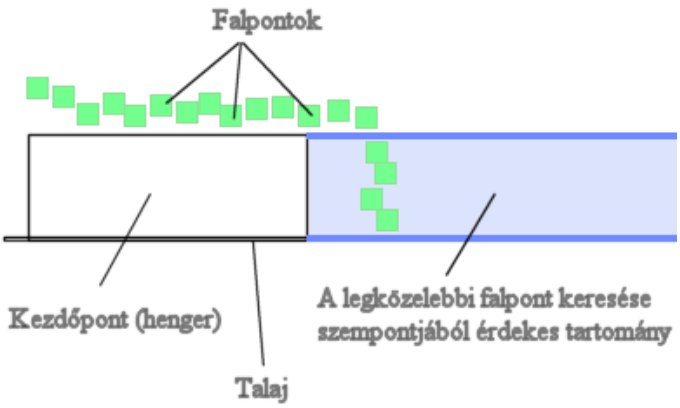
Ha a szomszédok mérete sokkal kisebb, mint a kezdőpont, akkor nagyon sok gráfpontot tárolunk el, ami az eljárást lassítja. Ezen úgy lehet segíteni, hogy amikor kiszámoljuk a pontok helyeit, akkor hasznosság szerint sorbarakjuk őket, és csak a legjobbakat (nem mindet) tároljuk el. Ez persze azt is jelenti, hogy ha a keresés nagy akadállyal találkozik (vagyis sokat kell „visszalépnie”), akkor nem az optimális útvonalat fogja visszaadni.

Henger alakú gráfpontok

Ha az eljárást csak 2,5D-s problémákra akarjuk alkalmazni, akkor a gömb helyett lehet jobb testet is találni. Ez különösen a nagy alapterületű, de alacsony belmagasságú termekben való navigáláshoz jöhet jól, hiszen ilyenkor előfordulhat, hogy a pontok méretét nem az akadályok, hanem a padló-plafon távolság korlátozza. Ilyenkor a gömbök helyett például

adott magasságú hengereket használhatunk. Ehhez előre ismerni kell a gravitáció irányát. A henger alapját a talajjal párhuzamosan helyezzük el. Ha az ágens tudja a magasságát változtatni (pl. guggolás), akkor kétféle hengermagasság fordulhat elő.

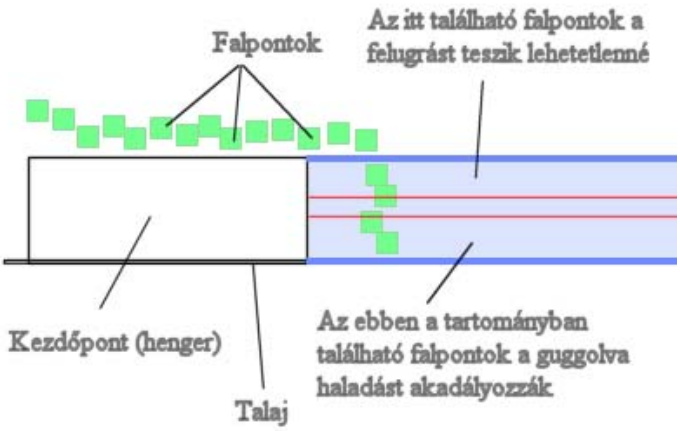
A gráfpontok száma csökkenthető (és így a teljesítmény növelhető), ha a szomszédokat kereső eljárásban figyelembe vesszük a henger magasságát. Az ágens szempontjából ugyanis a falpontok távolsága csak akkor számít, ha azok az „útjában" vannak. Ezzel a lépéssel elértük, hogy a nagy termekben csak annyi gráfpont lesz, amennyire valóban szükség van.



13. ábra - A közeli falpontok keresése

Némileg bonyolítja a helyzetet, ha a talaj nem folytonos (ilyen a valódi életben is van: lépcső, járdaszegély). Ahhoz, hogy a keresés ezt kezelni tudja, figyelembe kell venni a falpontok talajhoz mért távolságát is. Ha ez kisebb, mint a legnagyobb átléphető magasság, akkor az adott pontot figyelmen kívül lehet hagyni. Mi ezt az információt a játékból kaptuk, míg egy kerekes robot esetén kb. a hasmagasságnál alacsonyabb akadályokat nevezhetünk „átléphetőnek".

Hasonló problémát okoz az is, ha az ágensnek ugrania kell. Az ilyen helyzetekre az jellemző, hogy a vizsgált térrész alsó részében a falpontok közel vannak, míg a felsőben messzebb (tehát van egy perem, amin az egység meg tud állni). Bonyolítja a helyzetet, hogy a mi esetünkben az ágens magasabbra tud ugrani, mint amekkora a „leguggolt" mérete. Ez azt jelenti, hogy nem elég csak annyit tudni, hogy a legközelebbi falpont hol van, és annak a magasságából meghatározzuk, hogy át lehet-e bújni alatta, vagy fel kell ugrani rá. A megoldás a falpontok keresésére kijelölt térrész több részre osztása.



14. ábra - A falpontok keresése guggolás/ugrás esetén

Az alsó sáv (**14. ábra**) a guggolási magasságig tart. Ha ebben a tartományban van egy ágens-szélességnyi hely, akkor ott elhelyezhetünk egy gráfpontot. Az efölött található tartomány a legnagyobb elérhető magasságig terjed. Ha ebben a tartományban találunk falpontot, de fölötte nem, akkor egy olyan peremet találtunk, ahová az ágens fel tud ugrani. Ennek a feltétele az, hogy a legfelső tartományban levő legközelebbi pont elég messze legyen. Az általunk választott játékban az a perem, ami elég széles az ágens megtartásához, jóval kisebb, mint az ágens mérete. Ez azt jelenti, hogy ha azt akarjuk, hogy az egység olyan helyekre is feljusson, ahová több, egymást követő ugrással lehet csak „felmászni", akkor meg kell engednünk az ágens méreténél kisebb gráfpontokat. Ez nem okoz gondot, ha kikötjük, hogy csak ugráláskor használhatunk ilyen csomópontokat.

Henger vagy gömb?

Mindkét típusnak megvannak a maga előnyei és hátrányai. A hengerek nagyon jól alkalmazhatók 2,5D-s esetben, ha viszonylag keveset ugrál az ágens. Ez az eljárás ugyanis a vízszintes irányokat részesíti előnyben. Ennek ellenére alkalmas a több ugrást tartalmazó (felfelé vezető) útvonalak követésére is. Viszont kevésbé hatékony akkor, ha az ágens relatíve nagyokat tud ugrani, pláne, ha „repülés" közben irányváltoztatásokra is képes (ez utóbbi az általunk választott játékban is fennáll, azonban szerencsére a levegőben töltött idő túl kevés

ahhoz, hogy jelentős hibát okozzon). A hengeres módszer könnyebben észreveszi az alacsony átjárókat, hiszen a hengerek alapja mindig a talajszinten van.

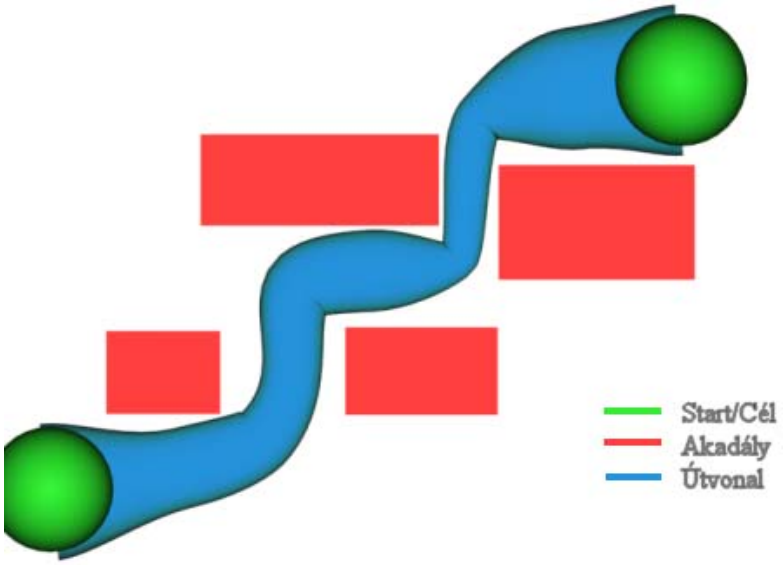
Ezzel szemben a gömbök használata esetén a nagyobb gömbök középpontja magasan a talajszint fölött van, ezért egy nagy teremben a talaj szintjén lévő átjárókat nem biztos, hogy észreveszi. Ezen valamelyest lehet segíteni a maximális gömbméret csökkentésével. Ha a környezet közelebb áll a 3D-shez, mint a 2,5D-shez, (az ágens hosszabb ideig képes pl. a levegőben maradni), akkor célszerű a gömböket választani, mivel ez – még a vízszintes irányokat favorizáló szomszédválasztás esetén is – több gráfpontot helyez el felfelé és lefelé, mint a hengeres eljárás. Így a hosszabb ugrással elérhető utakat biztosabban megtaláljuk a gömbök alkalmazásával.

Egy másik fontos szempont a sebesség lehet. Ha csak a legjobb szomszédokat tároljuk el, akkor a gömbös eljárás teljesítménye valamivel jobb, mint a hengeresé. Ennek az oka az, hogy a legközelebbi falpontot kiválasztó eljárás jóval egyszerűbb és gyorsabb.

Az eredmények értékelése

Az eljárás előnye, hogy a kikeresett útvonal lekövetése egyszerűbb, mintha „gráfos” keresést alkalmaztunk volna. Ebben az esetben ugyanis a játékprogramokban külön simítani, szűrni kell az utat [Pinter01], azért, hogy az ágens mozgása ne legyen darabos. Ezek a problémák nem csak a játékban való alkalmazáskor, hanem a robotban való használatkor is felmerülnek, mivel a robot sem „szereti” a szögletes útvonalakat, és ezeket általában lassabban tudja követni, mint a folytonos utakat.

A keresés közbeni gráfépítés ezt a problémákat kiküszöböli azzal, hogy a gráfpontokhoz méretet rendel. Ez azt jelenti, hogy az úthoz egy előre ismert követési hibát is adunk (**15. ábra**). A keresés eredménye tehát út helyett egy változó szélességű „sáv” (úgy, mint az ant search esetében, ld. később), amelyben az ágens szabadon mozoghat, anélkül, hogy bárminek is nekiütközne. Természetesen, szűk átjárókban ez a sáv lecsökkenhet annyira, hogy éppen az ágens orientációfüggetlen méretével lesz egyenlő, de ha feltesszük, hogy az egység csak előre tud haladni (és hosszabb, mint amilyen széles), akkor még mindig marad valamennyi oldaltávolság.



15. ábra - Az útvonal megengedett követési hibája

További előny, hogy az eljárás valódi 3D-s esetben is működik, csak a szomszédok kiválasztását kell átírni, úgy, hogy ne részesítsen előnyben egy irányt sem – vagyis adott számú gömböt kell elhelyezni hézagmentesen egy gömb felületén.

Az eljárás nagy hátránya az alacsony sebesség. A keresés közbeni gráfépítés viszonylag lassú, annak ellenére, hogy a keresési algoritmus ugyanaz, mint a „gráfos" esetekben. Ennek az oka az, hogy a klasszikus módszerekhez ember tervezi a gráfot, és így az sokkal kevesebb (és jobban elhelyezett) pontot fog tartalmazni, mint a mienk. Másrészt viszont minden egyes gráfpont generálásához ki kell keresni a legközelebbi falpontot. (Már egy kisebb világ is legalább tízezer pontot tartalmaz.)

Bővítési lehetőségek

A futási sebességén lehet javítani, egy olyan segédeljárással, amely a nagy, nyílt terekre (a maximalizált pontméret miatt ezekben általában túl sok gráfpont szokott lenni) előre elhelyez néhány, a megengedettnél nagyobb pontot, amelyeket később a keresés felhasználhat.

A legközelebbi falpont kikeresését egy jobb tárolási struktúrával lehetne javítani. Például egy egyszerű BSP-fával (binary space partitioning tree) a szükséges összehasonlítások számát tízezer helyett le lehetne csökkenteni néhányszor százra.

Egy másik továbbfejlesztési lehetőség a dinamikus környezet és a változó akadályok kezelése. Ilyen lehet például egy időnként kinyíló ajtó, illetve, a robot esetében akár egy felboruló kólásüveg is. A mostani verzióban a feltérképező eljárás nem ellenőrzi, hogy a környezet megváltozott-e. Ehhez a régi pontokat lassanként el kell felejteni – ez persze nem ilyen egyszerű, mivel ha az ágens sokáig nem jár egy környéken, akkor könnyen elfelejtheti az összes információját arról a területről.

Keresés sokdimenziós terekben

Nem csak a pozíció számít

Vannak olyan esetek, amikor bár a világmodell háromdimenziós, de a keresési tér ennél több szabadsági fokkal fog rendelkezni. Ilyen lehet például, ha a kiindulási és célállapotban nem csak az ágens helye, hanem orientációja is érdekes. Erre kiváló példa, ha egy autónak keresünk utat, hiszen vannak olyan kanyarok (sőt, a legtöbb kanyar ilyen), amelyeket csak adott irányba nézve lehet bevenni. Ez máris négy dimenzió (két pozíció, két irány). Vagy: egy aszteroidákat kerülgető űrhajó sem tud bármerre kanyarodni – ez már hat dimenziót jelent. Persze ennél találhatunk sokkal hétköznapibb példát is, elég, ha a közelharcot folytató vadászrepülőkre gondolunk. A harci pilótákat ma már szimulátorokon is oktatják, ezért fontos minél jobb szimulátorok készítése, ahol fontos szerepe lehet a mesterséges intelligencia módszerei alkalmazásának. A közelharcban, ahol a kis távolság miatt csak a gépfegyver használható, mindennél fontosabb, hogy a támadó gép jó „pozícióban" legyen, azaz a célpont fölött, mögötte, és lehetőleg legyen nagyobb a sebessége is (közeledjen hozzá). Ez máris hét dimenziót jelent (három pozíció, három orientáció, egy sebesség), feltéve, hogy a repülő sebessége az orra irányába mutat. De a robotika területén ennél több-dimenziós problémák is léteznek – például, ha a cél egy sok-szabadságfokú robot tervezése.

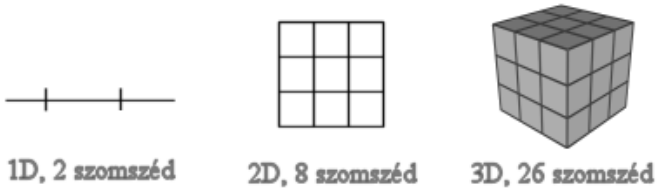
A fejezet első részében a cselekvéstéri keresés módszereit tárgyaljuk, majd rátérünk a hatékonyságot és keresési sebességet növelő trükkökre. A fejezet végén pedig megmutatjuk, hogy hogyan lehet egyszerű szabályozókört készíteni keresés segítségével.

