Kísérletek minimax és alfa-béta nyesés algoritmusokkal kétszemélyes játékokban

Mesterséges intelligencia házi feladat

2014./2015. ősz

Tallér Bátor (UGA4OQ)

Török Attila (FULGDF)

Konzulens:  
 Balassa Gábor

Tartalomjegyzék

[Feladatválasztás 3](#__RefHeading__147_1848191074)

[A megvalósítás 4](#__RefHeading__151_1848191074)

[Az ágensek rövid leírása 4](#__RefHeading__149_1848191074)

[A program használata 5](#__RefHeading__519_1372879235)

[Tapasztalatok 6](#__RefHeading__153_1848191074)

[Az ágensek játékbeli teljesítménye 6](#__RefHeading__532_1372879235)

[Az ágensek futási ideje 7](#__RefHeading__734_1372879235)

[A bejárási mélység korlátozásának hatásai 8](#__RefHeading__736_1372879235)

1. Feladatválasztás

Házi feladatunkként a minimax algoritmus kétszemélyes játékokban való alkalmazását, majd alfa-béta nyesés alkalmazásával való meggyorsítását, hatékonyabbá tételét választottuk. A konkrét játék, amelyet felhasználunk, a feladat szövegében is példaként említett nim, annak is az a változata, ahol nem elkerülendő, hanem a cél az utolsó kavics elvétele.

A szoftver, amelyet megvalósítunk, egyrészt egy általános nim-keretrendszer lesz, amely tetszőleges algoritmussal dolgozó ágenseket tud egymás ellen játszatni. Mérni fogja az időt, amely az egyes döntéseikhez szükséges volt, követni a játék állását (az egyes kupacokban pillanatnyilag lévő kavicsok számát), és kinyilvánítani a nyertest. Képes lesz továbbá mindezeket az adatokat naplózni is, későbbi elemzés céljából. A skálázhatóságot az adja, hogy a játékok elején beállítható lesz, hogy hány kupac legyen, és mennyi kavics legyen bennük.

Másrészt pedig nyilván tartalmazni fog néhány ágens-megvalósítást is, amelyekkel a méréseket elvégezzük. Az egyik természetesen a minimax stratégiát fogja alkalmazni, opcionálisan beállítható felső korláttal a keresőfájának mélységére. Egy másik hasonlóan fog működni, de az alfa-béta nyesést is alkalmazza, gyorsabb döntések reményében. Ezeken felül még több primitív ágenst is tervezünk megvalósítani, (például véletlenszerű kupacból vesz el véletlenszerű számú kavicsot, vagy mindig a legelső, nem üres kupacból vesz el egyet, és hasonlók), amelyek főként ellenségként fognak szolgálni a bonyolultabb algoritmusúak ellen, hogy a játékok automatikusan lefuttathatóak és tesztelhetőek legyenek.

A minimax stratégiát alkalmazó ágenseknél az egyszerűség kedvéért azt feltételeztetjük az ágenssel, hogy az ellenfele ugyanazokat a döntéseket hozza az egyes helyzetekben, mint ő maga tenné. Emellett az az érvünk, hogy a minimax elv feltételezi, hogy mindkét játékos optimálisan játszik. Amennyiben tehát az ágens a saját stratégiáját optimálisnak hiszi, alkalmazhatja azt az ellenség szemszögéből is. Ha pedig mégis tudna jobbat, amit az ellenség alkalmazhatna szerinte, akkor ő maga is aszerint játszana, tehát mégis egyformán játszanának.

Ezek mellett, amennyiben időnk engedi, beépítenénk egy „emberi játékos” ágenst is, amely cselekvéseit a felhasználó határozza meg, így ezen keresztül tulajdonképpen bármilyen algoritmus kipróbálható lesz a szoftverben, annak módosítása, vagy további bővítése nélkül is.

Megvalósításhoz a C# nyelvet választottuk, a grafikus felülethez a jól megszokott Windows Forms technológiát fogjuk használni.

1. A megvalósítás
   1. Az ágensek rövid leírása

**Random 1 ágens** (RandomHeapAgent): Minden körben egy véletlenszerűen kiválasztott, még nem üres kupacból vesz el pontosan egy követ.

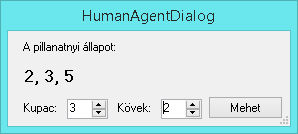
**Random N ágens:** (RandomAgent): Minden körben egy véletlenszerűen kiválasztott, még nem üres kupacból vesz el véletlenszerű számú, de legalább egy követ.

**Minimax ágens** (MinimaxAgent): A pillanatnyi állapotból kiindulva bejárja a teljes állapotfát, és a minimax algoritmus szerint értékeli az egyes lépések eredményességét. Amennyiben egy játékterv végén az ágens nyerne, 1-es pontszámot rendel hozzá, amennyiben vesztene, -1-est, ha pedig a játszma véget érte nélkül elérte a bejárási mélység-korlátot, jobb ismeret híján 0-val fog számolni.

**Nyeséses minimax ágens** (PruningMinimaxAgent): Ugyanúgy működik, mint az egyszerű Minimax ágens, de már alfa-béta nyesést is alkalmaz, így jelentősen lecsökken a fa bejárandó részeinek száma, ezáltal a döntési sebesség jelentősen nő, anélkül, hogy ez bármilyen hátránnyal járna a döntések minőségét illetően.

