INTELLIGENS RENDSZEREK I. LABORATÓRIUM

6. laborgyakorlat:

Evolúció és Játékelmélet

Készítette:

Kovács Dániel László
dkovacs@mit.bme.hu

Méréstechnika és Információs Rendszerek Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

2009, április.

Tartalomjegyzék

1. Követelmények

- A laborgyakorlat teljesítésére 4 órányi tiszta munkaidő áll rendelkezésre.
- Időre történő, maradéktalan teljesítéshez a mérési segédlet hiánytalan ismeretét feltételezzük.
- A sikeres teljesítéshez a gyakorlat során végzett munka – a futási eredményeket is beleértve – jegyzőkönyv formájában történő dokumentációja és határidőre történő leadása szükséges.

JEGYZŐKÖNYV:

A jegyzőkönyvnek – a megoldók nevén, neptun-kódján, és e-mail címén kívül – a következőket kell tartalmaznia **minden egyes részfeladatra vonatkozóan**:

- KI-MIT oldott meg (feladat felosztása és értelmezése).
- HOGYAN oldották meg (megoldás menetének leírása).
- MIÉRT úgy oldották meg (választott megoldás indoklása).
- EREDMÉNYEK összefoglalása (értelmezés, értékelés, screenshot-ok, magyarázat).

LEADÁS:

A jegyzőkönyvet (**PDF** formátumban), és a forráskódokat (azaz a `\jade\src\milab\lab06` könyvtár teljes tartalmát **ZIP**-be csomagolva) legkésőbb a **laborgyakorlati héten péntek délig** e-mail-ben, fájlként csatolva kell a **dkovacs@mit.bme.hu** címre elküldeni. A levél subject mezejében „**`[IR1LAB6]`**” szerepeljen (idézőjelek nélkül), míg a levél szövege rendre a *megoldók neve* és *neptunkódja* legyen. Több megoldó esetén csak egyetlen alkalommal, az egyik megoldónak kell elküldenie a megoldást, amit a többieknek is CC-ézzen.

2. Bevezető

A laborgyakorlat során evolúciós játékelmélettel fogunk foglalkozni. Megismerkedünk a Neumann János által több, mint fél évszázaddal ezelőtt kidolgozott (közel 100 éves múltra visszatekintő), és azóta számtalan forradalmi újítással kiegészült **Játékelmélet** alapjaival, majd az elmélet evolúciós vonatkozásaival a gyakorlatban.

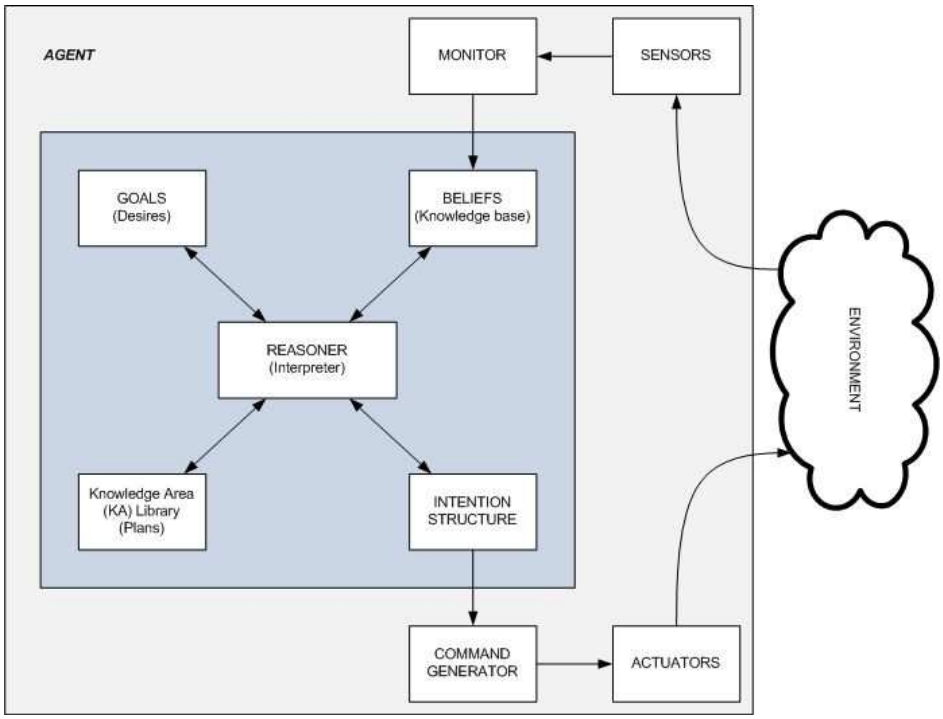
2/A. PRS – BDI – JADEX

Most is, ahogy az előző gyakorlatok során, a **JADE** (**J**ava **A**gent **DE**velopment framework) keretrendszert (`http://jade.tilab.com`) fogjuk használni. Sőt, jelen esetben ennek egy **JADEX** nevezetű kiterjesztésével (`http://vsis-www.informatik.uni-hamburg.de/projects/jadex`) is megismerkedünk.

A JADEX valójában egy, a JADE-hez hasonló komplex stand-alone keretrendszer – a FIPA szabvány (`http://www.fipa.org`) egy másfajta implementációja. Kevésbé általános, több megkötést tesz az ágensek felépítésére/viselkedésére, ámde nem csökkenti a megvalósítható rendszerek/funkcionalitások körét. Épp ellenkezőleg, JADEX-ben – hála a jóval konkrétabb ágens architektúrának – jóval módszeresebben tervezhetők elosztott intelligens rendszerek.

A JADEX lényegében egy **PRS** (**P**rocedural **R**easoning **S**ystem) rendszerként tekint az ágensre, azon belül is a **BDI** (**B**elief-**D**esire-**I**ntention) architektúrára alapoz (`http://en.wikipedia.org/wiki/BDI_software_agent`).

A PRS rendszerek nagyon hasonlóak a szabály-alapú szakértő rendszerekhez[1], de számos jelentős különbség is van, amiknek a részletezésébe most nem térünk ki. A következő ábra szemlélteti a PRS rendszerek vázlatos architektúráját.