Sok dimenzió – kevés cselekvés

Természetesen a sokdimenziós terek bejárása elvileg megoldható az első fejezetben leírt dinamikus gráfépítéssel is, gyakorlatilag csak a szomszédkeresést kell átírni többdimenziósra. Azonban sokkal egyszerűbb megoldásokat is találhatunk: általában megfigyelhető, hogy a természetben a bonyolult rendszerek sokdimenziós mozgatását viszonylag kevés

„mozgatóprimitív” végzi [Nell02]. A repülőgépek egy egyszerű modellje esetében ez tízféle primitívet jelent: három dimenzióban foroghatunk két irányban, növelhetjük, vagy csökkenthetjük a tolóerőt, illetve kinyithatjuk, vagy behúzhatjuk a fékszárnyakat.

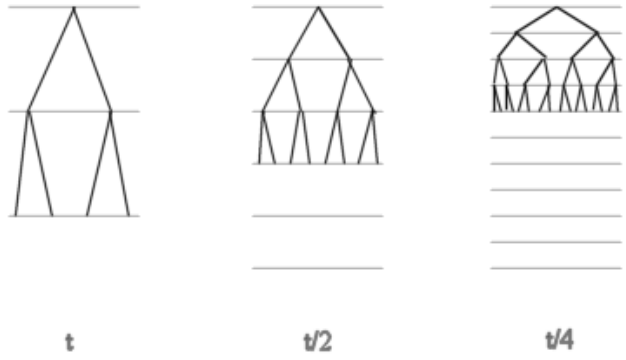
Ha tehát a keresési fát ilyen operátorokkal építjük fel, akkor tízes nagyságrendű (a cselekvéseket többféle paraméter-beállítással használjuk, ld. később) lesz az elágazási tényező. Ezzel szemben, ha dinamikus gráfépítéssel próbálkozunk, akkor (az egyszerűség kedvéért a számításhoz gömb helyett négyzetet, illetve kockát használva) „n” dimenzióban 3^n-1 lesz ez az érték, vagyis hét dimenzió esetén 2186. Persze ezen a dinamikus pontméret valamennyit javít, azonban még így sem veheti fel a versenyt a primitívtérben végzett keresés 10-es értékével.



16. ábra - Szomszédos gráfpontok

Cselekvésalapú gráfépítés

Ezt az eljárást akció-alapú diszkretizálásnak (action-based discretization [Nell02]) nevezzük, és többféle verzió létezik belőle, aszerint, hogy melyik paramétert választjuk állandónak az algoritmus során – ugyanis rögzíthetjük a primitívek kezdő időpontját és a paramétereit is. A legegyszerűbb, ha a cselekvések paraméterei fixek, csak az időparaméter változik, ez a SADAT (Static Action Parameters, Dynamic Action Timing, állandó cselekvésparaméterek, változó időzítés) algoritmus. Persze ehhez egy új keresési módszerre van szükség, hiszen a hagyományos eljárások azonos méretű darabokra vágják a keresési teret. Nekünk azonban olyan algoritmusra van szükségünk, ami lehetővé teszi, hogy a cselekvések időtartamát lerövidítsük, vagy meghosszabbítsuk. Ez egy meglehetősen bonyolult probléma, amit sokkal könnyebb megkerülni, mint megoldani.



17. ábra - Iteratív időfinomítás (SADAT)

Az iteratív finomítás

A változó cselekvéshosszak problémájának „megkerülésére” találták ki az iteratívan finomító (iterative refinement) módszert [Nell02]. Az eljárás lényege, hogy az időparamétert egy kezdeti értékre állítjuk, majd elvégzünk egy keresést. Ezután az időparamétert csökkentjük (azaz az időfelbontást finomítjuk), majd újra keresünk. Ezt addig ismételjük, amíg megoldást nem találunk. Ennek az előnye az, hogy a lehető legkevesebb cselekvést igénylő megoldásokat találjuk meg – az azonban egyáltalán nem biztos, hogy ezek más szempontból is optimálisak lesznek, mivel nem feltétlenül a legegyszerűbb cselekvéssorozat eredményezi a legrövidebb utakat.

A klasszikus algoritmusok közül soknak megírták már az iteratívan finomító verzióját (kezdve az egyszerű mélységi kereséstől az ε-A*-ig [Nell02]). A mi tapasztalataink azt mutatják, hogy az egyszerűbb eljárások is éppen annyira jók, mint a bonyolultabbak. Ennek az oka az, hogy ha nem játékot vagy szimulátort tervezünk (hanem mondjuk modellező programhoz plugint), akkor gyakorlatilag nem számít a sebesség, ezért általában elég az egyszerűbb algoritmusokat implementálni. Ha viszont futásidő-kritikus alkalmazásban szeretnénk használni a cselekvéstéri keresést, akkor tapasztalatunk szerint a mai számítási teljesítmények mellett a bonyolult eljárások sem lesznek elég gyorsak.

Az iteratívan finomító algoritmusok közel sem tökéletesek. Ennek az egyik oka az, hogy előre ismernünk kell azt a teljes időtartamot, amennyi a megoldás eléréséhez kell. A másik az, hogy a kapott „útvonal” távol áll a természetestől, mivel egyszerre csak egy cselekvés van benne, ami szögletessé, darabossá teszi a viselkedést. Hiányzik tehát a mozgáskoordináció.

Dinamikus cselekvés-paraméterek

Ezekre megoldást jelenthet a DADAT (Dynamic Action Parameters, Dynamic Action Timing) eljárás [Nell02]. A cselekvések ebben az esetben dinamikusak, ami újabb problémát vet fel. Ezzel a lépéssel ugyanis a keresési tér újra folytonos lett. Ennek megoldására a cselekvési primitívekből komplex cselekvéseket építünk fel, amelyek a primitívek különböző lineáris kombinációit tartalmazzák. Erre több módszer is van.

Képzeljünk el egy olyan repülőt, ami csak két tengelye körül tud fordulni! Ha a gépet a vége körül forgatjuk, akkor az orra egy gömb felületét írja le. A gömbben levő összes pont a gömbi koordinátái révén egy-egy elforgatást ad meg. Ehhez még hozzávehetjük harmadik szabadsági foknak a tolóerőt is, amit a gömb középpontjától való távolságnak feleltetünk meg, és máris kész a repülőgép. (Természetesen, ha a harmadik tengely körüli forgatást és a fékszárnyat is be akarjuk venni a modellbe, akkor a „gömb" három helyett ötdimenziós lesz.)

A cselekvések kiválasztása

Az akciók megválasztásának legegyszerűbb módszere, ha véletlenszerűen szétszórunk pontokat a gömbben (*random discretization*). Ennél valamivel jobb eredményt ad, ha a pontokat egyenletesen osztjuk el a gömb térfogatában (*uniform discretization*). A harmadik lehetőség a „szétterített" pontok módszere (*dispersed discretization*). Ennek során a pontokat véletlenszerűen helyezzük el, majd taszító erőket rendelünk hozzájuk, amelyek pl. $1/r^2$-esen hatnak. Ezután néhányszor (pl. ezerszer) kiszámítjuk az új helyüket, az erőtérnek megfelelően mozgatva őket. A tapasztalatok szerint a pontok jelentős része a gömb felülete környékén fog elhelyezkedni, ez azonban az algoritmusok működését nem befolyásolja.

Az eddigi módszerek közül a leghatékonyabb a DADAT, mivel ez az összes paramétert engedi változtatni. A gyakorlatban azonban erre ritkán van szükség, mivel – időkritikus esetben – már a keresés egyszeri lefuttatása is elég hosszú ideig tart ahhoz, hogy ne legyen időnk további időlépcsők kipróbálásához. Ha elhagyjuk az iteratív időfinomítást a DADAT-ból, akkor a DASAT (Dynamic Action Parameters, Static Action Timing, [Nell02]) algoritmust kapjuk. Ennek az előnye az, hogy a keresést csak egyszer kell lefuttatni

(használhatók a klasszikus eljárások), cserébe viszont egyáltalán nem biztos, hogy a legkevesebb cselekvést tartalmazó megoldást találjuk meg. Ezt a cselekvési időtartam helyes megválasztásával lehet valamelyest ellensúlyozni. Egy másik lehetőség a lehetséges komplex cselekvések számának növelése. A tapasztalat azt mutatja, hogy minél többféle cselekvést engedünk meg, annál nagyobb lehet az időlépcső. Ez egyébként érthető is, hiszen a cselekvések mozgatóerőket jelentenek, a mozgásegyenlet megoldása pedig egy integrál. Vagyis a rövid ideig tartó nagy erők és a hosszú ideig tartó kicsik hatása azonos, ha az erőgörbe alatti terület egyenlő. Ezt a tényt kihasználva a DADAT eljárás helyett a DASAT-ot alkalmaztuk.

Dinamikus akcióparaméterek, statikus időosztás

Az is a DASAT mellett szól, hogy a számítási pontatlanságok miatt a fizikai szimuláció eredménye függ az időlépcsőktől. Ez azt jelenti, hogy ha kiszámoljuk egy objektum helyét egy másodperces lépésekben tízszer, majd két másodperces lépésekben ötször, akkor nem pontosan ugyanazt az eredményt fogjuk kapni. Ez a probléma természetesen csak időkritikus programokban lép fel, mivel ezekben a fizikai rendszer állapota minden kiszámított képben egyszer frissül. Az időkritikusság azt jelenti, hogy amint a program végzett egy kép számításával, rögtön nekilát a következőnek – vagyis az állapot frissítésének periódusideje nem állandó, nem lehet előre tervezni. A gond akkor van, ha egy időlépcső hosszabb, mint egy kép megjelenítési időtartama. Ekkor ugyanis az adott cselekvést legalább két lépésben kell végrehajtani, vagyis az eredmény valószínűleg el fog térni a szimulációétól. Az olyan alkalmazásokban, amelyek vagy nem időkritikusak (pl. modellezőprogramok, ahol a másodpercenként kiszámolt képek számát valamilyen minimális értékre szokták rögzíteni), vagy nem tartalmaznak képszámítást (ilyen például a később ismertetésre kerülő szabályozó), ez a probléma nem lép fel. Egy megoldás lehetne erre, ha a kezdő időlépcsőt a minimális képidőhöz igazítanánk. Csakhogy a mai játékprogramok gyakran 100 képet is megjeleníthetnek másodpercenként, ami tíz ezredmásodperces időlépcsőt jelent. Ez pedig túl rövid ahhoz, hogy iteratívan finomító algoritmust indítsunk belőle.

A megoldást tehát a fix, a képidőnél rövidebb lépcsőket használó időosztás használata. Ennek megvan az az előnye is, hogy így az elágazási tényezőt (a cselekvésszámot) is lejjebb vehetjük.

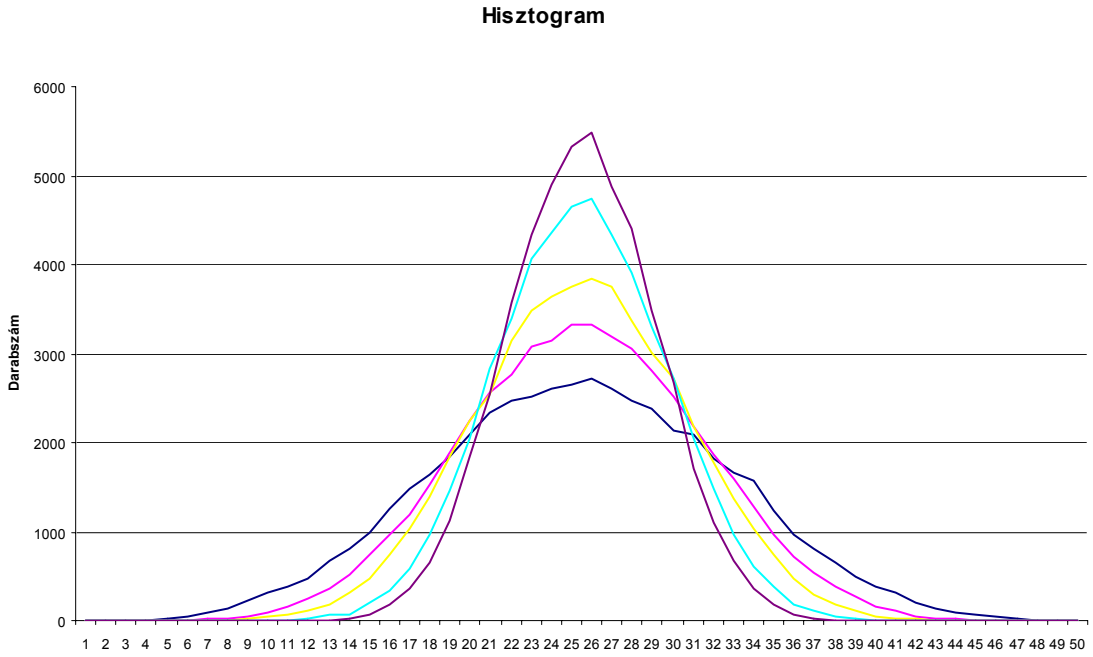
Véletlen cselekvéskészlet

A továbbiakban csak a véletlenszerűen választott cselekvésekkel foglalkoztunk, mivel ezek alkalmazásához kell a legkevesebb emberi beavatkozás. Az ilyen akciók ráadásul egymástól teljesen függetlenek, és az egyes elemek sem függenek a teljes készlet számosságától. Ez azt jelenti, hogy igény szerint lehet újabb cselekvéseket generálni, vagy kevésbé hasznosakat törölni a listánkból – vagyis nem szükséges előre generált készletet fenntartani, hanem elég a keresés során minden egyes lépésben legenerálni néhány akciót. Ezzel a módszerrel az eljárás jobban skálázhatóvá válik, mivel így ha tudjuk, hogy mennyi időnk van egy-egy lépésre, akkor ehhez tudjuk igazítani a kipróbált cselekvések számát, vagyis az elágazási tényezőt. Ha viszont az alkalmazás nem időkritikus, akkor egy lépésenkénti hasznosságnövekedést is előírhatunk: egyszerűen addig generálunk új cselekvéseket, amíg valamelyiknek a hasznossági értéke el nem éri az adott határértéket. Természetesen semmi nem garantálja, hogy mindig találunk ilyen akciót, ezért célszerű a kipróbált cselekvések számára is valamilyen ésszerű felső határt szabni. A kipróbált akciók közül a legjobbakat eltároljuk, ezzel biztosítjuk, hogy az eljárás akkor is utat (ha nem is optimálist) találjon, ha zsákutcába téved.