**Összegző minimax ágens** (SummingMinimaxAgent): Egy alternatív pontszámítást használó, minimaxhoz hasonló elven működő ágens, amely nem egyszerűen minimumot és maximumot keres az egyes csomópontokban, hanem összegzi az adott részfából kiindulva elérhető végállapotok pontszámát, így egy finomabb felbontású értéket kap egy-egy lépés jóságáról. Ennek viszont az a hátulütője, hogy nem lehet nyesni a fát, hiszen az összegzés miatt az egyes csomópontok értékének meghatározásához mindenképp teljesen be kell járni a hozzájuk tartozó részfát. Ezen felül a bejárási mélység korlátozása is jelentősebb romlást okoz az ágens döntéseinek jóságában.

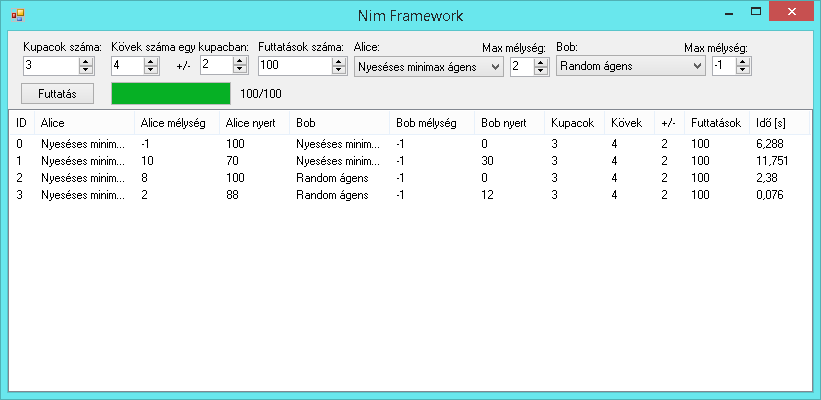
**Felhasználó ágens** (HumanAgent): Ez nem egy konkrét algoritmus, mindössze egy párbeszédeblakban kijelzi a játék pillanatnyi állását a felhasználó felé, aki ezután megadhatja, hogy melyik kupacból hány követ szeretne elvenni. A párbeszédablak kinézete a következő:



Használata magától értetődő.

A program használata

Az elkészült szoftver fő felhasználói felülete:



A legfelső sorban beállíthatjuk a szimuláció paramétereit. A kőkupacok számát, az egy kupacban lévő kövek átlagos számát, valamint az attól való legnagyobb véletlenszerű eltérést (az eloszlás egyenletes, illetve amennyiben ez a beállítás 0 értékű, az összes kupacban ugyanannyi kő lesz), és az egymás után lejátszatni kívánt játékok számát. Kiválaszthatjuk a két ágens, Alice és Bob stratégiáját – azaz hogy melyik ágens-implementáció töltse be a szerepüket – és a hozzájuk tartozó fabejárási mélység-korlátot, ahol értelmes. Ez utóbbi beállításnak a Random és a Felhasználó ágenseknél nincs szerepe. A játékot mindig Alice kezdi.

A második sorban lévő gombbal elindíthatjuk és megállíthatjuk a szimulációt, a folyamatjelzővel pedig nyomon követhetjük, hogy az ütemezett játékok közül hány futott már le.

Legalul pedig az összes eddigi, sikeresen lefuttatott szimuláció eredményét láthatjuk egy listában, azok minden lényeges adatát – a kiinduló beállításokat és az eredményeket is – feltüntetve.

Ugyanezt a listát nagyon hasonló alakban egy pontosvesszővel tagolt csv fájlba is lementi a program a futtatási könyvtárába, results\_*yyMMdd*\_*HHmmss*.csv néven, ahol *yyMMdd* a program indításakori rendszeridő év, hónap és nap része, *HHmmss* pedig az óra, perc és másodperc. Ez a fájl könnyedén importálható bármilyen adatfeldolgozó alkalmazásba későbbi feldolgozás, elemzés céljából.

1. Tapasztalatok

Az ágensek játékbeli teljesítménye

Az összes ágens implementációt az összes ellen 3 kupaccal, kezdetben mindegyikben 3 ± 2 kővel, egymás után 100 meneten keresztül játszatva a következő eredményeket kaptuk:



A bal oldali felsorolásban Alice, a kezdő játékos, fent pedig Bob, aki másodikként lép először. A számok azt jelentik, hogy a 100 meccsből hányat nyert meg Alice.

Az, hogy a mátrix nem „komplementer szimmetrikus” (olyan értelemben, hogy a mátrix egy elemének és a transzponáltjának ugyanazon elemének nem mindenhol 100 az összege) a fő tengelyére, illetve hogy a főátlóban nem csupa 50-esek szerepelnek, arra utal, hogy a kimenetelt nagyban befolyásolja az, hogy ki teszi meg az első lépést.

Ez a Random ágensek által játszott játékok esetében viszont nem így van, hiszen azok nem is egy megfontolt stratériát követnek, „ész nélkül” játszanak, akármi is a játék állása.

Megnyugodva figyeltük meg, hogy az alfa-béta nyesés alkalmazása valóban nem vezet semmiféle csökkenéshez az ágens ítélőképességét tekintve, mint ahogy nem is szabad ezt tennie. A nyeséssel ugyanis nem dobunk el semmilyen információt, ami bárhogy hozzájárulhatna a döntéshozatalhoz, csak felesleget.

Az ágensek futási ideje

A bejárási mélység korlátozásának hatásai