1. ábra: PRS rendszerek vázlatos architektúrája

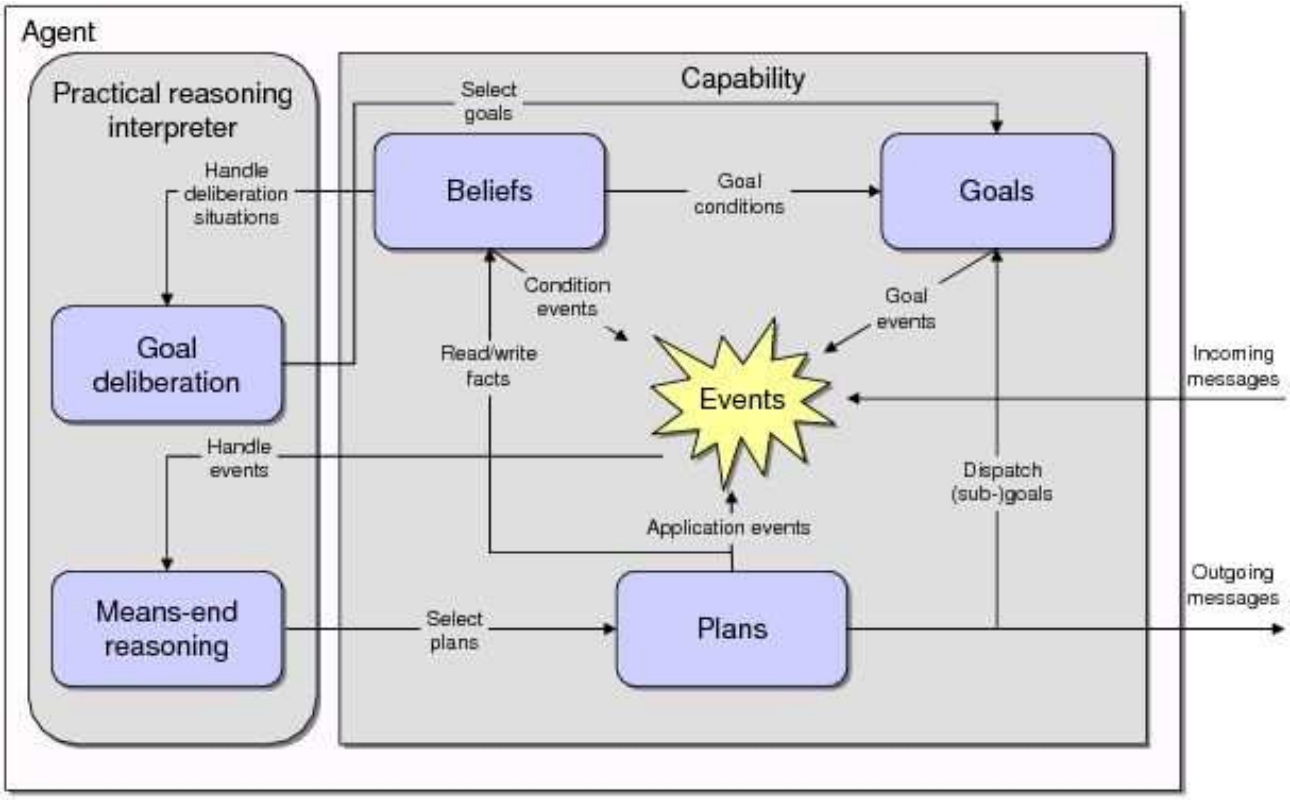
[1] Az egyik előző gyakorlaton már foglalkoztunk JESS-es szakértő rendszerekkel (http://herzberg.ca.sandia.gov/jess).

Az `1. ábra` jól mutatja a BDI rendszerek stílusjegyeit: szerepelnek benne a **hiedelmek**, a **vágyak**, és a **szándékok** számításelméleti reprezentációi.

A JADEX keretrendszer esetén a hiedelmek (`Beliefs`) tények (`Facts`) formájában állnak rendelkezésre az ágens hiedelem-bázisában (`Belief Base`).[2] A vágyak (`Desires`) célok (`Goals`) formájában adottak. A szándékok (`Intentions`) pedig terveket (`Plans`) jelentenek.

Le kell szögezni, hogy a JADEX nem egy előre tervező tervkészítő rendszer. A JADEX BDI alapú, célvezérelt logikai következtetést végez (hasonlóan egy visszafelé láncolt szabály-alapú szakértő rendszerhez). A működés lényege, röviden: minden időpillanatban adottak valamiféle célok, illetve hiedelmek, adott továbbá tervek egy készlete. A céloktól visszafelé indulva a rendszer megpróbálja „illeszteni" a terveket (mint HA-AKKOR alakú szabályokat). Ha van olyan terv, amely kielégíti/előállítja a célt, akkor már csak a terv előfeltételeit kell (esetleg más tervek betervezésének, majd pedig végrehajtásának eredményeképp) előteremteni.

A BDI alapú rendszerekben a tervek azonban nem puszta szabályok. JADEX-ben például alapértelmezésben tetszőleges Java program lehet terv, amelynek a teljes Hiedelembázishoz van írás/olvasási hozzáférése. Sőt, már maguk a hiedelmek sincsenek valamiféle korlátos kifejezőerejű logikához/nyelvezethez kötve: tetszőleges objektum lehet hiedelem (JADEX-ben).[3] Sőt, mi több, a célok sem puszta tény-konjunkciók. Több különböző fajtájú célt különböztet meg a rendszer (`achieve goal`, `perform goal`, `maintain goal`, stb), amik további eszköztárat adnak az ágens-tervező szakember kezébe. Az előbb elmondottakat szemlélteti a következő ábra.



2. ábra: a JADEX ágensek absztrakt architektúrája

[2] Érdemes felfigyelni erre az elnevezésre. Nem Tudásbázis (Knowledge Base), hanem Hiedelembázis (Belief Base). Már az elnevezés is magában hordozza azt, hogy immár explicite modellezük azt, hogy/ahogy a rendszer (az ágens) modellezi a környezetét/önmagát, és az ezzel kapcsolatos esetleges bizonytalanságokat. Ilyen módon kifejezetten rugalmas, és adaptív intelligens rendszerek hozhatok létre. Természetesen nem csak „strukturált" alapon képzelhetők el intelligens rendszerek/ágensek. Gondoljunk csak például Rodney Brooks alárendelt (subsumption) architektúrájára (lásd. http://en.wikipedia.org/wiki/Subsumption_architecture).

[3] Ennek hátránya nyilván a következtetési mechanizmus bizonyítható teljességének, és optimalitásának elvesztése.

Összességében tehát a JADEX adaptív, intelligens, célvezérelt, mobil ágensek létrehozására ad lehetőséget. Ráadásul az ágensek leírása immár nem egy lefordítandó Java-programban található, hanem egy XML-leírásban, egy úgynevezett **ADF** (**A**gent **D**escription **F**ile) fájlban. Ezeknek a fájloknak kiterjesztése mindig `*.agent.xml`.

A JADEX-es ágensek futtatása egyáltalán nem nyilvánvaló. JADE-ben például a JADEX csomag JADE-Adapter alcsomagja ad erre lehetőséget. A JADEX-es – XML nyelven leírt – ágensek futtatását gyakorlatilag egy `JadeAgentAdapter` nevezetű, interpreter-ágens futtatásával oldhatjuk meg. Paraméterként tetszőleges (megfelelő) JADEX ágens-XML leírást adhatunk neki, plusz az interpretált ágens saját paramétereit.

Lényegében tehát a JADEX is – a JADE-hez hasonlóan – a FIPA szabvány által bevezetett koncepciók egy megvalósítása: vannak ágens-platformok; rajtuk ágensek; akik FIPA-SL tartalomnyelven leírt tartalommal rendelkező **ACL** (**A**gent **C**ommunication **L**angauge) nyelvű üzenetekkel kommunikálnak; van **AMS** (**A**gent **M**anagement **S**ystem), **DF** (**D**irectory **F**acilitator) ágens, és még számos egyéb, alapvető ágens. A lényeg azonban az, hogy a JADEX és a JADE egy „kaptafára” megy, és ezért viszonylag csekély energia befektetéssel összhangba hozható.

2/B. A laborgyakorlat ágensei

A jelenlegi laborgyakorlat során példának okáért JADEX-es, és szokványos JADE-es ágensek fogják benépesíteni a platformot.

A laborgyakorlat elnevezéséhez híven adottak lesznek úgynevezett **játékos-ágensek** (`milab.lab06.PlayerAgent.Player`), **felhasználói-ágensek** (`milab.lab06.UserAgent.UserAgent`), és a központi szerepet betöltő **játék-ágensek** (`milab.lab06.GameAgent.GameAgent`).

Az említett ágensek működése röviden a következő: a `GameAgent` ágens egy adott típusú játékot hostol. Indítás után elkezdi keresni a platformon található játékos és/vagy felhasználói ágenseket. A játékosok és a felhasználók közötti különbség, hogy az előbbi gépi, míg az utóbbi emberi játékost takar. Tehát az utóbbi fajtájú (wrapper) ágens arra szolgál, hogy emberi felhasználók is becsatlakozhassanak a platformon zajló játszmákba.

Miután a `GameAgent` ágens legalább 2 megfelelő ágenst talált, küld feléjük egy REQUEST üzenetet, melyben közli, hogy ki, kivel, és milyen szerepben játszik. A játékosok, miután megkapják ezt az értesítést, egy INFORM üzenettel válaszolnak, amiben közlik többek között a választott stratégiájukat.