Adaptív cselekvésválasztás

A tapasztalatok azt mutatják, hogy a kipróbált cselekvéseknek csak igen kis része mondható hasznosság szempontjából elfogadhatónak. Ezek az akciók általában egymáshoz közeli pontoknak feleltethetők meg a cselekvéstérben. Mivel a rossz cselekvéseket úgysem fogjuk eltárolni, ezért a hatékonyság szempontjából jó lenne, ha ezeknek a számát minimálisra lehetne csökkenteni – vagyis jó lenne a véletlent valamilyen módon irányítani. Ez nem túl bonyolult feladat, hiszen csak az eloszlásfüggvényt kell megváltoztatni. Mi a jelfeldolgozásban közismert Gauss-zajt [DSP] választottuk, egyrészt azért, mert az eloszlásfüggvény képletében a szórás és a várható érték is egyszerűen megadható, másrészt viszont van egy egyszerű közelítése, ami bár nem túl pontos, de a mi céljainknak megfelel. Egyszerűen összeadunk N darab [0..1)-en egyenletes eloszlású véletlen számot, majd kivonunk belőle N/2-t. Ez a központi határeloszlás tétele értelmében Gauss-eloszláshoz tart. A

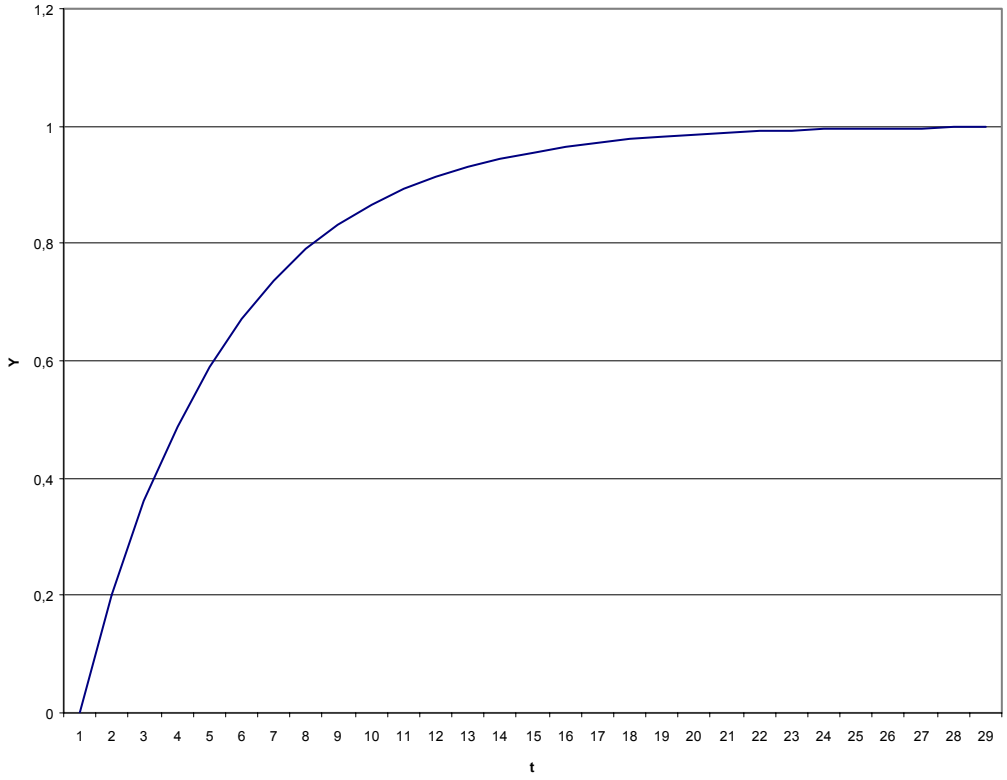
18. ábrán a hisztogramok láthatók az N függvényében, a leglaposabb görbéhez 4, majd a többihez 6, 8, 12, 16-os értékek tartoznak, megfelelően. Látható, hogy az N változtatásával a szórás szépen szabályozható.



18. ábra - A Gauss-zaj közelítésének hisztogramjai N függvényében

$$Y[k] = (1 - Q) \cdot Y[k-1] + Q \cdot X[k] \qquad (1)$$

Az adaptív cselekvésválasztás lényege a következő: miután letároltuk az új gráfpontokat, az eloszlásfüggvény középpontját exponenciális átlagolással (amelynek a rekurzív képlete (1)) a legjobb cselekvés felé toljuk el. Kiszámítjuk a középpont és a legjobb cselekvés távolságát, és ha ez az előző gráfpont generálása óta csökkent, akkor a szórást valószínűleg csökkenthetjük. Ha azonban a távolság nőtt, az azt jelentheti, hogy a cselekvéseink kezdenek eltávolodni a középponttól, vagyis, ha nem igazítunk az eloszlásfüggvényen, akkor a rossz kipróbált akciók aránya növekedni fog. Mivel a cselekvésválasztás véletlenszerű (zajos), ezért csak akkor korrigálunk a szóráson, ha a távolságkülönbség elér egy adott küszöbértéket. Erre azért is szükség van, mert a közelítő eljárásban nem tudjuk a szórást folytonosan változtatni.



19. ábra - Az exponenciális átlagolás válasza az egységugrásra (Q=0,2)

Ennek a módszernek két problémája van. Az első, hogy az átlagolás miatt a középpont viszonylag lassan követi a legjobb cselekvést, vagyis a keresés elindítása után eltelik egy kevés idő, amíg a hatékonyság elfogadható szintre növekszik. Ezen persze lehet segíteni, például azzal, hogy a kezdőpontra többször is lefuttatjuk a keresést, de nem tárolunk el egy új pontot sem. Ezzel gyakorlatilag előre beállítjuk az eloszlásfüggvényt.

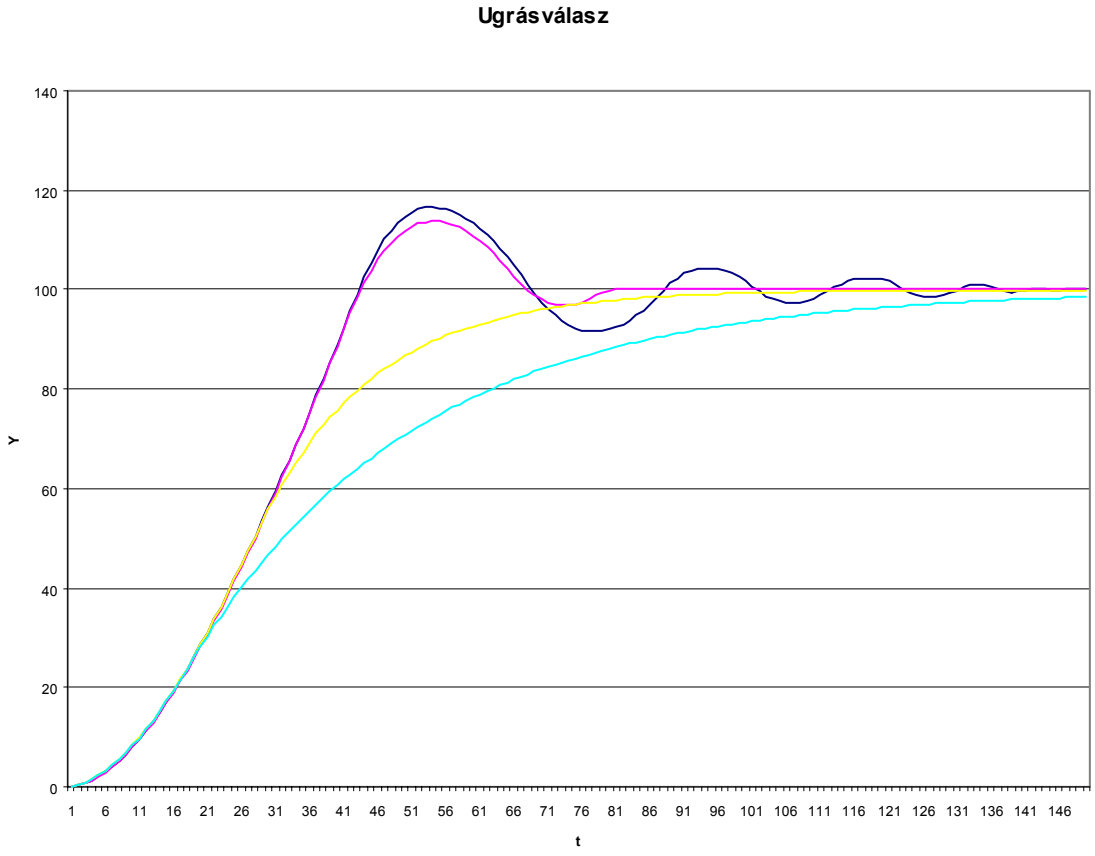
A másik gond az, hogy előfordulhat olyan eset, hogy az egymást követő cselekvések nagyon eltérőek lehetnek. Ez azt jelenti, hogy a cselekvésváltozáshoz rendelhető frekvencia elég nagy ahhoz, hogy az exponenciális átlagolás (mint aluláteresztő szűrő) kiszűrje – vagyis az eloszlásfüggvény nem fogja tudni követni ezeket a változásokat. Ezt a problémát meg lehet oldani az időlépés további csökkentésével, de sokat segíthet az is, ha az eloszlásfüggvényt „szélesebbre” vesszük, azaz a szórást növeljük. Ezzel viszont vigyázni kell, mert a túl széles görbe a hatékonyságot is csökkentheti.

A mohó algoritmus

Az eljárás ebben a formában már használható, de van néhány olyan fogás, amivel még jobbá tehető: például ha előre ismerjük a keresési teret, és tudjuk, hogy nincsenek benne zsákutcák, akkor a bonyolult keresési módszerek helyett real-time mohó algoritmust is alkalmazhatunk. Ennek megvan az az előnye, hogy a cél kijelölése után rögtön használható (követhető) útvonalszakaszokat kapunk eredményül – vagyis keresés közben már elkezdhetünk végigmenni az „eredményen". Ezzel az eljárással egy egyszerű, de annál hatékonyabb szabályozókört hozhatunk létre, amelynek viselkedése a cselekvéskészlettel és a választott időlépéssel jól befolyásolható.

Mi egy egydimenziós esetet vizsgáltunk. A szabályozás célja, hogy a célponttól való távolság (és a sebességet is) minimalizálja. Ez utóbbi tényező súlyával az algoritmus „előretekintését" lehet beállítani, vagyis nem csak az adott pillanatbeli távolságot vizsgáljuk, hanem azt is, hogy ha a sebesség nem változik, akkor adott idő múlva milyen távol leszünk a céltól. Ha ennek a paraméternek megfelelő értéket adunk, akkor nem akadálymentes tér is bejárhatóvá tehető mohó algoritmussal, csak bele kell integrálni az állapotok hasznosságát számító eljárásba azt, hogy automatikusan kizárja az olyan állapotokat, amelyek akadályban vannak – és azokat is, amelyekben (ha a sebesség nem változik), az ágens az előretekintésnek megfelelő idő múlva érne a falhoz. Ha a „zsákutcák" méreteit is ismerjük, akkor „többpontos" előrenézéssel elméletileg ezeknek a kikerülése is megoldható, azonban ennek a megvalósításához a keresési tér részletes ismerete szükséges, ezért ezzel a továbbiakban nem foglalkozunk.

Az alábbi ábrán a szabályozó egységugrásra adott válasza látható. A leglaposabb görbéhez 2 másodperc, a következőkhöz pedig rendre 1 másodperc, 100 ezredmásodperc és 10 ezredmásodperc „előretekintési" idő tartozik.



20. ábra - A szabályozó ugrásválasza az "előretekintés" függvényében

A mohó algoritmus előnye, hogy a gráfpontok listájára nincs szükség, vagyis minden lépésben elég csak addig generálni cselekvéseket, amíg egy megfelelő hasznosságút nem találunk, majd azt végrehajtjuk. Így a keresés sebessége növelhető. Természetesen a mohó eljárásnál is lehet alkalmazni az eloszlásfüggvény „előkészítését" a tényleges keresés megkezdése előtt. Ez egy fix reakcióidő hozzáadásával ekvivalens – azonban ez az időtartam még mindig kevesebb, mintha nem mohó (pl. A*) algoritmust alkalmaznánk.

Elsőre talán nem triviális, hogy mire lehet egy mohó algoritmusos szabályozót használni egy játékprogramban. Mi ennek a segítségével oldottuk meg az ágensek célzását a kiszemelt játékprogramban. A cél nem a nagy pontosság, hanem az emberek minél jobb utánzása volt, amire először egy PID-kontrollert alkalmaztunk, de ennek a paraméterezésével komoly gondjaink voltak, ráadásul igen gyakran instabillá is vált. Ezzel szemben ennek a módszernek nincsenek ilyen hátrányai, ráadásul kevesebb változtatható paramétere van, amelyekkel a működés azonban jól szabályozható. Külön előny, hogy ezek a paraméterek könnyen

értelmezhető fizikai tartalommal bírnak, ami a beállításukat könnyíti – a mi esetünkben ezek a változók a súrlódás, egy tömeg illetve az előretekintési idő voltak.

Továbbfejlesztési, alkalmazási lehetőségek

A cselekvéstéri keresés az előző fejezetben bemutatott dinamikus gráfépítéssel kombinálva egy robot tervezését jelentősen megkönnyítheti. Ehhez csak a szomszédkiválasztást kell úgy módosítani, hogy a gráfpontokat lehetséges cselekvések kipróbálásával határozza meg – így a keresés lefutása után a robotnak mindössze az élekhez rendelt „mozgatóparancsokat" kell végrehajtania. Ezek pedig akár közvetlenül a motorokba vezethető jelek is lehetnek. A valóságban azért nem ilyen szép a helyzet, ugyanis az időlépést az adaptív cselekvésválasztás miatt célszerű kis értéken tartani, vagyis ha az élekhez egy időlépést rendelünk, akkor a gráfpontok túl közel lennének egymáshoz. Egy valódi robot esetében azonban ez nem okoz túl nagy gondot, mert a működés során az idő nincs úgy kvantálva, mint egy szimulátor, vagy egy játékprogram esetén – vagyis nincsen az a bizonyos, sok problémát okozó képszámítási idő. Ez azt jelenti, hogy az időlépések tetszőleges (akár egymástól eltérő) értékeket is felvehetnek, ami lehetővé teszi, hogy – kellő számítási teljesítmény birtokában – akár az iteratívan finomító algoritmusokat is alkalmazzuk.

A cselekvéstéri keresés jó kiegészítésként szolgálhat az első fejezetben bemutatott útkereséshez. Bár a mai processzorteljesítmények nem teszik lehetővé, hogy játékprogramokban komplex feladatok megoldására alkalmazzuk, azonban néhány apró trükk bevetésével (mint pl. az adaptív cselekvésválasztás) kisebb problémákra megoldást nyújthat. Nem időkritikus alkalmazásokban, mint amilyenek például a modellezőprogramok, mindenféle gyorsítás nélkül is felhasználhatók. Természetesen, ha nem okoz problémát, hogy a robotnak hosszú reakcióideje van, akkor az eljárást nyugodtan felhasználhatjuk a robotikában is.