A `GameAgent` ágens, miután megkapja az adott körben játszó összes játékostól a választ, összesíti az eredményeket, majd pedig személyre szabottan kihirdeti az eredményt (a kifizetéseket) minden egyes játékos számára: INFORM. A játékosok ennek hatására felülvizsgálják saját belső állapotuk (Belief-Revision, GUI update, stb), majd – amennyiben hasznuk nem csökkent zéró alá, s így nem pusztultak el – újabb játszmára felszólító REQUEST üzenetre várnak.

A `Player` ágenseknek, lévén autonóm módon hozzák meg döntéseiket, különböző típusai lehetnek attól függően, hogy milyen módon választanak játékstratégiát. Az alapértelmezett típusokon felül számos saját típust definiálhatunk. Részben ez is feladata lesz a gyakorlatnak.

Az ágenseknek (mindhárom fajtának) többféle paramétere van/lehet. Ezekről **a függelék 4/H szakasza ad bővebb tájékoztatást**.

2/C. Elméleti tudnivalók

Mielőtt rátérnénk a konkrét feladatokra, célszerű volna még tisztáznunk egy-két elméleti alapfogalmat azok közül, amiket a gyakorlat során „élesben” használni fogunk.

Mit is jelent például a „játék”? – Játékosok összessége? Szabályok összessége?

Esetünkben szerencsére egyáltalán nem ködösek ezek a kifejezések, és habár most a matematikai egzaktságot mellőzve fogjuk tálalni őket, teljesen egyértelmű lesz a jelentésük.

Definíció (Játék): Játéknak nevezzük a játékosok (ágensek) halmazát, a játékosok lehetséges stratégia-halmazainak halmazát, és az egyes játékosokhoz tartozó stratégia-halmazokból képezett Descartes-i szorzatokhoz valós számértéket rendelő haszonfüggvények halmazát egybefogó 3-ast.

Tehát általánosságban van *n* darab játékosunk. Ezekhez rendre tartozik 1-1 stratégia-halmaz, és 1-1 haszonfüggvény. A játékosok stratégia-halmaza tartalmazza azokat a stratégiákat, amiket a játékosok a játék során játszhatnak. Amennyiben minden játékos választ 1-1 stratégiát a saját stratégia-halmazából, egy stratégia-kombináció alakul ki (a stratégia halmazok Descartes-i szorzatának egy eleme). Ezekhez társít egy-egy valós számértéket, egy-egy hasznot a haszonfüggvény. Minden játékosnak van tehát 1-1 saját haszonfüggvénye. Természetesen ideális esetben mind arra törekszenek, hogy ezt a függvényt (azaz saját hasznukat) maximalizálják. Viszont ennek módja egyáltalán nem nyilvánvaló, és épp ez a játékelmélet egyik kulcspontja.

Az egyes játékosok ugyanis csak a saját stratégiájukat tudják megválasztani. Mindenki külön-külön választ stratégiát. Ráadásul egymás választásának ismerete nélkül, mondhatni egyszerre kell meghozniuk a döntést.

Vegyük például a jól ismert Fogolydilemma nevű játékot (lásd. 4/C szakasz). A játékban 2 játékos szerepel, mindkettőnek 2-2 stratégiája van: „Vall”, illetve „Hallgat”. Egymástól függetlenül, mindenféle kommunikáció nélkül kénytelenek stratégiát választani. A játék kimenetele (az egyes játékosok haszna/kifizetése) pedig akkor adódik, mikor már mindketten választottak.

Ha esetleg kicsit erőltetettnek tűnik ez a modell (pl. hogy egyszerre kell választani, a többiek választását nem ismerve), akkor tartsuk szem előtt, hogy egyelőre elvben még vajmi kevés ismerettel rendelkezünk a Játékelméletről, és nem volna célszerű elhamarkodottan képet alkotnunk róla. A Játékelmélet ugyanis, még ilyen alapvető formájában is, számos valós világbeli helyzet leírására/modellezésére alkalmas. A modell, amiről eddig beszéltünk, az úgynevezett „normál alak”. A játékoknak van azonban egy úgynevezett „extenzív alakjuk” is, amely már jóval mélyebb/részletesebb bepillantást enged a stratégiák szerkezetébe, a környezet/játék szabályszerűségeibe, a játékosok által birtokolt ismeretekbe, a játékmenet (azaz játszma) dinamikájába, temporális és logikai felépítésébe, stb. …és mindez még csak a klasszikus játékelmélet alapvető eszköztára. A modern játékelmélet, mint például az evolúciós játékelmélet, és társai, rengeteg hasznos újdonságot hoztak be a modellbe.

Most azonban elégedjünk meg a „normál alakkal”. Ráadásul a laborgyakorlat során csak 2-szereplős játékokkal fogunk foglalkozni, amikben ráadásul mindkét játékosnak csak és kizárólag 2-2 stratégiája van. Tehát a lehető legegyszerűbb játékokat fogjuk célba venni, és ezeken próbáljuk meg bemutatni a Játékelmélet, illetve az evolúciós játékelmélet egy-két frappáns, érdekes, és hasznos vonatkozását.

Példának okáért itt van a Fogolydilemma. Miért dilemma ez? – Próbáljunk meg belegondolni, hogy mit tennénk, ha valamely játékos helyében volnánk! Bizonyára érezhető, hogy a döntés (hogy milyen stratégiát válasszunk) egyáltalán nem egyszerű, még ilyen egyszerű esetben sem.

A Fogolydilemma esetén látható, hogy ha mindketten vallunk, akkor rosszabbul járunk, mint ha mindketten hallgatnánk, nyilván. Mégis, megkockáztatnánk vajon a hallgatást? Akár 10 évünkbe is kerülhet (elvben).

A Fogolydilemmában a legésszerűbb megoldás az, ha az ember/játékos/ágens vall. „De mégis milyen értelemben ésszerű?" – tehetnénk fel a kérdést. A Játékelmélet egyik legfontosabb mondanivalója éppen ez: mit tekintünk racionálisnak? Mikor racionális egy ágens/játékos? Attól függ, hogy mit cselekszik, vagy attól, hogy hogyan dönt arról, hogy mit cselekszik? Esetleg a kettő elegye?...

A félreértések elkerülése céljából épp ezért többféle racionalitás-definíció létezik. A Játékelméletben igazából a stratégia-választás elvét tekintjük racionalitási alapelvnek. Ezekből kívánunk most néhányat bemutatni, mivel a gyakorlat során is elő fognak kerülni.

Definíció (Domináns stratégia): egy játékos valamely stratégiáját dominánsnak nevezzük, ha a többi játékos döntésétől függetlenül, minden esetben jobb eredményt (magasabb kifizetést) ad, mint a játékos többi stratégiája (minden adott esetben).

A domináns stratégiák definíciója, így első olvasatra egészen kézenfekvőnek tűnik. …és hát, valljuk be, az is. ☺ Így hát válaszoljuk is meg gyorsan a következő kérdést: van-e *Fogolydilemmában valamely játékosnak domináns stratégiája? Van-e olyan stratégiája valamely játékosnak, amely mindig jobb eredményt ad, mint a többi?*

Tekintsük a többi játékos összes lehetséges stratégia-kombinációját, és nézzük meg, hogy van-e egy olyan stratégiánk, amely minden esetben jobb, mint a többi! Ha van ilyen stratégia, akkor bízvást racionálisnak nevezhetjük, igaz? …mert hiszen mindig jobb eredményt ad, mint a többi.

Sajnos azonban ez egyáltalán nem ilyen egyértelmű. A domináns stratégiák választásának elve igazából nem veszi figyelembe a többi játékos hasznát (sem önző módon, sem pedig önzetlenül), márpedig a többi játékos döntésétől is függ, hogy mennyi lesz a hasznunk! Foglalkoznunk kell tehát az ő motivációikkal is (a sajátjainkon túl). A többi játékos motivációját pedig lényegében a különböző kimenetelekhez rendelt haszonértékük határozza meg.

Definíció (Nash-egyensúly): Nash-egyensúlynak nevezzük azt a stratégia-kombinációt, amelytől egyik játékosnak sem érné meg egyoldalúan eltérnie.

Tehát itt arról van szó, hogy egy olyan cellát keresünk a 2-szereplős játék táblázatában, amelyre teljesül, hogy olyan kifizetések szerepelnek benne az egyes játékosok számára, amiknél nem kaphatnak nagyobb értéket azok a játékosok, akik egyedül eltérnek egyensúlyi stratégiájuktól.[4]

Természetesen, ha többen összefognak, és együttesen változtatnak stratégiát, az már más kérdés. Nézzük csak meg újra a Fogolydilemmát! Hol van ott Nash-egyensúly? A táblázat mely mezőjére teljesül, hogy onnan már senkinek sem érné meg eltérnie (ha az a kimenetel adódna egy lejátszásban)?

Rövid gondolkodás után beláthatjuk, hogy a „Vall-Vall" stratégia-kombináció által azonosított esetben teljesül a Nash-egyensúly feltétele. Ha ebben az esetben bármely játékos eltérne, azaz stratégiát változtatna, nem járhatna jobban (sőt, rosszabbul járna).

Ha viszont együttesen változtathatnának stratégiát (esetleg valamiféle közösen osztott irányelv (pl. betyárbecsület, altruizmus, önzetlenség, Kant-i kategorikus imperatívusz) miatt, vagy előzetes egyeztetés nyomán, kölcsönös bizalomra építve), akkor akár még az összességében legjobb, „Hallgat-Hallgat" kimenetelnél is kiköthetnének.

[4] A Nash-egyensúly, főleg ha unikális, tehát olyan, mint egy „önbeteljesítő jóslat"…

Miért nem kötnek ki ott? Miért vallunk, ha hallgathatnánk is? Tényleg annyira kockázatos?

A Fogolydilemma, mint látjuk, nem kooperatív játék. Kizárja a kooperációt. Alapfeltevése, hogy a játékosok nem kooperálnak, és mindenki egyéni hasznának maximalizálására törekszik. Ebből kifolyólag tehát nem meglepő, ha mindkét játékos a domináns stratégiáját választja, és vall…

Definíció (Pareto optimum): Pareto-optimumnak nevezzük azt a stratégia-kombinációt, amelytől már senkinek sem érné meg egyoldalúan eltérnie úgy, hogy közben más nem jár rosszabbul.

A Pareto-optimum, mint elv, tehát már bizonyos fokig kooperatív. Látható, hogy a többiek helyzetével kapcsolatos „empátia” is megjelenik benne. A Fogolydilemma esetén is van ilyen stratégia-kombináció: ha mindkét játékos hallgat.

Eddig mindvégig csak „egylövetű” játékokkal foglalkoztunk. Mi lenne, ha egy játékot adott esetben többször is játszanánk valakivel? Mi lenne, ha nem csak egy játszma lenne egymás után, hanem több? Befolyásolná ez a döntésünket, vagy minden játszmában, vakon ugyanúgy cselekednénk?

Nyilván nem. Az ismételt játékok (repeated games) tanulmányozása is fontos része a klasszikus Játékelméletnek. Számos esetben előfordul, hogy egy játékot ismételt módon, egymás után többször is lejátszunk valakivel, és bizony akkor érdemes figyelembe venni azt, hogy mi történt (ki-mikor-mit tett) előtte.

Definíció (Szemet-szemért – Tit for Tat – stratégia): ismételt játékok esetén azt nevezzük szemet-szemért stratégiának, mikor egy játékos kezdetben kooperál, majd a következő körökben mindig azt cselekszi, amit ellenfele tett az előző körben.

A „Tit for Tat” stratégia, a maga egyszerűsége ellenére, hatalmas adaptivitást biztosíthat az előbbiekben taglalt „statikus” irányelvekhez képest (az ismételt játékokban). Érdemes megfontolni!

De miért is beszélünk mi most itt ismételt játékokról? – Nyilván azért, mert a laborgyakorlaton is fogunk velük foglalkozni. A gyakorlat során, mint ahogy már a 2/B szakaszban is olvashattuk, egy evolúciós szcenárióba helyezzük az ismételt játékokat. A koncepció alapötletét és hátterét a 4/B szakaszban ismertetjük. Ezt **mindenképp olvassuk el**, mielőtt tovább folytatnánk ezt a részt!

A „magasabb” játék tehát körökre van osztva. Minden körben véletlenszerűen, egyenletes eloszlás szerint párokat képezünk az ágens-populáció egyedeiből. A párok ezek után 1-1 két-szereplős játékban vesznek részt egymás ellen/egymással. A játszma végeztével az abból eredő kifizetésükkel növelik saját aggregált hasznosságuk (ami kezdetben leginkább zérus).

Ha valakinek a haszna valamely körben nulla alá csökken, úgy terminálódik. Ha viszont valamely játékosnak már elég számottevő hasznot sikerült felhalmoznia, úgy reprodukálódhat. A reprodukció nyomán (amely, mint látjuk, olyan mint a sejtek/baktériumok osztódása: aszexuális) a szülő egyed haszna a reprodukció költségével csökken, miközben egy új, a szülő mintájára paraméterezett egyed jön létre.

Ha tovább göngyölítjük ezt a fonalat, láthatjuk, hogy végső soron a 4/B szakaszban bemutatott absztrakt modell egyfajta speciális implementációjáról van szó. Ez természetesen nem jelenti azt, hogy az implementáció megfelel a modellnek. Például miért is felelne meg ez a nyílt végű, nem felügyelt evolúciós folyamat (szaporodás és kiválasztódás) a 4/B szakaszban tárgyalt replikátor dinamikának?

3. Feladatok

1. Indítsunk el paraméterek nélkül 2 darab `milab.lab06.UserAgent.UserAgent`, és 1 darab `milab.lab06.GameAgent.GameAgent` ágenst! Ismerkedjünk meg az ágensek kezelői felületével, értelmezzük a működésüket, majd pedig kérjük meg csoporttársunkat (ha nincs, akkor a mellettünk ülőt), hogy játsszon velünk néhány kört/játszmát. A játék során az egyik fél rejtse el a másik elől, hogy mit lép! Csak akkor derüljön ki, hogy ki-mit lépett, mikor már mindketten léptek! Továbbá próbálják meg visszaidézni a 2/C szakaszban leírt elméleti alapvetéseket, és kapcsolatba hozni őket a jelenlegi játszmákkal.

2. Fogadjanak más csoportok más gépeiről érkező `UserAgent` ágenseket, és azokat is vonják be a játékba. Mi történik, ha (esetleg távolról) több `UserAgent` ágens is belép a játékba?

3. Vonjunk be a játékba különböző típusú `milab.lab06.PlayerAgent.Player` ágenseket! A játékosok típusának (`myType` paraméter) megadásakor az integer-értékeket a `milab.lab06.PlayerType` osztály konstansai közül böngésszük ki. De legyünk tekintettel arra, hogy kezdetben nem minden típus elérhető (nem mind van implementálva)! Tekintsük meg a `milab.lab06.PlayerAgent.PlayerPlayPlan.chooseStrategy()` metódusban található SWITCH-et! Mit tesz ez a metódus?

4. Próbáljuk meg több, különböző (implementált!) típusú gépi ellenfél ellen felvenni a harcot! Melyik ellen milyen stratégia bizonyult hatékonynak, mikor/mikor nem, és miért?

5. Próbálkozzunk más csoportok távoli platformjára is indítani `Player` ágenseket (egyszerre akár többet is, akár különböző fajtából)! Milyen tapasztalatokat sikerült leszűrnünk? Melyik típus melyik típus ellen tudja/nem tudja felvenni a harcot, mikor/mikor nem, és miért?