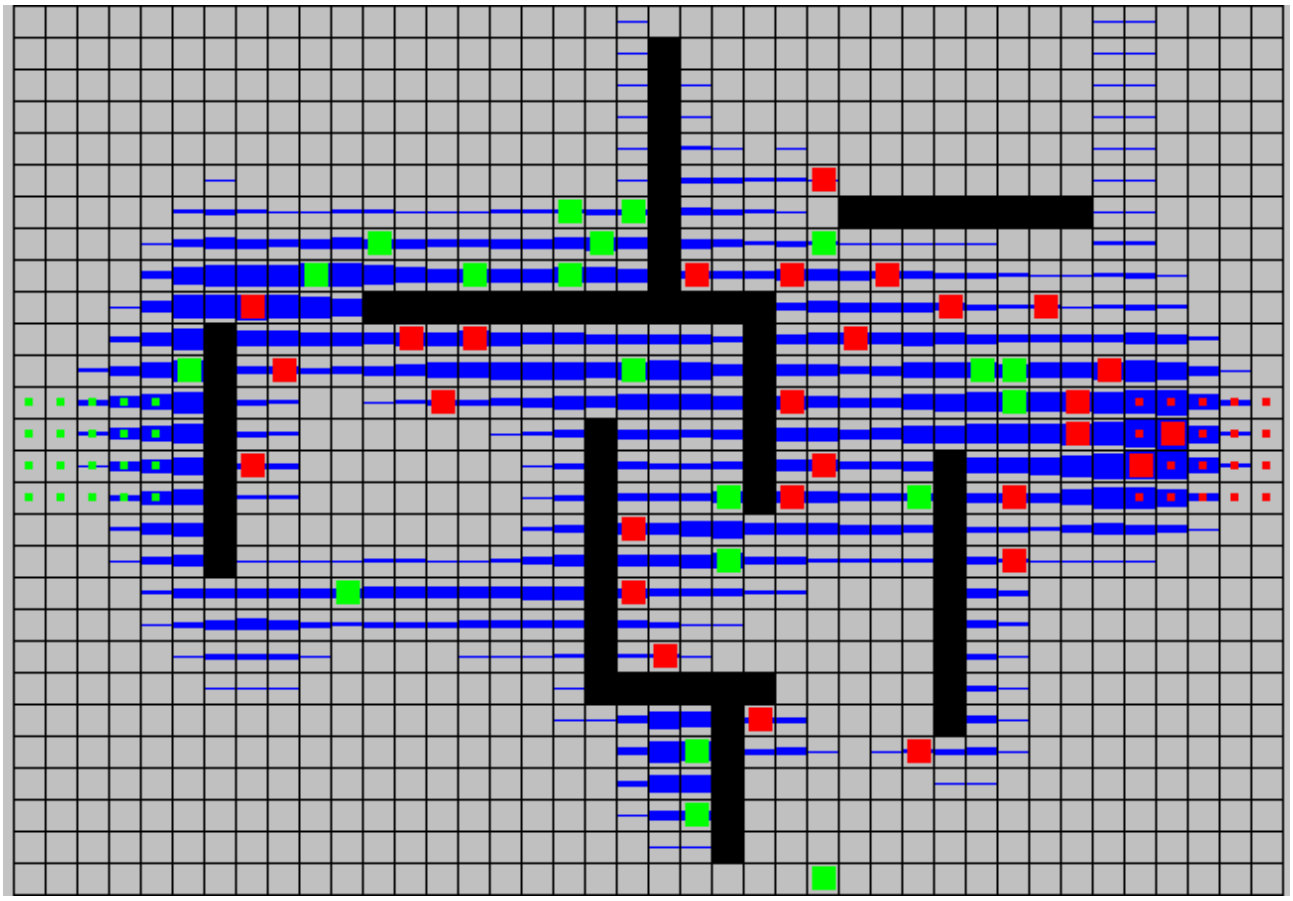
A cselekvéstéri keresés nem csak a robotikában, hanem a számítógépes modellezésben és a karakteranimációban is alkalmazható. A mai modellezőprogramok (pl. 3D Studio MAX [3DS]) többnyire inverz kinematikát [Elias] használnak a „fogd meg a kilincset" jellegű problémák megoldására. Ennek az eljárásnak azonban nagy hátránya a pontatlanság és az,

hogy az eredmény gyakran nem valószerű. Az inverz kinematika ugyanis egy egyszerű „csontvázszerű” modell (ún. kapcsolt hierarchia) segítségével dolgozik. A modell egyes elemeinek a célpontját lehet kijelölni. Minden egyes „ízülethez” (a csontok csatlakozási pontjaihoz) megadhatunk ún. kényszereket (constraints), amelyek a különböző irányokban való forgatásokat korlátozzák. Az eljárás azonban nem veszi figyelembe a „csontok” tömegét, valamint az izmok erőkifejtési képességét, ami a cselekvéstéri kereséssel elvileg megoldható. Erre a problémára a cselekvéstéri keresés megoldást nyújthat.

Az "ant search" (hangya alapú keresés)

A hangyák által a természetben alkalmazott keresési algoritmust már sokan megpróbálták különböző szimulációs módszerek segítésével felhasználni. A felhasznált irodalomban szereplő publikációk ([Botee99], [Levine02], [Kaegi03], [Gordon], [Reimann02], [Fujita02], valamint [Ants02] és [ACO] sok publikációja.) azonban egy viszonylag kis gráfban keres utat az „ant search” segítségével. (Ilyen feladat például az utazóügynök probléma, melyre a [Botee99] cikk is megoldást keres.) A mi alkalmazásunkkal ellentétben, a fenti esetekben az egyes gráfpontok mind lehetséges célpontok, melyek között az utak költségének a minimalizálása a cél. Célját tekintve [Kroon02] a közlekedési torlódások figyelésével részben a miénkhez hasonló célt tűzött ki, ám az ő alkalmazásának célja - mint ahogy azt a továbbiakban látni fogjuk - még mindig jelentősen eltér a mi algoritmusunkétól.

A fentiektől lényegesen eltér az a megközelítés, hogy a hangyák módszerét egy olyan keresési térben alkalmazzuk, melyben az egyes célpontok között nagyon sok gráfponton vezethet át az út. A szemléletes bemutatáshoz tételezzünk fel egy négyzetárcsos térképet, melyen időről időre falakba ütközhet az ember. Az általunk kifejlesztett algoritmus a térkép távolabbi pontjai közötti útvonaltervezést hivatott elvégezni, miközben az akadályok között vezető utak általában szélesek, így (mivel minden egyes négyzetrács egy gráfpontnak felel meg), a keresési térben igen nagy az élek száma, és magát a végeredményt is „széles” utak képezik.



21. ábra - Térkép, melyet gráfként kezelve nagyon sok élt kapunk.
(A jelmagyarázat a demonstrációs program leírásában szerepel.)

Többek között az A* algoritmus is alkalmas lehet arra, hogy ilyen feladatokat viszonylag gyorsan megoldjon, de a mi algoritmusunk - mint azt a továbbiakban látni fogjuk - egyéb hasznos információkat is szolgáltat. Az útvonalat szélesebb sávban határozza meg (részben a térképen rendelkezésre álló hely függvényében), emellett pedig például stratégiai játékok számára hasznos stratégiai információkat is szolgáltat.

Ahogyan a hangyák a legrövidebb utat keresik és követik

A hangyák keresési módszerének különlegessége, hogy az egyes egyedeknek nem kell ismerniük az egész térképet, csak a közvetlen környezetüket. Ennek ellenére az igen egyszerű kommunikáció segítségével idővel globálisan is jó megoldást találnak. (A globálisan jó megoldás itt azt jelenti, hogy valószínűleg nincsen jelentősen kisebb költségű megoldás. Ez sok alkalmazásban elegendő.)

A hangyák kezdetben véletlenszerűen barangolnak és megpróbálnak élelmet találni. Mozgásuk során egy bizonyos feromon nevű hormonnal nyomot húznak maguk után, így ha célhoz értek, ezt a nyomot követve visszajutnak a bolyhoz. Ha egy hangya feromonnyomot talál, akkor nagy valószínűséggel elkezdi követni azt. Ha tehát egy hangyának sikerül élelmet találnia, akkor a többi arra járó jó eséllyel szintén meg fogja találni, mivel a nyom odavezeti őket. A boly és az élelemforrás között ingázva az arra járók egyre jobban „összekenik" az útvonalat feromonnal. Ennek következtében a nyom egyre erősebben irányítja majd a többieket is, vagyis egyre többen mennek majd a helyes irányba.

Ha két hangya eltérő útvonalon jut el az élelemforráshoz, akkor az, amelyik a rövidebb utat találta, ugyanannyi idő alatt többször fordul majd a boly és a cél között, mint a másik. Ekkor viszont a saját útján a feromonnyom is erősebb lesz, mivel többször megy rajta végig. Ha az egyik nyom jelentősen erősebb lesz, mint a másik, akkor a másikon járó hangya is át fog térni a rövidebb útra, mivel azt a nyomot erősebben fogja érezni, mint a sajátját.

Ez a gondolatmenet kiterjeszthető sok hangyára is, így a kezdeti véletlenszerű bolyongások után kialakulnak azok az útvonalak, melyeket a jelentős részük használ.

A természetes módszer szimulációja

Ahhoz, hogy a fenti módszert a gyakorlatban fel tudjuk használni, szimulálnunk kell a hangyák viselkedését a „térképen". A kereséshez sok hangya kell, mivel a kezdeti bolyongások során több gyorsabban megtalálja a célpontokat. (A nagy hangyaszámnak ezen kívül még több oka is van, melyeket később részletezünk.) Sok hangya hatékony szimulációjának előfeltétele, hogy egy hangya "működése" egyszerű legyen. A nagy kérdés az, hogy mit kell tudnia egy hangyának ahhoz, hogy gyorsan lehessen szimulálni, de együttesen gyorsan találjanak egy jó megoldást.

A hangyák az általunk kifejlesztett algoritmusban 6 alapvető szabály alapján mozognak a négyzetrácsos, síkbeli térképen. A további, kiegészítő szabályok célja a keresés gyorsítása és a lokális maximumok által okozott nehézségek megoldása.

Az alapvető szabályok a következők:

- Minden hangya egyszerre egy négyzetrácsnyit tud lépni a 4 szomszédos mezőre.

- Minden hangya minden lépésben feromonnyomot (nyomvonalat) húz maga után.
- A hangyák a továbbhaladási irányukat a környezetükben érzékelt nyomvonalak erősségének függvényében, a véletlen bevonásával választják. (Annak érdekében, hogy néha a kedvezőtlen irányokba is forduljon a hangya, a legjobb irányt csak egy bizonyos valószínűséggel követi. Ellenkező esetben egyenlő valószínűséggel, véletlenszerűen választja ki az irányt.)
- A hangyák előnyben részesítik az egyenes vonalú mozgást.
- A hangyák nem szeretnek hátrafordulni, vagyis a visszafelé mutató irányt kisebb valószínűséggel választják.
- Természetesen a hangyák nem mehetnek át az akadályokat jelentő falakon.

(Arra, hogy a hangyák átmászhatnak-e egymáson, később térünk vissza.)

A fenti szabályok egy része a természetes módszerből következik (nyomvonal húzása és követése, falak áthatolhatatlansága), míg a többi a keresés sebességét hivatott növelni: a gyakori fordulás (különösen a hátrafordulás) lassítja az előrehaladást és növeli a lokális maximumok (maximális nyomvonalszint) környékén előforduló „beragadás” esélyét is. (Beragadás alatt itt azt értjük, amikor a nyomvonal erősségének lokális maximumától nehezen tudnak eltávolodni a hangyák, viszont egyre többen odagyűlnek.)

A szimuláció másik oldala a környezet viselkedésének szimulációja. Ide a térkép folyamatos frissítésén kívül a nyomvonalak (általában a hormonok) párolgásának szimulációja tarozik. Bolyongásuk során a hangyák sokszor olyan útvonalon haladnak, melyet többször nem is tesznek meg, mivel találnak egy jobbat. Az ilyen nyomoknak idővel el kell tűnniük, mivel csak zavarják az előnyösebb utak nyomvonalak alapján történő megtalálását és követését. (Az előnyös utat keresztező, gyengébb nyomvonallal jelzett mellékutak számos hangyát eltéríthetnek, így lassítva az algoritmus működését.)

A bővítési lehetőségek

A bővítések során figyelembe kell venni azt a fontos feltételt, hogy az egyes hangyák képességeit úgy növeljük, hogy a keresés sebessége ne amiatt romoljon el, hogy a hangyák ugyan „intelligensebbek”, de a szimulációjuk aránytalanul sok erőforrást igényel. (Itt a

keresés sebességén a megoldás megtalálásának sebességét értjük.) Előfordulhat ugyanis, hogy egy nagy bővítés emiatt mégsem jelent igazi előrelépést.

Az első és meglehetősen triviális bővítési lehetőség az, hogy egy kicsit segítünk a hangyáknak a célpont megkeresésében. Ennek érdekében két új „hormont" vezetünk be, melyek közül az egyik a céltól (élelemforrás), a másik a bolytól indulva radiálisan gyengül. Ha a haladási irány kiválasztásában ennek kicsi a jelentősége, akkor a hangyákat nem befolyásolja a megtalált nyomok következésében, viszont segít nekik akkor, ha nem látnak nyomot (főleg a kezdeti bolyongáskor), amikor ennek hiányában több, azonos esélyű irány lenne. (Fontos tehát, hogy a nyomvonalat, amin haladnak (amennyiben éppen nyomvonalon haladnak), csak a cél irányát mutató hormon miatt ne hagyják el.)

Az eddig vázolt módszer vizsgálata során hamar kiderül, hogy a hatékony működéshez ez még kevés. A hangyákat vezérlő szabályok bővítésre szorulnak, mivel külső beavatkozás hiányában nagyon könnyen saját csapdájukba esnek.

A legfontosabb probléma a következő: Ha sok hangya van egy kisebb területen, akkor jó eséllyel beszorulnak, mivel folyton egymásnak fognak ütközni. Mivel nem jutnak messzire, ezért egyre jobban megerősítik a nyomot maguk alatt, tovább csökkentve a távozás esélyét. Erre több megoldási lehetőség is van:

- A hangyák csak akkor húzzanak nyomvonalat, ha haladtak is! Amennyiben egyik irányba sem tudnak menni, akkor ne erősítsék tovább a nyomot. Sajnos ez a megoldás nem hatékony, mivel még nagy tömörülés esetén is viszonylag ritka, hogy egyáltalán ne tudjanak lépni. Maximum annyit érnénk el, hogy a tömeg szélén erősödne csak a nyom, ami miatt a hangyák egy kis gyűrű alakba tömörülnének, ami aztán a valószínűségi alapú mozgás és a többi hangya miatt blokkolt irányok miatt idővel visszaalakulna a kiindulási alakzatba.
- A hangyák mászhassanak át egymáson! Ennek a szabálynak a beiktatása bizonyos helyzetekben (a keresés bizonyos fázisaiban) hasznos lesz, de ezekre később térünk ki. A jelenlegi problémára van jobb megoldás is, mivel ha egy zsákutcába tömörülnek össze, akkor az csak ront a helyzeten, ha még egymásra is tudnak mászni. (Ha nem tudnak, akkor elegendő hangya esetén előbb-utóbb megtelik a zsákutca.)
- Amennyiben a hangyák társukkal ütköznek, bocsássanak ki egy olyan (a nyomvonaltól eltérő) hormont, melyet a továbbiakban igyekeznek elkerülni! Így kerülni fogják a tömörüléseket, valamint ha egy lokális maximumot jelentő zsákutcába kerülnek, azt is

könnyebben elhagyják, ha ott tömeg alakul ki. Ez az a megoldás, mely végül elegendően hatékonynak bizonyult. Ez a hormon a Stacked Canyon („torlódási hormon”) nevet kapta.

Mivel a torlódási hormon jelenléte csak átmeneti problémát jelöl, ezért ez a hormon gyorsan kell, hogy párologjon. Mivel ugyanis a hangyák elkerülik az ilyen területeket, ezért a torlódás hamar megszűnik. Utána a terület természetesen újra gond nélkül járható, ráadásul könnyen előfordulhat, hogy fontos útvonal részét képezi. Amennyiben a torlódás mégis tartósan fennállna (például mert a cél helyét jelző hormont követő hangyák zsákutcába kerülnek), a hormon szintje sem fog lecsökkenni a párolgás miatt, mert a hangyák folyamatosan pótolják.