6. Tekintsük meg a `milab.lab06.Game` osztály konstruktorában található SWITCH-et. Mire szolgál? Hozzuk létre – a függelékben olvashatóaknak megfelelően – a SWITCH-ben nem implementált játékokat (5 darab)! A létrehozás előtt alaposan értsük meg a konstruktorban szereplő adatstruktúrákat, és használatuk módját.

7. Teszteljük az újonnan kialakított játékokat először csupán `UserAgent` ágensekkel.

8. Vonjunk be a tesztelésbe `Player` ágenseket is!

9. A Héja-Galamb játék esetében próbálkozzunk meg a *V (Value)* és *C (Cost)* paraméterek hangolgatásával. Mit tapasztalunk, mi változik ennek a hatására a játékosok teljesítményét tekintve?

10. Implementáljuk a `PlayerPlayPlan.chooseStrategy()` metódusban lévő SWITCH-ben nem implementált típusoknak megfelelő algoritmusokat: **(1)** Nash-egyensúly számítása 2 szereplő és 2-2 stratégia esetén (ügyeljünk arra, hogy több egyensúly is lehet, de az is lehet, hogy egy sincs!!), továbbá **(2)** Pareto-optimum számítása 2 szereplő és 2-2 stratégia esetén. Az utóbbi esetben is legyünk tekintettel a megoldások számára.

11. Teszteljük a kidolgozott új típusainkat (akár az eddigiekkel összevetésben) új `Player` ágensek létrehozásával.

12. Végezzünk kísérleteket `Player` ágensek felhasználásával különböző, 4/H szakaszban leírtaknak megfelelő paraméterezések esetén! Mik például a reprodukcióra vonatkozó egyéb megkötések hatásai? Mennyire számít, hogy egy-egy adott típusú ágensből hány darab volt kezdetben? Mi a többi paraméter hatása? Állítsuk át továbbá a `milab.lab06.PlayerAgent.PlayerBRFPlan.reproduction_type` változót is! Mi ennek a hatása?

    Megjegyzés: a `GameAgent` *ágens leállítását követően, az* `RMA` *ágens GUI-ján egy-egy konténerre kattintva, és azon kiadva a „Kill” utasítást, igen hatékonyan ki tudjuk iktatni a felesleges, esetleg nagyszámú ágenseket. A konténerek kiiktatását követően ne felejtsük el Eclipse-ben is becsukni az immár leállított Console ablakokat.*

13. Dolgozzon ki egy saját játékos típust! Definiálja a `PlayerType` osztályban, szerepeltesse a `PlayerType.getTypeColor()` metódusban, majd implementálja a `PlayerPlayPlan.chooseStrategy()` metódusban! Kiindulásképp a `PlayerType.TIT4TAT` típust vegye alapul, és próbáljon meg ennél jobbat létrehozni!

14. Amennyiben maradt még ideje, szorgalmi feladatként megpróbálkozhat saját – 2 szereplős, 2-2 stratégiát tartalmazó – játékok kigondolásával, és létrehozásával!

4. Függelék

4/A. Héja-Galamb (Hawk-Dove) játék

Adott a Héja-Galamb-játék, amelynek keretében a játékosok egy adott erőforrásért folytatott harcban vagy héjaként (agresszíven) vagy galambként (békésen) viselkedhetnek. Az erőforrás értéke V. A galambok békés úton, egyenlő arányban osztják el egymás közt az erőforrást, míg a héják harcolnak érte. A harc – legalább zéró – költsége C. Ebből a következően az alábbi esetek/kifizetések adódnak: (H – héja; G – galamb):

- Amennyiben egy *héja* típusú játékos egy *héjával* találkozik, úgy hasznuk fejenként: (V-C)/2
- Amennyiben egy *héja* típusú játékos egy *galambbal* találkozik, úgy a héja mindent visz (haszna V), a galambnak pedig nem marad semmi (haszna 0).
- Amennyiben egy *galamb* típusú játékos találkozik egy *héjával*, úgy hasznuk épp ugyanaz, mint előbb, csak fordítva (mivel a játék szimmetrikus).
- Amennyiben egy *galamb* típusú játékos egy *galambbal* találkozik, úgy hasznuk fejenként: V/2

Mindez bimátrix (avagy táblázatos) formában a következőképp fest.

		II. játékos	
		Héja	Galamb
I. játékos	Héja	$\left(\frac{V-C}{2};\frac{V-C}{2}\right)$	$(V;0)$
	Galamb	$(0;V)$	$\left(\frac{V}{2};\frac{V}{2}\right)$

Legyen $C < V$, akkor a kifizetések sorrendje: $0 < \frac{V-C}{2} < \frac{V}{2} < V$. Ez éppen a **Fogolydilemmának** felel meg (lásd. 4/C). Ha viszont $C > V$, akkor a kifizetések sorrendje: $\frac{V-C}{2} < 0 < \frac{V}{2} < V$. Ez leginkább a **Gyáva Nyúl** játékhoz hasonlít (lásd. 4/D).

4/B. Evolúciós játékelmélet: példa

Legyen adott egy nagy komplexitású, azaz nagyszámú alkotóelemből álló rendszer. Az egyes alkotóelemek ágensek (játékosok), és így a rendszer ágensek populációja. Tegyük fel, hogy ennek a populációnak *N* páros számú komponense van, továbbá *N* tart a végtelenhez.

A populáció legyen 2 részre osztható, *C*-re és *D*-re (kooperálók, és nem-kooperálók). A *C* és *D* részhalmazok számosságának arányát a halmaz egészéhez képest rendre p_c és p_d jelölje. A kiindulási arányokra teljesüljön a következő.

(0) $p_c^{(1)} + p_d^{(1)} = 1$ és $0 < p_c^{(1)} < 1$, illetve $0 < p_d^{(1)} < 1$.

A játékot innentől kezdve körökre osztjuk. Minden körben egyenletes eloszlás szerint, véletlenszerűen összepárosítjuk a populáció tagjait, és a párokat egy-egy 2-személyes játéknak tesszük ki. $\Im_1(c,c)$ jelölje az 1. játékos hasznát, ha mindkét játékos kooperatívan (C) cselekszik. A második játékos haszna ebben az esetben, általánosságban $\Im_2(c,c)$. Általánosságban ez a következő. Hasonlóképp mind a 4 kimenetel esetében megadható az egyes játékosok haszna. Táblázatos formában, összefoglalva mindezek a kifizetések a következőképp festenek.

2-es játékos / **1-es játékos**	$s_2 = d$	$s_2 = c$
$s_1 = d$	$(\Im_1(d,d), \Im_2(d,d))$	$(\Im_1(d,c), \Im_2(d,c))$
$s_1 = c$	$(\Im_1(c,d), \Im_2(c,d))$	$(\Im_1(c,c), \Im_2(c,c))$

Most pedig specializáljuk tovább a játékot. Először is tegyük fel, hogy szimmetrikus, azaz

(a) $\Im_1(c,c) = \Im_2(c,c) = \Delta\Im(c,c)$

(b) $\Im_1(c,d) = \Im_2(d,c) = \Delta\Im(c,d)$

(c) $\Im_1(d,c) = \Im_2(c,d) = \Delta\Im(d,c)$

(d) $\Im_1(d,d) = \Im_2(d,d) = \Delta\Im(d,d)$

Ekkor a játék mátrixa a következő formában is felírható:

2-es játékos / **1-es játékos**	$s_2 = d$	$s_2 = c$
$s_1 = d$	$(\Delta\Im(d,d), \Delta\Im(d,d))$	$(\Delta\Im(d,c), \Delta\Im(c,d))$
$s_1 = c$	$(\Delta\Im(c,d), \Delta\Im(d,c))$	$(\Delta\Im(c,c), \Delta\Im(c,c))$

Ezek után tegyük fel, hogy a következő egyenlőtlenségek is teljesüljenek (a kifizetésekre).

(1) $\Delta\Im(d,c) > \Delta\Im(c,c) > \Delta\Im(d,d) > \Delta\Im(c,d)$

A kifizetések sorrendje jól láthatóan éppen a Héja-Galamb játéknak (lásd. 4/A) felel meg abban az esetben, amikor a harc által okozott veszteség kisebb, mint az erőforrás haszna (és így „elvileg” megéri harcolni), ez pedig végső soron a Fogolydilemma (lásd. 4/C).

Próbáljuk most megjósolni, hogy mi lesz az előbbiekben definiált játék alakulása a körök előrehaladtával! Vezessük be a következő jelöléseket. A csoportok (*C* és *D*) átlag-jóságát az *(i+1).* körben – kihasználva a játék szimmetriáját – a következő rekurzív képlettel kapjuk.

(2) $W_c^{(i+1)} = W_c^{(i)} + p_c^{(i)} \cdot \Delta\Im(c,c) + p_d^{(i)} \cdot \Delta\Im(c,d)$

(3) $W_d^{(i+1)} = W_d^{(i)} + p_c^{(i)} \cdot \Delta\Im(d,c) + p_d^{(i)} \cdot \Delta\Im(d,d)$

Tehát, például a (2)-es egyenletet tekintve, arról van szó, hogy a *C* csoport *i.* körben vett átlagos haszna $W_c^{(i)}$. A két csoport aránya ugyanekkor rendre $p_c^{(i)}$, és $p_d^{(i)}$. Annak a valószínűsége tehát, hogy egy *C* csoport beli játékos egy másik, *C* csoport beli játékossal találkozik $p_c^{(i)}$. *D* csoport beli játékossal ugyanez $p_d^{(i)}$. Az előbbi esetben a – C csoport beli –

játékos haszna $\Delta\Im(c,c)$, míg az utóbbi esetben $\Delta\Im(c,d)$. A (3)-as egyenlet ugyanez, csak a *D* csoport beli játékosok szemszögéből nézve.

Mivel (1) alapján tudjuk, hogy $\Delta\Im(d,c)>\Delta\Im(c,c)$ és $\Delta\Im(d,d)>\Delta\Im(c,d)$, ezért adott $p_c^{(i)}$ és $p_d^{(i)}$ mellett, ha $W_c^{(i)}=W_d^{(i)}$, akkor (2) és (3) alapján $W_d^{(i+1)}>W_c^{(i+1)}$ adódik. Tegyük fel tehát, hogy kezdetben $W_c^{(1)}=W_d^{(1)}$ teljesül!

Ebből tehát már következik, hogy $W_d^{(2)}>W_c^{(2)}$. No, de mi lesz később? Mitől függ ez? - Hát nyilván attól, hogy a részpopulációk aránya (azaz $p_c^{(i)}$ és $p_d^{(i)}$) idővel miképpen változik.

Tegyük fel, hogy a részpopulációk aránya átlagos hasznosságukkal egyenes arányban változik körről körre. Ezt nevezik **Replikátor Dinamiká**nak, ami képletekben a következőképp foglalható össze.

(4) $$p_c^{(i+1)}=p_c^{(i)}\cdot\frac{W_c^{(i)}}{\overline{W^{(i)}}},\text{ és}$$

(5) $$p_d^{(i+1)}=p_d^{(i)}\cdot\frac{W_d^{(i)}}{\overline{W^{(i)}}}$$

ahol

(6) $$\overline{W^{(i)}}=p_c^{(i)}\cdot W_c^{(i)}+p_d^{(i)}\cdot W_d^{(i)}$$

a teljes populáció átlag hasznát jelöli az *i.* körben. Ebből már felírható a differenciál, azaz a részpopulációk változásának mértéke. Írjuk fel a (4)-es egyenletet az *i+1.* és *i.* időpontban, majd vonjuk ki a kettőt egymásból! A következőt kapjuk.

(7) $$\Delta p_c^{(i)}=p_c^{(i+1)}-p_c^{(i)}=p_c^{(i)}\cdot\frac{W_c^{(i)}}{\overline{W^{(i)}}}-p_c^{(i)}=p_c^{(i)}\cdot\frac{W_c^{(i)}}{\overline{W^{(i)}}}-p_c^{(i)}\cdot\frac{\overline{W^{(i)}}}{\overline{W^{(i)}}}=p_c^{(i)}\cdot\frac{W_c^{(i)}-\overline{W^{(i)}}}{\overline{W^{(i)}}}$$

Hasonlóképp járva el az (5)-ös egyenlettel, a következő adódik.

(8) $$\Delta p_d^{(i)}=p_d^{(i)}\cdot\frac{W_d^{(i)}-\overline{W^{(i)}}}{\overline{W^{(i)}}}$$

Az előzőekben már láthattuk, hogy tetszőleges – a (0)-ás feltételt kielégítő – $p_c^{(1)}$ és $p_d^{(1)}$ kiindulási részarányok mellett, ha kezdetben $W_c^{(1)}=W_d^{(1)}$, akkor az (1)-ben megadott hasznosságok miatt $W_d^{(2)}>W_c^{(2)}$, pontosabban $W_d^{(2)}>\overline{W^{(2)}}>W_c^{(2)}$ adódik. Mivel $W_d^{(2)}>W_c^{(2)}$, ezért (2) és (3) alapján $W_d^{(3)}>W_c^{(3)}$, pontosabban $W_d^{(3)}>\overline{W^{(3)}}>W_c^{(3)}$ is igaz lesz, e miatt azonban $W_d^{(4)}>\overline{W^{(4)}}>W_c^{(4)}$ is igaz lesz, sit., ad infinitum. Tehát egészen egyszerűen indukcióval következik, hogy

(9) $W_d^{(i)}>\overline{W^{(i)}}>W_c^{(i)}$ teljesül $\forall i>1$ esetén.

Ebből azonban (7) és (8) alapján az következik, hogy

(10) $\Delta p_d^{(i)}>0$ és $\Delta p_c^{(i)}<0$ teljesül $\forall i>1$-re.

Tudva, hogy $p_c^{(i+1)}=p_c^{(i)}+\Delta p_c^{(i)}$ és $p_d^{(i+1)}=p_d^{(i)}+\Delta p_d^{(i)}$, végeredményben azt kapjuk, hogy

(11) $$\lim_{i\to\infty}p_c^{(i)}=0\text{ és }\lim_{i\to\infty}p_d^{(i)}=1$$

A (11) jelentése: a **populáción belül idővel elszaporodnak a nem kooperáló egyedek, a kooperáló egyedek pedig teljességgel kivesznek**. Ez tehát ennek a játéknak az „evolúciós egyensúlya”, avagy a nem-kooperálás az úgynevezett **ESS** (**E**volúciósan **S**tabil **S**tratégia).

Mindez persze csak akkor igaz, ha igazak az előfeltevéseink (pl. a részpopulációk aránya valóban a Replikátor Dinamikának megfelelően alakul körről-körre).

4/C. Fogolydilemma (Prisoners' Dilemma) játék

Egy súlyos bűntény kapcsán két gyanúsítottat tartóztat le a rendőrség. Mivel nem áll rendelkezésre elegendő bizonyíték a vádemeléshez, ezért elkülönítik őket egymástól és mindkettejüknek ugyanazt az ajánlatot teszik. Amennyiben az első fogoly vall és társa hallgat, akkor az előbbi büntetés nélkül elmehet, míg a másik, aki nem vallott, 10 év börtönt kap. Ha az első tagadja meg a vallomást és a második vall, akkor a másodikat fogják elengedni és az első kap 10 évet. Ha egyikük sem vall, akkor egy kisebb bűntényért fejenként 6 hónapot kapnak, ha viszont mindketten vallanak, fejenként 6 évet kapnak.

A játék szimmetriáját, és a kimenetelek hasznosságának sorrendjét szem előtt tartva, a következő bimátrix reprezentáció adható.