A Stacked Canyon egy a valós idejű stratégiai játékokkal foglalkozó szakirodalomban ([Pottinger99]) előforduló kifejezés. Olyan helyzetekre utal, amikor a játéktéren mozgó objektumok egy szűkösebb átjáró (például kanyon) bejáratánál összetorlódnak és mivel mindannyian befelé igyekeznek, végül egyáltalán nem, vagy csak nagyon lassan jutnak át. Erre a problémára egy lehetséges megoldást jelent ez a "hormon", mely nem engedni, hogy a hangyák túlságosan összetorlódjanak.

Paraméterezés, statisztikák és további bővítések

Az elemzések során kiderült, hogy az egyes lépések útirányválasztásához elegendőek a fenti szabályok, viszont szükség van még egy olyan szintre, mely a hangyák működési paramétereit (például környezeti jellemzők súlyozása az irányválasztásban)

- a hangyák szintjén
- globális szinten

módosítani képes.

A szükséges módosításokat mindkét esetben egyszerű mérések és statisztikák alapján meg lehet határozni. Ezek a módosítások azért szükségesek, mert a különböző keresési terekben más beállítások mellett hatékony ez a keresési algoritmus. Ha pedig nem jók a beállítások az adott helyzetben, az a fent említett mérésekkel és statisztikákkal gyorsan kimutatható.

A hangyák által egyénileg készített mérések és felhasználásuk

Az egyes hangyák három mennyiség változását követik nyomon:

- elmozdulás: Minden hangya kiszámolja, hogy n lépés alatt milyen messzire jutott el. (A jelenlegi példa-implementációban erre a célra a Manhattan-távolságot alkalmazzák, n=30.)
- hormonszint változása: Az aktuális céltól távolodva radiálisan gyengülő (így a cél irányát mutató) hormon mennyiségének a változása n lépés alatt.
- lépésszám: A paraméterek módosításának szükségességét a legutolsó célpont elérése óta megtett lépések számának vizsgálatával is el lehet dönteni, így a hangyák ezt is nyilvántartják.

Amikor egy hangya lokális maximumba kerül (vagyis olyan zsákutcába, melyből az aktuális paraméterezése mellett csak kis eséllyel jön ki elsősorban a cél felé irányító hormon miatt), akkor a fenti mennyiségek mindegyike alkalmasnak tűnik a helyzet felismerésére. A gyakorlati elemzések azt mutatják, hogy az elmozdulás és a hormonszint változása mégsem annyira előnyös erre a célra, mert a hangyák alapvetően zajos mozgása miatt nehéz meghatározni egy olyan korlátot, amely alatti értékek esetén zsákutcára gyanakodhatunk. (A korlát optimális értékének meghatározása már egy konkrét feladat esetén is nehéz.) A harmadik mennyiség, vagyis a legutóbbi célpont (jelen esetben az élelemforrás vagy a hangyaboly) elérése óta megtett lépések száma tűnik a legalkalmasabbnak a fenti ellenőrzéshez.

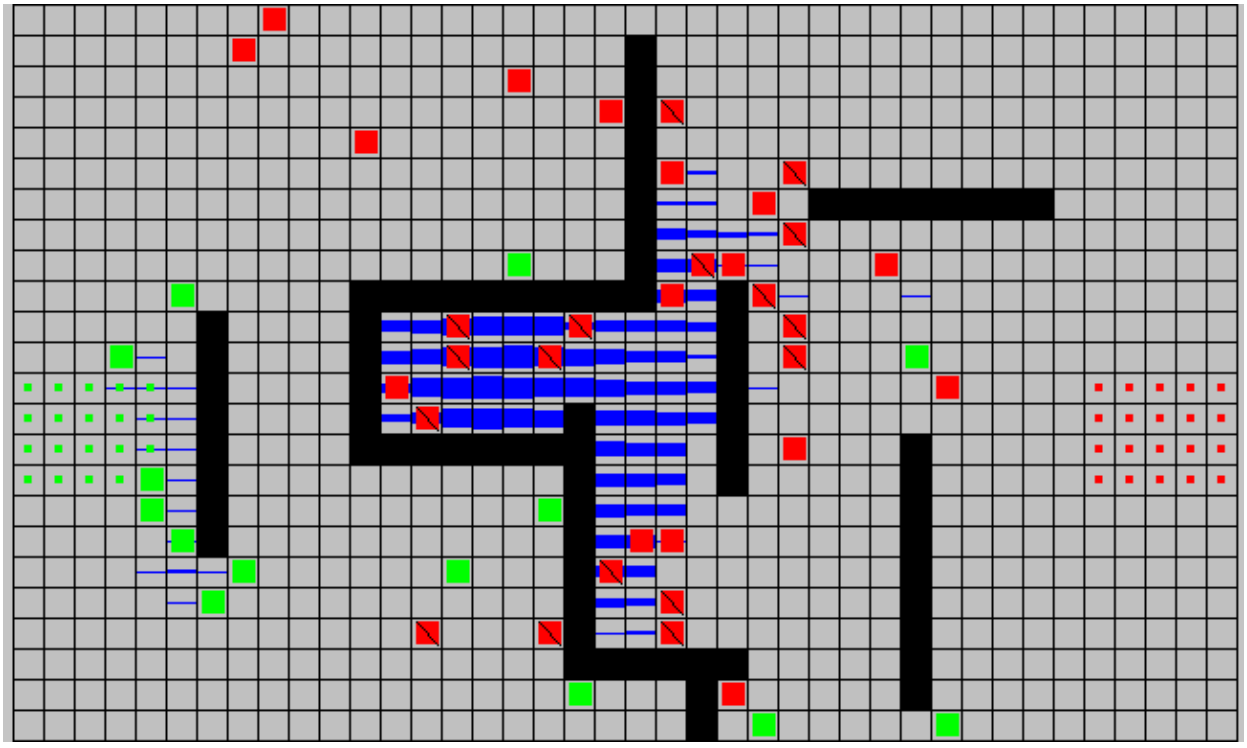
Amennyiben egy hangya egy bizonyos lépésszám alatt nem éri el a célját, akkor valószínűleg vagy hosszabb a két célpont közötti út, vagy rossz irányba ment, esetleg lokális maximumba került.

- Ha nagyobb az út hossza, mint a lépésszámkorlát: Ez az eset akkor áll fenn, ha a korlátot túl kicsire választottuk meg. Ezen könnyen lehet segíteni, mert ha a szimuláció során nem lesz olyan hangya, aki ennyi lépés alatt átér a bolytól az élelemforrásig vagy vissza, akkor növelni kell ezt az értéket. (A folyamatosan frissített statisztikák között megtaláljuk az ilyen esetek számát, de erre később térünk ki.)
- Ha rossz irányba ment, akkor a hasonló a helyzet ahhoz, mint hogyha lokális maximumba került volna.

- Ha lokális maximumba került, akkor onnan valahogy ki kell juttatni. Ennek lehetőségeit tekintjük át a következőkben.

Ha egy hangya jó eséllyel lokális maximumba került, akkor számos dolgot lehet tenni. Az általunk vizsgált megoldási lehetőségek:

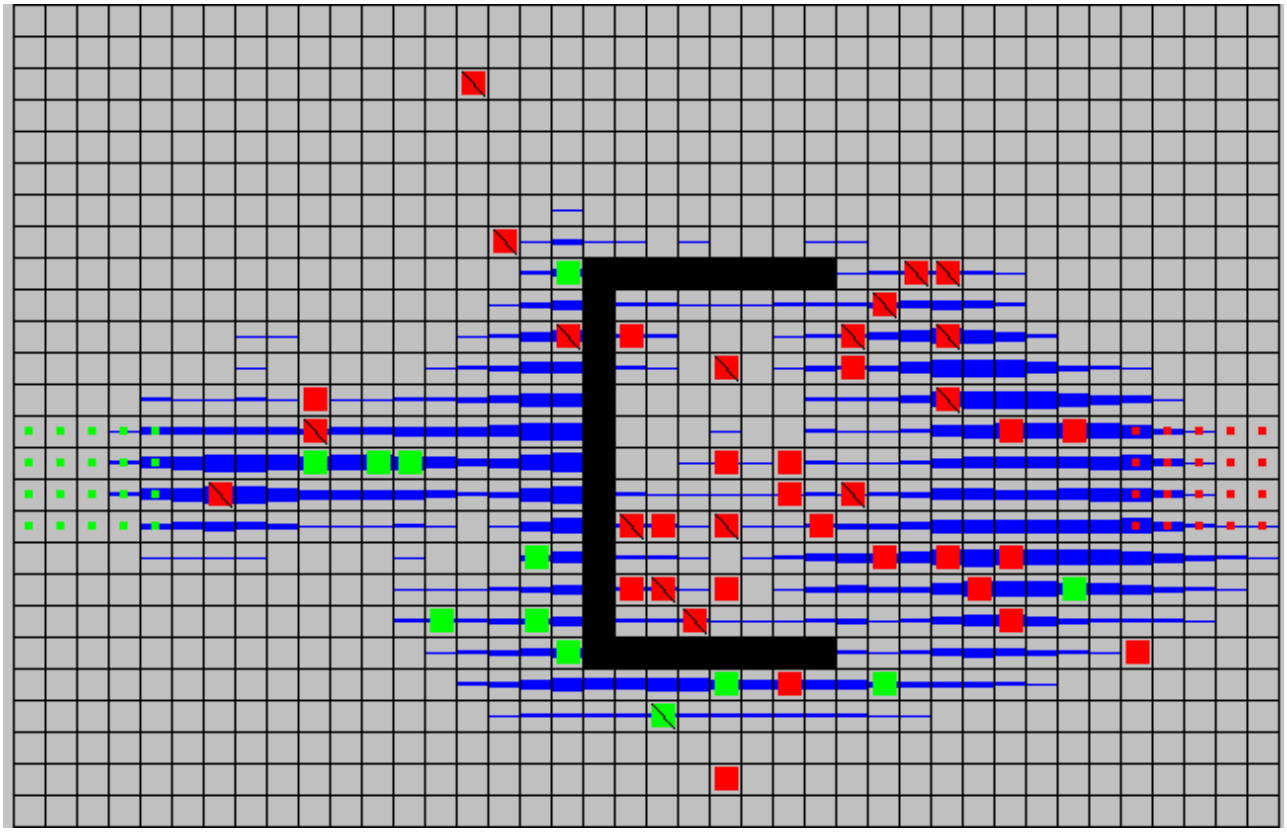
- Nem teszünk semmit. A hangyák iránykiválasztásában a véletlen is szerepet játszik, így bízhatunk abban, hogy előbb-utóbb kijut a zsákutcából. Ha elegendően sok hangya van a keresési térben, akkor a helyes út közelében lévő zsákutcák hamar „megtelnek” hangyákkal (feltéve, hogy egymáson nem mehetnek át), így előbb-utóbb több már nem fér be. Ezt a folyamatot a torlódási (vagy Stacked Canyon) hormon is elősegíti, mert csökkenti a hangyák "sűrűségét", így kevesebb hangya is elég ahhoz, hogy a többiek már kis valószínűséggel menjenek be a zsákutcába.



22. ábra - A zsákutcában sok torlódási hormon halmozódik fel.

- Ha egy hangya a megadott lépésszám alatt nem jut el a célhoz, akkor a korlát átlépése után már nem húz nyomvonalat (feromonnyomot). Ez a megoldás kombinálható a későbbiekkel is. Ezzel át lehet hidalni azt a jelenséget, hogy a zsákutcákban nagyon

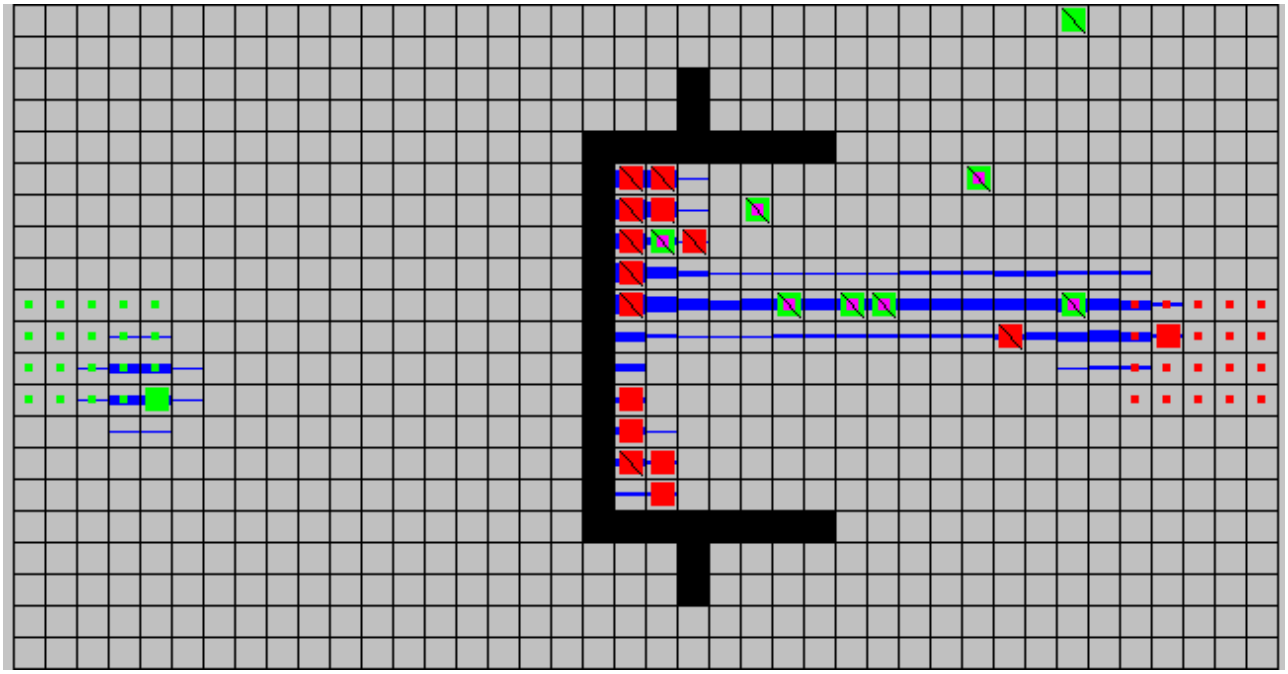
sok az erős nyomvonal (mivel a beragadt hangyák nagyon sokat mászkálnak kis területen), így a bekerülő hangyák a többieket is „odacsalják”.



23. ábra - A zsákutcába került hangyák egy idő után nem húznak nyomot.

A módszerek kombinálása esetén a nyomvonal húzásának kisebb lépésszámhatárt is meg lehet szabni, mint az egyéb beavatkozásoknak (például a következőnek leírt visszafordulásnak). Az implementációban a nyomvonal húzása már a visszafordulási korlát felénél befejeződik. Ez azért jó, mert a hangyáknak több ideje van átjutni a következő célponthoz, de ha egy kicsit is felmerül a téves út gyanúja, akkor már nem húznak nyomvonalat. Ezzel a megoldással az optimálisnál hosszabb utak száma is jelentősen csökkenthető.