		II. játékos	
		Vall	Hallgat
I. játékos	Vall	(-2, -2)	(3, -3)
	Hallgat	(-3, 3)	(2, 2)

A „Vall" stratégiát versengőnek tekintjük, míg a „Hallgat"-ot kooperatívnak.

4/D. Gyáva Nyúl (Chicken) játék

Két autós száguld egymással szemben, és az veszít, aki elrántja a kormányt.[5] Nézzük a dolgot az ő szempontjukból: *„A legrosszabb eset nyilván az, ha **(1)** mindketten konfrontálunk, ekkor ugyanis összecsattanunk. Ennél jobb, ha **(2)** elrántom a kormányt, miközben a másik nem rántja el. Ekkor ugyanis életben maradok, habár a másik győz. Ennél jobb, ha **(3)** a másik is elrántja a kormányt, hiszen ekkor legalább nem arat győzelmet fölöttem. A legjobb eset pedig az, ha **(4)** a másik elrántja a kormányt, miközben én nem rántom el, és így én győzök."*

A játék szimmetriáját, és hasznosságainak sorrendjét szem előtt tartva, a következő bimátrix reprezentáció adható.

		II. játékos	
		Vakmerő	Gyáva
I. játékos	Vakmerő	(-10, -10)	(1, -1)
	Gyáva	(-1, 1)	(0, 0)

[5] A játék a *„Chicken Game"* elnevezést a James Dean főszereplésével készült, legendás *„Haragban a Világgal (Rebel Without a Cause)"* c. filmből kapta (lásd. http://www.youtube.com/watch?v=LGUYsuYudVA).

4/E. Nemek Harca (Battle of Sexes) játék

Képzeljünk el egy házaspárt. A férj focimeccsre szeretne menni, míg a feleség operába. Viszont ennél fontosabb számukra, hogy mindketten jobban szeretnének együtt lenni, mint külön-külön. Sajnos azonban már nem tudnak egyeztetni. Dönteniük kell, hogy hová mennek: focimeccsre, vagy operába. Mit tegyenek?

A játék asszimetriáját, és hasznosságainak sorrendjét szem előtt tartva, a következő bimátrix reprezentáció adható.

		Feleség	
		Opera	Focimeccs
Férj	Opera	(1, 2)	(-1, -1)
	Focimeccs	(0, 0)	(2, 1)

Amennyiben a két különböző játékos egyéni érdekre való törekvését versengésnek, ellenkezőjét pedig kooperációnak tekintjük (azaz pl. a Férj esetében a „Focimeccs” versengés, míg a Feleség esetében ugyanez együttműködés), úgy a játék a sorok és oszlopok megfelelő átrendezésével a következő, szimmetrikus alakra hozható:

		II. játékos	
		Verseng	Kooperál
I. játékos	Verseng	(0, 0)	(2, 1)
	Kooperál	(1, 2)	(-1, -1)

4/F. Vezérürü (Leader) játék

Adott egy egyirányú út, és rajta egy kereszteződés, ahol két autós áll egymással szemben, és épp arra készül, hogy felhajtson az egyirányú útra. Így gondolkoznak: „Ha **(1)** mindketten elindulunk, akkor összeütközünk (ez a legrosszabb eset). Ennél azért egy fokkal jobb, ha **(2)** egyikőnk sem indul el, bár úgy időtlen időkig ebben a kereszteződésben rostokolnánk... Ha azonban **(3)** én várnék, és a másik indulna előbb, úgy legalább előbb-utóbb feljutnék az útra. A legjobb persze az volna, ha **(4)** én indulnék előbb, és a másik várna."

A játék szimmetriáját, és hasznosságainak sorrendjét szem előtt tartva, a következő bimátrix reprezentáció adható.

		II. játékos	
		Megy	Vár
I. játékos	Megy	(-1, -1)	(2, 1)
	Vár	(1, 2)	(0, 0)

Ugyanez a dilemma más megfogalmazásban (Mérő László *„Minden Másképp Egyforma"* c. könyvének 87. oldalán) a következő:. „Két szuperjólnevelt ember egymást tessékeli előre egy ajtóban. A versengés – ebben az esetben – az a stratégia, ha ragaszkodunk ahhoz, hogy a másik menjen ki először. Kooperál az, aki vállalva annak ódiumát, hogy a másik megveti udvariatlanságáért, hajlandó előre menni. A legrosszabb eset a kölcsönös versengés, mert akkor ott halnak éhen az ajtó előtt. Ennél egy fokkal jobb, ha kooperálnak, és összeütköznek az ajtóban, de legalább némi gyűrődés árán átjutnak. Ha az egyik játékos verseng (udvariaskodik), a másik kooperál (mondjuk, hogy udvariatlan), akkor mindketten simán átjutnak, de a versengő játékos valamivel jobban jár, mert plusz nyereségként amellett, hogy viszonylag gyorsan kijutott, még meg is vetheti partnerét a neveletlenségéért."

Láthatjuk, hogy Mérő féle megfogalmazásban az „Udvarias" (versengő) stratégia felel meg az eredeti megfogalmazásban a „Megy" stratégiának, stb. Nyilván egy-egy formálisan adott játéknak tetszőleges számú informális megfelelője adható.

4/G. Érmepárosítás (Matching Pennies) játék

Két játékos, akik mindketten rendelkeznek egy-egy pénzérmével. A pénzérmét egymás elől rejtve fejre, vagy írásra állítják, majd pedig egyszerre felmutatják az érméjüket. Ha mindkét érme „paritása” azonos (azaz fej-fej, vagy írás-írás), akkor az I. játékos nyer, a II. veszít. Ha a két érme különböző, akkor pedig a II. játékos nyer, és az első veszít.

A játék asszimmetriáját, és hasznosságainak sorrendjét szem előtt tartva, a következő bimátrix reprezentáció adható.

		II. játékos	
		Fej	Írás
I. játékos	Fej	(1, -1)	(-1, 1)
	Írás	(-1, 1)	(1, -1)

4/H. A laborgyakorlat ágenseinek indítása és paraméterezése

Amint már a bevezető 2/B szakaszában is említettük, 3 különböző fajtájú ágenssel lesz dolgunk a laborgyakorlat során:

1. **Játék-ágens:** `milab.lab06.GameAgent.GameAgent`
2. **Játékos-ágens:** `milab.lab06.PlayerAgent.Player`
3. **Felhasználói-ágens:** `milab.lab06.UserAgent.UserAgent`

Az `GameAgent`, és a `UserAgent` ágens szokványos JADE-es ágens, tehát indításuk is ennek megfelelő. A következő – pirossal jelölt nevű – paramétereik vannak:

- `GameAgent`
  - **Játék azonosító száma**
    - Típusa: `int`
    - Opcionális (de kötelező, ha a második argumentumot is meg szeretnénk adni)
    - Alapértelmezett értéke (ha nincs megadva): `Game.HAWK_DOVE` konstans
    - Értékkészlete: lásd. a `milab.lab06.Game` osztály konstansait
    - Leírás: *arra szolgál, hogy a `GameAgent` ágens induláskor automatikusan be tudja azonosítani, hogy milyen típusú játékot kell host-olnia.*
  - **Játékosok maximális száma (körönként)**
    - Típusa: `int`
    - Opcionális (de csak az első argumentummal együtt lehet megadni)
    - Alapértelmezett értéke (ha nincs megadva): `500`
    - Értékkészlete: tetszőleges pozitív egész szám (legalább 2)
    - Leírás: *arra szolgál, hogy a `GameAgent` ágens tudja, hogy a `DF` ágens által egyes körökben visszaadott, potenciális játékosokat tartalmazó találatok közül legfeljebb mennyit választhat ki (véletlenszerűen, egyenletes eloszlás szerint) párba-rendezés és játék céljából. Tehát legfeljebb annyi játékos lesz körönként, amennyit itt megadunk.*
  - **Grafikus felhasználói felület (GUI) típusa**
    - Típusa: `int`
    - Opcionális (de csak az első argumentummal együtt lehet megadni)
    - Alapértelmezett értéke (ha nincs megadva): `GameAgentGui.GUI_TYPES_RATIO` konstans
    - Értékkészlete: lásd. a `milab.lab06.GameAgent.GameAgentGui` osztály `GUI_...` kezdetű konstansait
    - Leírás: *azt határozza meg, hogy a `GameAgent` ágens miképpen, milyen fajtájú diagram formájában jeleníti meg az evolúció menetét.*