- A lépésszámkorlát elérésekor a hangya visszafordul, vagyis ha eddig az élelemforrás felé ment, akkor a bolyt veszi célba illetve fordítva. (Tesztelési céllal a lépésszámkorlát elérése után az általunk készített program felajánlja a „bolyongás” lehetőségét is (ld. program leírása), de ez nem bizonyult hatékonynak.)



24. ábra - A zsákutcába került hangyák visszafordulnak.

Globális statisztikák és felhasználásuk

A szimulátor a hangyák mozgása során a következő statisztikákat készíti adott számú szimulációs lépésekből álló „ablakokra”. (Egy szimulációs lépésben minden hangya pontosan egy lépést tesz meg, amennyiben tud lépni.)

- Célponthoz érkező hangyák száma és ennek eloszlása a legutóbbi x ablakban. (Az implementációban x=20.)
- Azon esetek száma, amikor egy hangya az általa készített mérések alapján úgy döntött, hogy lokális maximumba került és ezért az előző részben leírtak szerint módosított a viselkedésén. (Ezt nevezzük önkalibrációnak.)
- A legutóbbi 1000 hangyalépésben (ami 1000 / „hangyák száma” szimulációs lépésnek fele meg) a nyomvonalon lépő hangyák aránya. (Ebből elsősorban azt lehet majd eldönteni, hogy a hangyák mennyire koncentrálódnak az utakra.)

Amennyiben huzamosabb ideig (az implementáció esetében 150-200 szimulációs lépés után) nagyon alacsony (10% alatt van) azon hangyák száma, akik célba értek (a fenti statisztika szerint), akkor az az útkeresés sikertelenségére utal. Ennek több oka lehet:

- Hosszú az út, ezért a hangyák nem húzzák elég sokáig a nyomvonalat az alacsony lépésszámkorlát miatt. Erre megoldás a korlát növelése.
- Sok a lokális maximum, vagy kevés van, de az nagy területű zsákutca és rengeteg hangya belekerült. Ha a lépésszámkorlát növelése nem segít a helyzeten, akkor ez az eset áll fenn.

Ha a statisztikák szerint a lokális maximumok akadályozzák az útvonal megtalálását, akkor a következő lépéseket lehet tenni:

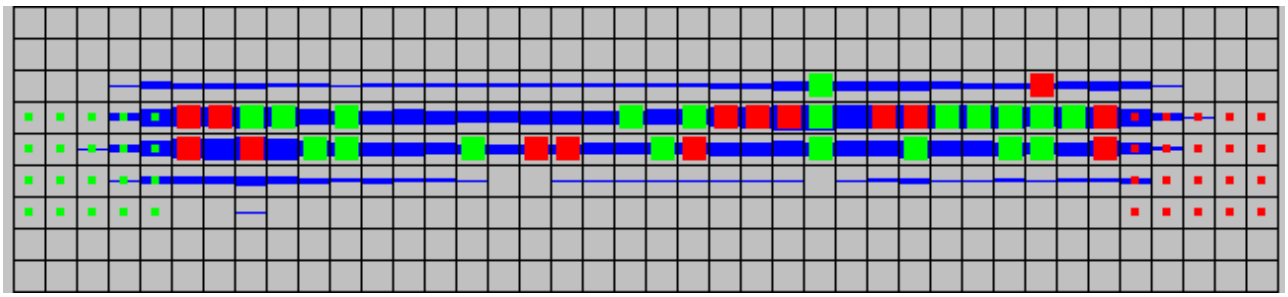
- Amennyiben a hangyák nem vennék figyelembe a torlódási hormon szintjét, akkor ezt érdemes bekapcsolni, hogy a zsákutcában csökkenjen a sűrűségük.
- Ha a hangyák átmehetnek egymáson, ezt meg lehet tiltani. (Ekkor nem tudnak annyira összetömörülni.)
- Meg lehet változtatni az irányválasztási súlyozást. Erre az implementáció több lehetőséget is biztosít, melyeket a program leírásában részletezünk.

Ilyen esetekben érdemes növelni a torlódási hormon negatív hatását és csökkenteni a célhormon súlyát. A nyomvonal követésének csökkentése is segíthet, de hosszabb ideig ez a hangyák teljes szétszóródását okozza a kereséséi térben. Rövid időre ez is hasznos. (A hangyák "szétszórását" akkor érdemes egy rövid időre bevetni, ha vagy más módszerek sikertelenek voltak, vagy pedig alternatív utakat is szeretnénk kerestetni.)

Ha az útkeresés a statisztikák alapján sikeres, akkor is előfordulhat, hogy a megoldás igen zajos. Ez azt jelenti, hogy a megtalált út széles, esetleg több utat is találtak a hangyák. (Útkeresési szempontból a megoldás alatt konkrétan a nyomvonaltérképet értjük.) A nyomvonaltérkép tisztítása érdekében a következőket érdemes elvégezni:

- Mivel ha a hangyák nem mászhatnak át egymáson, akkor a kikerülések miatt szélesebb lesz az út, ezért a tisztítás során érdemes engedélyezni az egymáson való átmászást.
- Ha a nyomvonalhúzási lépésszámkorlát magasabb a szükségesnél, akkor azok a hangyák is végig nyomot fognak húzni, akik nem teljesen a megtalált utat követték, hanem közben elkóboroltak, így nyomvonalmellékágakat hoztak létre. A korlát óvatos és lassú csökkentésével ez javítható. Ha a statisztika alapján a célba érő hangyák száma elkezd csökkenni, az azt jelenti, hogy a korlát már túl alacsony, így ekkor vissza kell állítani egy magasabb értékre.

- A hangyák irányválasztási súlyozása jelentősen befolyásolja, hogy mennyire szigorúan követik a nyomvonalat. Ha a torlódási hormon súlya alacsony, a nyomvonalé pedig magas, akkor a hangyák sokkal szorosabban követik a megtalált útvonalat még akkor is, ha sokan járnak rajta. (Ilyenkor ugyanis a torlódási hormon inkább csak szélesíti az utakat, amire ekkor már nincs szükség.)



25. ábra - A hangyák szigorúan követik a nyomot, átmászhatnak egymáson és egyáltalán nem veszik figyelembe a torlódási hormont.

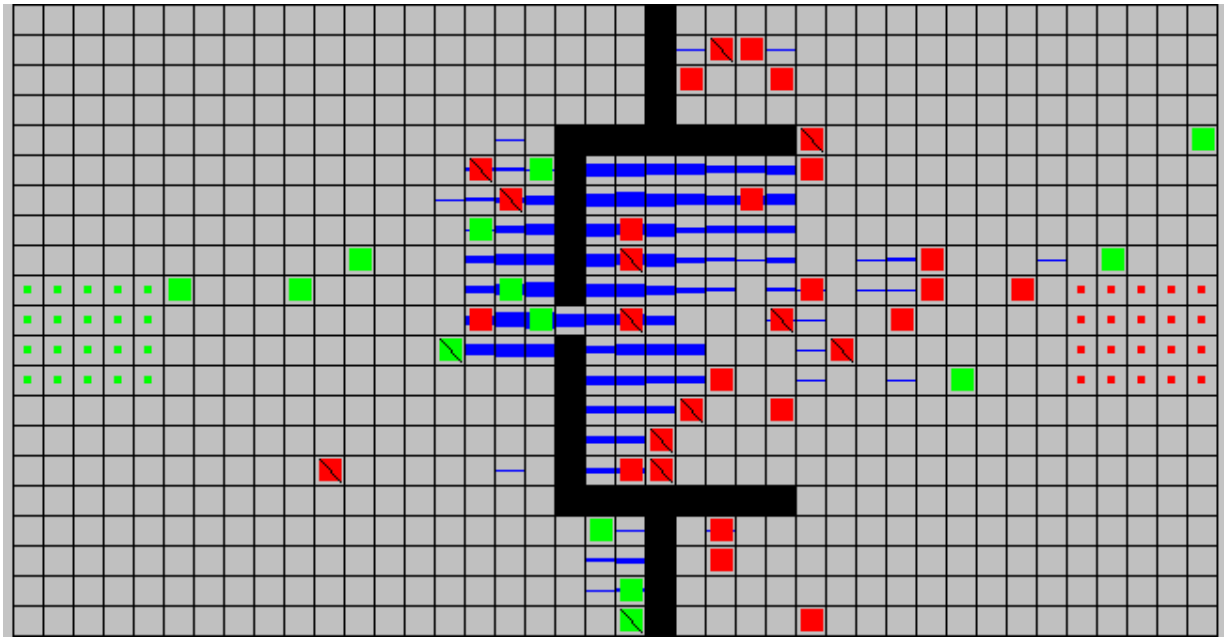
Az ant search ilyen megvalósítása esetében általánosan is elmondható, hogy az egyes paramétermódosítások hatása a statisztikákban lassan jelenik meg, így minden változtatás után eltart egy ideig, amíg az új statisztikai értékek stabilizálódnak. (A konkrét időigény a térképtől - főleg annak méretétől - függ. A 40×40-es térképen kb. 40-50 szimulációs lépés lehet szükséges a változások egyértelmű jelentkezéséhez.)

A szimuláció globális beállítása mellett (melyek a hangyák viselkedését határozzák meg) van még egy fontos paraméter, mégpedig a hangyák száma. Ezt egyrészt a feladat bonyolultsága (zsákutcák száma, útvonal hossza) határozza meg, másrészt ezzel lehet szabályozni a keresés erőforrásigényét. A hangyák száma akkor elegendő, ha a megtalált útvonalon a nyomvonal nem párolog el annyi idő alatt, amíg a következő hangya oda nem ér. Ellenkező esetben a hangyák elveszítik a nyomot, és nem stabilizálódik a megoldás, vagyis nem kezdenek el ezen az úton ingázni. (Programunkban a 40 × 40-es térképen 50 hangya az egyszerűbb feladatokra már elegendő.)

Felhasználás

Az ant search általunk vizsgált megvalósításának eredménye két hormon mennyiségi térképe. Az egyik a nyomvonal, mely kirajzolja a bolyként és élelemforrásként megadott két pont közötti utat. A másik a torlódási hormon térképe, mely azokat a területeket jelöli ki a keresési térben, ahol a hangyák sokszor feltorlódtak. Ennek a térképnek az olyan alkalmazások vehetik hasznát, mint például a stratégiai játékok.

A modern RTS (Realtime Strategy) játékok mesterséges intelligenciái nagyon gyakran használnak olyan módszereket az egyszerű útkeresésen kívül, melyek a játéktér egyéb fontos jellegzetességeit próbálják felkutatni. Ilyenek például a szűk átjárók és a stratégiai fontosságú helyek. Az ant search alkalmazásával egy ilyen játék amellett, hogy megkapja a számára fontos pontok közötti utakat a nyomvonaltérkép formájában, a torlódási hormon térkép ismeretében megtudhatja, hogy hol lehet kevés a hely csapatainak mozgatásához és melyek azok az átjárók, melyek nagy és kevés átjáróval összekötött területek között helyezkednek el.



26. ábra - Stratégiailag fontos helyek – torlódási (vagy Stacked Canyon) hormon

További stratégiai információk nyerhetők például úgy, hogy a nyomvonaltérképen bizonyos értéktartományokat kiszűrünk. Így kaphatjuk meg a bázis közvetlen környezetét a bejáratokkal együtt, vagy a bázis körüli, lépésszámban adott sugarú gyűrűt.

Mivel egy RTS játékban általában nem csak két pont között kell útvonalat tervezni, ezért az ant search-öt többször kell futtatni. A kettőnél több célpont közötti utak kereséséhez szükséges kiterjesztésről a további kutatási irányokkal foglalkozó fejezetben lesz szó. A bolyhoz tartozón kívül egyetlen cél felé vezető hormonnal több ponthoz vezető utak felkutatására nem alkalmas a módszer, mert a hangyák csak a legközelebbi célpontot fogják megtalálni. Ez abban az esetben hasznos, ha a feladat a legközelebbi lehetséges célpont felé vezető út megkeresése. (Például el kell jutni a legközelebbi kitermelőhelyhez, de hogy melyikhez, az mindegy.)

Az ant search egyik jelentős előnye, hogy egy további hormon felvételével képessé tehető arra, hogy a keresett útvonal bizonyos területeket a lehetőségekhez mérten kerüljön el. Ehhez mindössze egy olyan hormon kell, amit a hangyák inkább elkerülnek. Ez olyan megoldás, mint amivel az RTS játékokban „influence map" néven találkozhatunk. (Az influence map jelöli ki azokat a területeket, amiket az útvonalkereső algoritmusnak érdemes elkerülnie, vagy pont érintenie. Technikailag ezt általában a keresési gráf súlyainak változtatását jelenti.) Az ant search az ilyen hatásokra akár futása közben is képes reagálni, tehát akár a szimuláció futása közben is változhat a területek veszélyességi foka, az addigi eredmények nem vesznek el teljesen, csak attól kezdve a hangyák esetleg más utakat részesítenek előnyben.) Ezt a tulajdonságot nevezzük progresszivitásnak.

Ennek a keresési algoritmusnak egy másik fontos (és előnyös) tulajdonsága, hogy már akkor is szolgáltat valamennyi információt az eredményről, amikor az még nem végleges. Ha tehát például időhiány miatt azonnal el kell indulni a megtalált úton, akkor valószínűleg nem járunk rosszul, ha a keresés futása közben már elindulunk a nyomvonalon.

Az RTS játékok gyakran szembekerülnek a Stacked Canyon problémával, melyet az erről elnevezett hormon bemutatásánál leírtunk. Az ilyen veszélyekre az ant search eredménye fel tudja hívni a számítógép irányította játékos figyelmét, ami a formációk megtartása és a késések elkerülése miatt fontos stratégiai jelentőséggel rendelkezik. (Ilyen térkép készítésekor a hangyák nem mehetnek át egymáson, mivel akkor nem derül ki, hogy a gyakori ütközések miatt nem jutnak át az átjárón.)

Amennyiben egy RTS játékban a háttérben fut az ant search, annak eredményei alapján a játékban szereplő objektumok a nyomvonalakat követve már egyszerű, pusztán a lokális környezetet figyelő algoritmussal képesek eljutni egyik helyről a másikra. (A lokális keresésre részben a nyom követése miatt van szükség, másrészt a nyomvonaltérkép nem tartalmazza a pillanatnyi helyzetét a többi mozgó objektumnak, így azok kikerülésére amúgy is fel kell készülni.)

A modern RTS játékok másik fontos eljárása a „terrain analysis" ([Ensemble]), ami a térkép elemzését jelenti. Ennek során a térképet kisebb, például homogén és konvex területekre bontjuk, melyeken belül már triviális az útvonaltervezés és mivel a térkép mezőinél lényegesen kevesebb van belőlük, taktikai döntéseknél is jobban kezelhetők. Az ant search szükség esetén széles utak formájában ki tudja jelölni a játékos számára fontos területeket, így könnyítve meg a terrain analysis munkáját. (A térképfelbontásra vonatkozó további információkat ebben a dolgozatban a gráf nélküli keresés fejezetben olvashat, valamit a [Stout99]-ben.)

Amennyiben a játékban az útvonalakat gráf formájában kell megadni, akkor a nyomvonaltérképet gráffá kell alakítani, de egyszerű, a nyomokat követő hangyák szimulálásával ezt is meg lehet tenni jelentős nehézségek nélkül.

Elemzés

Az ant search itt bemutatott megvalósításának nagy előnye, hogy igen jól skálázható és ezzel együtt jól hangolható a szabad erőforrások mennyiségének megfelelően. A szimulált hangyák számának csökkentésével az erőforrásigény (főleg processzoridő) csökkenthető, bár ez a teljesítmény csökkenését is magával vonja. Amennyiben a megoldandó feladat túl összetettnek látszik (nagyon lassan, vagy egyáltalán nem talál megoldást), akkor mindössze növelni kell a hangyák számát. (Amennyiben erre nincs lehetőség, akkor le kell lassítani a nyomvonalak párolgását, hogy na tűnjenek el, mire a következő hangya odaér.)

A szükséges hangyák számát elsősorban a várható úthossz ismeretében lehet megbecsülni. Konkrét párolgási paraméterek ismeretében legalább olyan sűrűn kell elmennie egy hangyának a nyomvonalon, hogy annak szintje hosszabb időtartam alatt se csökkenjen még akkor sem, ha éppen az aktuális beállítások miatt a hangyák nem követik szigorúan a

nyomvonalat. (A példaprogramban 50 szimulációs lépésenként párolog annyit a nyomvonal, mint amennyit egy hangya kibocsát, tehát legalább 50 lépésenként frissíteni kell a nyomot. Mivel azonban a hangyák részben véletlenszerűen mozognak, ezért érdemes +50%-kal számolni.) Amennyiben a keresési térben zsákutcák is előfordulhatnak, akkor a folyamatosan ingázó hangyák mellett figyelembe kell venni azokat is, melyek ezekbe a zsákutcákba kerülve megakadályozzák majd, hogy társaik is bemenjenek. Ezeknek a száma arányos a zsákutcák összméretével. (A példaprogramban általában a torlódási hormon figyelembe vétele esetén kb. 1/6-a a zsákutcában lévő hangyák száma a területéhez képest, ha a hangyák nem mászhatnak át egymáson.) Természetesen ilyenkor csak azokat a zsákutcákat kell figyelembe venni, melyek ugyanazon keresés esetén veszélyt jelenthetnek a hangyák számára. A fentiek mellett pedig bele kell számolni a „hangyaigénybe" azokat a hangyákat, melyek éppen nem a végül megtalált nyomon mozognak. Mivel ezek száma attól is függ, hogy a keresés mennyire talált már utat, valamint hogy éppen mennyire szigorúan követik a hangyák a nyomvonalakat, ezért érdemes legalább 20-30% „ráhagyással" számolni. Természetesen a fenti gondolatmenet alapján megbecsült hangyaszám csak iránymutató, de mindenképpen hasznos kiindulási pontot ad a pontos szám (jó eséllyel próbálgatásokon keresztüli, iteratív) meghatározásához.

Az algoritmus különösen akkor hasznos, ha széles utak vannak a keresési térben. Ilyenkor, ha egy egyszerű gráfkereső algoritmussal keresnénk az utat, akkor nagyon nagy lenne az elágazási tényező, az eredményként kapott útvonal pedig (például egy A*-hoz hasonló keresés esetében) nem fejezi ki, hogy az út széles, így nem kell szigorúan a megadott gráfpontokat követni. (Ha pedig csak lazán követjük a „hagyományos" keresés eredményét, akkor szűk szakaszokon mehetünk rossz irányba (nem a szűkület bejárata felé), amire a keskenyedő nyomvonal figyelmeztetne.) Az ant search eredménye tehát támogat egy bizonyos követési hibát.

Felvetődik a kérdés, hogy miért is lehet előnyösebb az ant search például egy fejlettebb, A*-hoz hasonló keresési algoritmusnál. Természetesen az, hogy melyik előnyösebb, az nagyon alkalmazásfüggő. Az ant search előnyei (például egy RTS játék keretein belül):

- Jól skálázható. A játék kevésbé mozgalmas szakaszaiban (például rögtön az elején, amikor még mindenki csak építkezik) bőven van erőforrás arra, hogy a szükséges szimulációkat a háttérben lefuttassuk. Ilyen játékokban a számítógép által irányított játékos általában már jó előre tudhatja, hogy hova akarja majd az egységeit eljuttatni,

így lehetősége van arra, hogy előre lefuttassa a kereséseket. (Emberi játékos esetében ennyire nem lehet előre tervezni.)

- Hatékonyan tudja támogatni az "influence map"-eket, vagyis az elkerülni, vagy pont érinteni kívánt területek megadását a kereső számára.
- Széles utak esetén az eredményben szereplő nyomvonal is szélesebb, így ha két egységnek ki kell kerülnie egymást, akkor egyiknek sem kell teljesen lemennie a nyomvonalról. (A kitérő objektum könnyen visszatalál a nyomvonal legerősebben jelzett részeire.)
- Az ant search a megtalált útvonal mellett egy RTS játék mesterséges intelligenciája számára további fontos információkat szolgáltat a torlódási hormon segítségével.
- Az eredmény ugyan nem egy egyértelmű útvonal, de a többi mozgó objektum kikerülése érdekében az egységeknek általában amúgy is szüksége van egy csak a lokális környezetet vizsgáló, rövid távú kereső algoritmusra, melynek segítségével a nyomvonal követése is megoldható.

Hátrányok

- Lassan stabilizálódik a megoldás.
- Sok erőforrást igényel, mivel hosszan kell futtatni a háttérben a szimulációt.

(Ez a két hátrány az RTS játékok fentebb említett egyenlőtlen erőforrásigénye miatt az ilyen alkalmazásokban például nem okoz gondot.)

- A szimuláció mellett kell egy magasabb szintű réteg, mely a paramétereket a statisztikáknak megfelelően, menet közben hangolja.

Összefoglalás

Összességében tehát az általunk kifejlesztett algoritmus a hangyák természetben alkalmazott módszerének egy a szakirodalomban hagyományosnak tekinthető „ant search”-től eltérő alkalmazása. Az eltérés egyrészt abban mutatkozik meg, hogy a mi esetünkben a keresési tér sokkal inkább egy nagy, homogén, járható területeket tartalmazó térkép, mint egy gráf. A lehetőségeket kihasználva a mi algoritmusunk eredményül egy olyan útvonalat határoz meg, mely szélesebb utakat tartalmaz (amennyiben van elég hely a térképen), így a rajtuk navigáló objektumok ki tudják kerülni egymást anélkül, hogy utána gondot okozna a visszatalálás az útra. A másik eltérés a keresés során kialakuló egyéb, a keresési térre

vonatkozó információk halmaza. A torlódási hormon segítségével fontos adatokhoz jutottunk, melyek a hagyományos útkereső algoritmusok egyáltalán nem szolgáltatnak.

Nemdeterminisztikus algoritmusunk ugyan nem garantálja az optimális útvonal megtalálását, de megfelelő paraméterezése esetén (mely a dolgozatban említett szabályszerűségek segítségével megtehető) nagy valószínűséggel sok alkalmazás számára elegendően jó megoldást szolgáltat. Jelentős erőforrásigénye ellenére hasznos olyan esetekben, amikor egy akár kis mértékben változó keresési térben olyan (progresszív) keresési algoritmus kell, mely az útvonalak forgalmi terhelésére vonatkozó információkat is szolgáltat. Az ilyen módszerek igen hatékony segítséget nyújtanak például az RTS játékokban, melyeknek gyakori, torlódásokkal kapcsolatos hiányosságaik az egész módszer kidolgozásának ötletét felvetették.

A folytatás

Az ant search előnyeinek és hátrányainak elemzése után következzenek további kutatási irányok, melyeket a későbbiekben érdemes megvizsgálni:

A szimulátor paramétereinek genetikus algoritmussal ([Russell], 727. o.) vagy PSO-val (particle swarm optimization, [Gordon]) történő javítása: A szimuláció számos paraméterét jelenleg egy a szimulátor felett álló rétegből lehet módosítani a statisztikáknak megfelelően. A paraméterek akár általános, akár egy esetre vonatkozó (és így menet közben történő) optimalizálását ehelyett genetikus módszerrel, vagy az ehhez hasonló PSO eszköztárával is lehet hangolni. (A PSO lényege, hogy a paraméterértékek terében kis „részecskék” szimbolizálják az egyes beállításokat (mint az egyedek a genetikus módszereknél), a jósági függvény kiértékelése után pedig mind az éppen legjobb megoldást szimbolizáló részecske felé mozdulnak el.)

A szimulátort hangoló réteghez szakértői rendszer fejlesztése: mint ahogy a dolgozatunkból is látszik, a paraméterek beállításának módja viszonylag egyszerű szabályokkal megadható. Ezeket bele lehet ágyazni egy szakértői rendszerbe, mely aztán akár tovább is fejlesztheti őket.

A korábban már említett „terrain analysis” (melyet például a modern RTS játékok alkalmaznak a térkép vizsgálatakor) alapjául szolgálhatnak az ant search eredményei. A térkép kisebb (a játék szempontjából homogén) területekre való felbontásában az ant search jelentős segítséget jelenthet, mivel kijelöli a csapatmozgások szempontjából eltérő fontosságú területeket.

Érdemes megvizsgálni, hogy további hormonok felvételével milyen egyéb információk nyerhetők ki az ant search eredményeiből.

A hangyák differenciálása

Amennyiben több „cél felé irányító” hormont használunk és minden hangyának meg lehet adni, hogy ő melyiket kövesse, akkor egyszerre kettőnél több pont között is kereshetünk utat, ráadásul az utak kereszteződésénél jelentkező esetleges torlódások veszélyét is ki tudjuk mutatni. (Azért kell ehhez több „célhormon”, mert különben a legközelebbi célt találná meg a keresés.)

Definiálhatunk olyan hangyákat, melyek kezdetben szándékosan kerülik a nyomvonalakat, utána pedig véletlenszerűen bolyonganak. Ha a bolyongás során nyomvonalat találnak, azt elkezdik „összekötni” azzal a ponttal, ahol legutoljára letértek a nyomvonalról. Így további mellékutakat nyerhetünk.

A hangyák állapotterének bővítése

Ha minden hangya akadály miatti fordulás esetén egy ideig még tárolja, hogy az akadály előtt merre ment és egy ideig még előnyben részesíti azt az irányt, akkor az előrehaladás sebessége valószínűleg növelhető, mert a hangyák a kis akadályokat könnyebben ki tudják kerülni anélkül, hogy teljesen elkanyarodnának és az addigihoz képest oldalirányba mennének tovább.

A program használata és néhány fontos implementációs kérdés

Ebben a részben az általunk készített demonstráló program használatáról és a jelentősebb implementációs kérdésekről lesz szó.

A program részei

A program Java nyelven íródott, elsősorban hordozhatóság okokból. Az indítható osztály az "Antsearch". Indítás után 3 ablak jelenik meg. Ezek a következők:

- Irányítópult: A szimuláció irányításához szükséges funkciókat innen lehet elérni.
- Térkép: A keresési tér aktuális állapotát és a hangyák helyzetét mutatja.
- Statisztika: A szimuláció során itt jelennek meg a statisztikai információk.

Az irányítópult nyomógombjai és a beállítási lehetőségek:

Start, Szünet, Kilépés

Jelentésük nem igényel részletes magyarázatot. A szimuláció indítására, felfüggesztésére illetve újraindítására, valamit a programból való kilépésre szolgálnak.

Megjelenítendő hormon

Az első listán választhatja ki a felhasználó, hogy a térképen kék csíkok formájában melyik hormon szintje jelenjen meg. (Nyomvonal, torlódás, Boly vagy Cél)

A minimális megjelenítendő hormonszint

A térképen, egy mezőn csak akkor jelenik meg a hormon szintje, ha eléri ezt a szintet. A térképen megjelenő csík minimális vastagságának ez az érték fog megfelelni, míg a legvastagabb csíkhoz mindig a maximális hormonszint tartozik. (A maximumot módosítani nem lehet.)

Szimuláció sebessége

Az egy megjelenítési ciklus alatt végrehajtandó szimulációs lépések számát itt lehet kiválasztani. Egy szimulációs lépés alatt minden hangya lép egyet, kivéve, ha semelyik irányba nem tud lépni. (A statisztika változásainak figyeléséhez érdemes százasával futtatni a szimulációs lépéseket.)

Irányválasztási súlyozás

A hangyák irányválasztásánál alkalmazott súlyozást határozza meg. Ez minden esetben 5 értéket jelent: az előre és hátrafelé mutató irány súlyát, a nyomvonal (többi irányhoz viszonyított relatív) szintjének súlyát, a torlódási hormon nyomvonalhoz hasonló súlyát, valamint az aktuális „célhormon” (szintén relatív értékének) súlyozását.

- „alap súlyozás”: Ez az alapértelmezett beállítás. A hangyák ritkán fordulnak vissza, törekszenek a nyomvonal követésére, nagyon kerülik a torlódási hormont, és ha a hormonnyomok nem segítenek az irányválasztásban, akkor a cél felé fordulnak.
- „terjeszkedő”: Ez a súlyozás radikálisan lecsökkenti a nyomvonalak követését, így a hangyák keresési térben való szétszórására szolgál. Rövid időre kiválasztva a hangyák szétszóródnak, így könnyen kijutnak a lokális maximumokból. Hosszabb távon nem érdemes alkalmazni, mivel az útvonalkeresést közvetlenül nem segíti.
- „szigorú”: Ez a beállítás lecsökkenti a torlódási hormon súlyát, így akkor hasznos, ha már megvan a keresett út és nem szerencsés, ha a kisebb tömörülések nagyon elriasztják a hangyákat.
- „útkövető”: A „szigorú” beállítás tovább szigorított verziója. A hangyák a nyomvonalak követésére törekednek. Minden egyéb szempont másodlagos. Előnye, hogy mivel az előre mozgás még itt is jelentős súllyal rendelkezik, az utakra simító hatással van.
- „zsákutca-elkerülő”: A hangyák ezzel a súlyozással igyekeznek követni a nyomvonalakat és messze elkerülni a tömörüléseket. Ez akkor lehet előnyös, ha egy zsákutca bejáratához közel kell elmenniük. Ebben az esetben - ha a zsákutca valóban sok hangyát tartalmaz és ezért sok benne a torlódási hormon - a hangyák jelentős része nem fog bemenni a zsákutcába.

beállítás	Előre	Hátra	Nyomvonal	Torlódási hormon	Célhormon
alap súlyozás	1	-4	3	-4	2
terjeszkedő	2	-4	1	-4	1
szigorú	1	-4	3	-2	2
útkövető	2	-4	4	-2	1
zsákutca-elkerülő	2	-4	3	-4	2

A gyakorlatban az egyszerűbb feladatok megoldása esetén általában elég az „alap” és a „szigorú” súlyozás.

Nyomvonalhúzási lépésszámkorlát

Az itt megadott lépésszám megtétele után a hangyák (ha nem értek célponthoz) már egyáltalán nem húznak nyomvonalat. Előtte a kibocsátott nyomvonalhormon szintje lineárisan csökken. Amennyiben az önkalibrációs feltétel a legutolsó célpont elérése óta megtett lépésszámra vonatkozik, akkor ennek a lépésszámnak a kétszerese után kerül sor az önkalibrációra, melynek hatását szintén az irányítópulton lehet kiválasztani.

Átmászás

Aktiválva a hangyák átmehetnek egymáson. A súlyozáson és torlódási hormon kibocsátáson ez nem változtat, mindössze azok az irányok is választhatók lesznek, amerre másik hangya van. Ez a beállítás akkor előnyös, ha a torlódási hormon szintjét nem akarjuk vizsgálni, viszont szeretnénk egy keskeny utat kapni. (Ilyenkor a hangyák keskenyebb sávban mozognak majd, mert nem kell egymást kikerülniük.)

A torlódási hormon hatástalan

Bekapcsolva a hangyák nem veszik figyelembe a torlódási hormont. Ebben az esetben a hangyák sokkal jobban besűrűsödhetnek a lokális maximumokban. Ez elsősorban az "átmászással" együtt hasznos, mivel annak engedélyezése még nem jelenti a torlódási hormon kibocsátásának kikapcsolását, így a keskeny utakon sok lesz ebből a hormonból, ami gátolja az út sávjának összeszűkülését. (Mivel a hangyák ugyan átmehetnek egymáson, de kerülni fogják azokat a pontokat, ahol erre gyakran sor kerül.)

Az önkalibráció feltétele

- „nincs”: Egyáltalán nincsen önkalibráció, vagyis a hangyák a saját méréseik alapján soha nem módosítanak a paramétereiken. Ez az alapértelmezés szerinti beállítás.
- „táv”: Akkor aktiválódik az „önkalibráció”, ha a legutolsó célpont elérése óta megtett lépések száma meghalad egy bizonyos küszöbértéket (lásd „Nyomvonalhúzási lépésszámkorlát”). A tesztek során ez a beállítás bizonyult a legelőnyösebbnek a keresés szempontjából.

- „távváltozás”: A feltétel akkor teljesül, ha a hangya egy bizonyos lépésszám alatt (jelenleg 30 lépés) nem mozdul el legalább egy megadott távolságra (jelenleg 4 lépésnyire).
- „szintváltozás”: Hasonló az előzőhöz, de a cél irányát mutató hormon szintjére vonatkozik a feltétel és nem a megtett távolságra.

Az önkalibráció hatása

Amennyiben az előző listán kiválasztott feltétel teljesül, akkor az önkalibráció hatása az itt meghatározott lesz. Ez a módosítás egészen addig fennáll, amíg a hangya el nem éri a célpontot.

- „Visszafordulás”: A hangya célpontnak azt a pontot jelöli ki magának, ahol legutoljára volt. (A mostani implementációban tehát ha a bolytól a cél felé ment, akkor most a boly felé megy tovább, ellenkező esetben a cél felé.) A tesztelések során ez a beállítás sokkal hasznosabbnak bizonyult, mint a „bolyongás”.
- „Bolyongás”: A hangya felezi a célhormon súlyozását, így ha az önkalibráció oka valóban az, hogy egy lokális maximumba került, akkor ennek hatására könnyebben kijut majd a zsákutcából, mivel nem ragaszkodik annyira a cél felé mutató irányhoz.

A térkép ablak

A térkép a keresési tér aktuális állapotát és a hangyákat mutatja. A fekete négyzetek a falakat szimbolizálják, a piros és zöld négyzetek a hangyákat, a kis piros és zöld négyzetek a két célpontot (A boly piros, a cél zöld). Adott színű hangya mindig az ellentétes színű célpont felé keresi az utat.

(Dolgozatunk nyomtatott, fekete-fehér verziójában a hangyákat a falaknál kicsit kisebb, világos- és sötétszürke négyzetek jelölik.)

A kiválasztott hormon szintjét kék (fekete-fehér nyomtatásban fekete), vízszintes csíkok jelölik a mezőkön. A minimális (1 pixel) vastagság felel meg az irányítópulton megadott minimális megjelenítendő hormonszintnek, a „kék négyzet” pedig a maximális szintnek felel meg.

A hangyák zöld vagy piros négyzete fekete, átlós vonallal át van húzva, ha már nem húznak nyomvonalat. Az önkalibráció aktív voltát a hangya négyzetének közepén kicsi lila (fekete-fehérben középszürke) négyzet jelöli.

Kezdetben a térképen nincsen fal, a hangyák pedig véletlenszerűen szétszórva indulnak. Falat az egérrel lehet rajzolni (a bal oldali egérgomb nyomva tartásával), törölni pedig hasonlóan, de a Shift billentyű lenyomása közben lehet.

A statisztika ablak

A statisztika ablak a globális mérések eredményeit tartalmazza.

„Beérkezésiszám eloszlása (20 mérés, egyenként 50 lépésre)”: A szimulátor 50 szimulációs lépésenként összeszámolja, hogy közben hányszor jutott el egy hangya egy célpontba. (Ebbe nem számít bele az olyan beérkezés, ami az önkalibráció miatti visszafordulás után következett, mivel az nem jelenti azt, hogy egy hangya átért a célponthoz.) A legutolsó 20 ilyen érték eloszlását mutatja a diagramm. A „+” sor a 10 feletti értékek számát mutatja.

„Önkalibrációk száma”: Az 50 szimulációs lépés alatt teljesülő önkalibrációs feltételek száma. Ha ez az érték magas (jelen esetben nem nulla), az azt jeleni, hogy sok hangya kerülhetett lokális maximumba.

„Nyomvonalon megtett lépések aránya”: A legutóbbi 1000 hangyalépésben (ami 1000 / "hangyák száma" szimulációs lépésnek felel meg) azon lépések aránya, amiket az adott hangya nyomvonalon állva tett meg, vagyis a nyomvonalhormon értéke alatta nagyobb volt, mint nulla,

Egyéb, implementációval kapcsolatos kérdések

Párolgás

A párolgás kérdése egy olyan témakör, mellyel eddig csak érintőlegesen foglalkoztunk. A vizsgálódások során kiderült, hogy a nyomvonalak esetében előnyösebb, ha a párolgás két részből tevődik össze:

- Viszonylag ritkán a nyomvonalak szintje csökken egy bizonyos értékkel.
- Amennyiben a nyomvonalak szintje több mint 10 térképmezőn eléri a maximális értéket, akkor az egész térképen mindenhol felezzük ennek a hormonnak ez értékét.

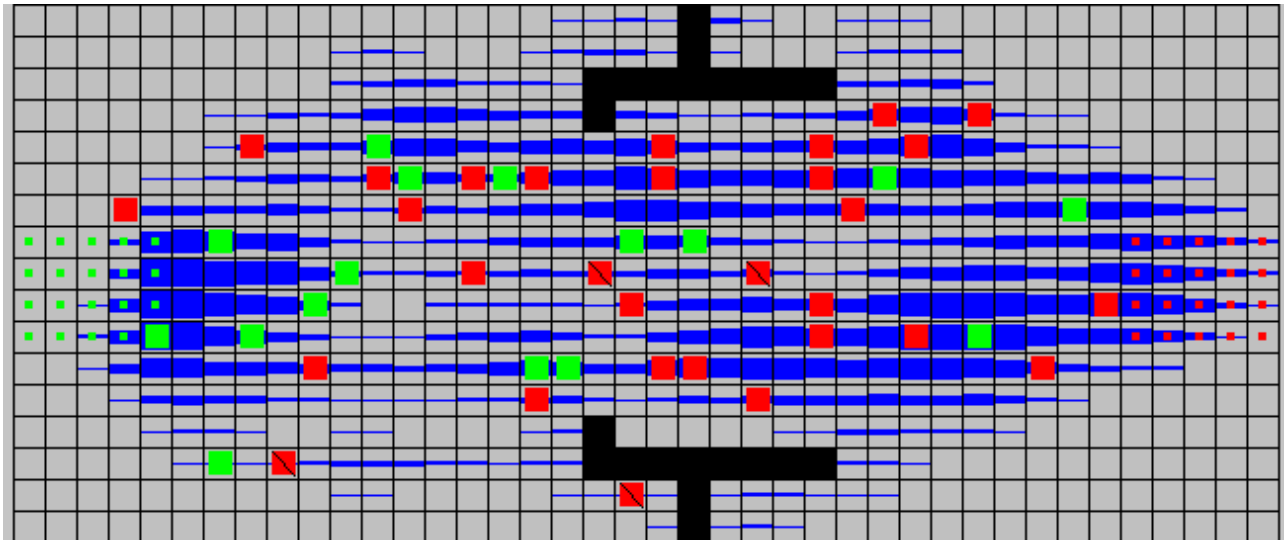
Ez a megoldás azért bizonyult előnyösebbnek, mert a hormonnak nem kell túl gyorsan párolognia ahhoz, hogy ne érje el túl sok helyen a maximális értéket. Így az utak több ideig „láthatóak”, de az sem okoz gondot, hogy az értékkészlet nagyon nagy lehet.

Hormonszintek és a súlyozás

Mivel a hangyák a hormonszinteket úgy veszik figyelembe, hogy előtte a környezetükben fellelhető értékeket a 0-100 intervallumra skálázzák, ezért ha minden irányban nagyon magas egy hormon szintje, a súlyozás miatt akkor sem fordulhat elő, hogy a beállításokkal ellentétben egy másik hormonnak mégsem lesz kellő jelentősége. (Ez a helyzet fennállna abban az esetben, ha simán csak a hormonszintet szoroznánk a súllyal.)

Az eredmény

Előfordul, hogy a futási eredmény egy erősen változó („pulzáló”) térkép. Ennek oka egyrészt a párolgás algoritmusa, másrészt az ant search véletlenszerű jellegéből következik. Amennyiben a konkrét alkalmazásban ezt ki kell küszöbölni, akkor ilyenkor a végeredményt az útvonalak stabilizálódása után több hormontérkép átlagolásával kell előállítani. (Például az utolsó 500 szimulációs lépés során minden századik után elmentjük a térképet, majd a végén ezeket átlagoljuk.)



27. ábra - A nyomvonaltérkép egy pillanatnyi állapota: a széles úton a pontos hormonszintviszonyok időben ingadoznak

Felhasznált irodalom

[3DS] Discreet *3D Studio MAX*,
http://www.discreet.com/3dsmax/

[ACO] *„Ant Colony Optimization”*,
http://iridia.ulb.ac.be/~mdorigo/ACO/publications.html

[Ants02] 3. ANTS 2002 (Brussels, Belgium)
http://www.informatik.uni-trier.de/~ley/db/conf/antalg/antalg2002.html

[Aurenh] Franz Aurenhammer, Rolf Klein: *„Voronoi Diagrams”*,
http://wwwpi6.fernuni-hagen.de/publ/tr198.pdf

[Baert00] Stefan Baert: *„Motion Planning Using Potential Fields”*, www.gamedev.net,
http://www.gamedev.net/reference/articles/article1125.asp

[Botee99] Hozefa M. Botee and Eric Bonabeau: *„Evolving Ant Colony Optimization”* (Santa Fe Institute, 1999.) http://www.santafe.edu/sfi/publications/wpabstract/199901009

[DSP] *„Jelfeldolgozás”*, az Eötvös Loránd Tudományegyetem Információtechnológiai Oktatási Laboratórium elektronikus jegyzete,
http://itl7.elte.hu/html/jelfel/node41.htm

[Elias] Hugo Elias: *„Inverse Kinematics”*,
http://freespace.virgin.net/hugo.elias/models/m_ik.htm

[Ensemble] *„Ensemble Studios Terrain Analysis in Realtime Strategy”* (Ensemble Studios - Studios Programmers Journal)
http://www.ensemblestudios.com/news/devnews/terrain1.shtml (Terrain analysis)

[Fujita02] Kazuyuki Fujita, Akira Saito, Toshihiro Matsui, Hiroshi Matsuo: *„ An Adaptive Ant-Based Routing Algorithm Used Routing History In Dynamic Networks”* (Deparment of Electrical and Computer Engineering, Nagoya Institute of Technology, Gokiso-cho, Showa-ku, Nagoya, 466-8555, JAPAN)
http://mars.elcom.nitech.ac.jp/~matsuo/SEAL02-2.pdf

[Gordon] David Gordon: *“Collective Intelligence in Social Insects”* (AI depot) http://ai-depot.com/Essay/SocialInsects-Swarm.html (Particle swarm optimization)

[GPG2] Game Programming Gems II. (Charles River Media), 3.10. (324.-329.)

[Greulich] René Greulich: Diplomarbeit *(„Charakterverhalten in Computerspielen: Pathfinding”*), http://www.gm.fh-koeln.de/~faeskorn/diplom/diplom_greulich.pdf

[HL] *Half-Life* (Sierra Entertainment), http://half-life.sierra.com/

[Kaegi03] Simon Kaegi and Tony White: *"Using Local Information To Guide Ant Based Search"* (School of Computer Science, Carleton University, 2003.) http://www.scs.carleton.ca/~arpwhite/documents/lbt-iea-aie-2003-final.pdf

[Kroon02] Ronald Kroon: *„Dynamic vehicle routing using ant based control"* (Master of Science thesis Delft University of Technology, 2002.) http://www.kbs.twi.tudelft.nl/Publications/MSc/2002-Kroon-MSc.html

[Kwang92] Yong K. Kwang, Narendra Ahuja: *„Gross motion planning—a survey"* (ACM Computing Surveys, ISSN: 0360-0300, 1992.)

[Levine02] John Levine and Frederick Ducatelle: *"Ant Colony Optimization and Aggressive Local Search applied to Bin Packing and Cutting Stock Problems"* (Division of Informatics University of Edinburgh - Informatics Jamboree 23rd May 2002) http://www.aiai.ed.ac.uk/~johnl/antbin.pdf

[Nell02] T. W. Neller: *„Action-Based Discretization for AI Search"*, in S. Marshall, ed., Proceedings of the Game Developers Conference 2002 (GDC 2002, San Jose Convention Center, San Jose, California, USA, 2002), CMP United Business Media LLC, 2002.

[Pinter01] Marco Pinter: *„Towards More Realistic Pathfinding"*, www.gamasutra.com, 2001. 03. 14., (http://www.gamasutra.com/features/20010314/pinter_01.htm)

[Pottinger99] By Dave C. Pottinger: *„Implementing Coordinated Movement"* (Gamasutra 1999, Originally Published in the February 1999 issue of: Game Developer) http://www.gamasutra.com/features/19990129/implementing_01.htm (Stacked Canyon probléma)

[Reimann02] Marc Reimann: "Ant Based Optimization in Goods Transportation" http://www.ads.tuwien.ac.at/teaching/ws02/EvolAlg/ACO-Teil2.pdf

[Russell] Stuart J. Russell – Peter Norvig: *Mesterséges intelligencia modern megközelítésben,* Panem-Prentice Hall, 1995.

[Stout99] W. Bryan Stout: *„Smart Moves: Intelligent Pathfinding"*, www.gamasutra.com, 1999. 02. 12., (http://www.gamasutra.com/features/19990212/sm_09.htm)

[UR] *Optimising A** http://www.ur.co.nz/urcorp/default.asp?data_article=242

Ábrajegyzék