Példa egy `GameAgent` ágens indítására:

```
java jade.Boot -container ga:milab.lab06.GameAgent.GameAgent(0 800 3)
```

- `UserAgent`
  - **Játék azonosító száma**
    - Típusa: `int`
    - Opcionális (de kötelező, ha a második argumentumot is meg szeretnénk adni)
    - Alapértelmezett értéke (ha nincs megadva): `Game.HAWK_DOVE` konstans
    - Értékkészlete: lásd. a `milab.lab06.Game` osztály konstansait

- Leírás: *arra szolgál, hogy a* `GameAgent` *ágens induláskor automatikusan be tudja azonosítani, hogy milyen típusú játékban vesz részt.*

- **Kezdeti haszon**
  - Típusa: `double`
  - Opcionális (de csak az első argumentummal együtt lehet megadni)
  - Alapértelmezett értéke (ha nincs megadva): `0.0`
  - Értékkészlete: zérusnál nem kisebb valós számok
  - Leírás: *az ágens kezdeti haszna.*

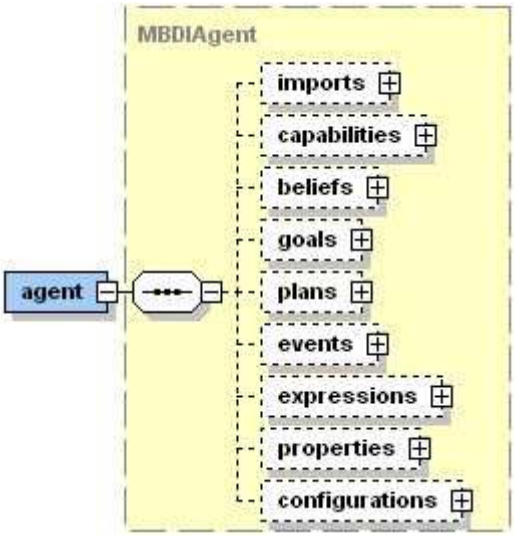
Példa egy `UserAgent` ágens indítására:

```
java jade.Boot -container ua:milab.lab06.UserAgent.UserAgent(0 1.2)
```

Amint fentebb is látható volt, a `Player` ágens már nem egy szokványos JADE-es ágens, hanem egy JADEX-es ágens, melynek a

```
\jade\src\milab\lab06\PlayerAgent\Player.agent.xml
```

fájl tartalmazza a leírását. Ennek a leírásnak lényegében 9 főbb (jórészt opcionális) eleme van. Ezeket sorolja fel a következő ábra.



3. ábra: a JADEX ágenseket leíró ADF fájlok vázlatos XML sémája (XSD-je)

Az ADF-ek felépítéséről, és a JADEX-ről bővebben a következő elérésen olvashatunk:

`http://vsis-www.informatik.uni-hamburg.de/projects/jadex/jadex-0.96x/doc_overview.php`

Most térjünk vissza a JADEX-es `Player` ágens paraméterezésének, és indításának ismertetésére. Összesen/legfeljebb 7 darab paramétert adhatunk meg. Ezek a következők.

- `Player`
  - **G**
    - Típusa: `int`
    - Teljesen opcionális
    - Alapértelmezett értéke (ha nincs megadva): `Game.HAWK_DOVE` konstans
    - Értékkészlete: lásd. a `milab.lab06.Game` osztály konstansait

- Leírás: *arra szolgál, hogy a* `Player` *ágens induláskor automatikusan be tudja azonosítani, hogy milyen típusú játékban fog részt venni. Mondhatni, ez a játékról alkotott modellje/hiedelme.*

- **`myType`**
  - Típusa: `int`
  - Teljesen opcionális
  - Alapértelmezett értéke (ha nincs megadva): `PlayerType.RANDOM` konstans
  - Értékkészlete: lásd. a `milab.lab06.PlayerType` osztály konstansait
  - Leírás: *arra szolgál, hogy megadjuk, hogy játék közben milyen elven válasszon stratégiát a* `Player` *ágens.*
- **`utility`**
  - Típusa: `double`
  - Teljesen opcionális
  - Alapértelmezett értéke (ha nincs megadva): `0.0`
  - Értékkészlete: tetszőleges zérusnál nem kisebb valós szám
  - Leírás: *arra szolgál, hogy megadjuk, hogy a játékba történő belépéskor, az első kör kezdetén milyen haszonnal rendelkezzen a* `Player` *ágens.*
- **`max_reproduction_num`**
  - Típusa: `int`
  - Teljesen opcionális
  - Alapértelmezett értéke (ha nincs megadva): `0`
  - Értékkészlete: tetszőleges egész szám, de legalább `0`
  - Leírás: *arra szolgál, hogy megadjuk, hogy a játék során összesen hányszor reprodukálódhasson a* `Player` *ágens.*
- **`reproduction_cost`**
  - Típusa: `double`
  - Teljesen opcionális
  - Alapértelmezett értéke (ha nincs megadva): `20.0`
  - Értékkészlete: tetszőleges pozitív egész szám (nem lehet zérus)
  - Leírás: *arra szolgál, hogy megadjuk, hogy a játék során legalább mennyi hasznot kell összegyűjtenie a* `Player` *ágensnek ahhoz, hogy reprodukálódhasson. A reprodukció nyomán épp ennyivel fog csökkeni az aktuális haszna.*
- **`memlimit`**
  - Típusa: `int`
  - Teljesen opcionális
  - Alapértelmezett értéke (ha nincs megadva): `1000`
  - Értékkészlete: tetszőleges pozitív egész szám (legalább `1`)
  - Leírás: *arra szolgál, hogy megadjuk, hogy a játék során egyszerre legfeljebb hány különböző ellenféllel kapcsolatban tárolhasson emlékeket a* `Player` *ágens. Amennyiben elérjük ezt a limitet, és újabb ellenféllel találkozunk, akivel addig még nem lettünk összepárosítva, úgy a legrégebben frissített ellenfelünkkel kapcsolatos emlékeket töröljük az emlékezetünkből, és helyette bevesszük az új ellenféllel kapcsolatos legfrissebb emlékeket.*
- **`oppmem_limit`**
  - Típusa: `int`
  - Teljesen opcionális
  - Alapértelmezett értéke (ha nincs megadva): `4`
  - Értékkészlete: tetszőleges egész szám, de legalább `1`
  - Leírás: *arra szolgál, hogy megadjuk, hogy a játék során egy ellenféllel kapcsolatban legfeljebb hány emléket (megfigyelést) tárolhasson a* `Player` *ágens az emlékezetében. Amennyiben egy adott ellenfél kapcsán elérjük ezt a*

limitet, úgy elhagyjuk az adott ellenféllel kapcsolatos legrégebbi emlét, és helyette bevesszük a legújabbat.

A paraméterek nevénél megfigyelhettük, hogy a JADE-es ágensekkel ellenben itt most valódi argumentum-neveket soroltunk fel. Arról van szó, hogy amíg a JADE-es ágensek esetében a paraméterek „névtelenek”, és csak a sorrendjük számít, addig JADEX esetében ez pont fordítva van: `név=érték` párok formájában, ámde tetszőleges sorrendben kell megadnunk az ágensek paramétereit.

Eclipse-ben a legkönnyebb elindítani az ágenseket. A JADE-es ágensek indításához hasonlóan elég, ha jobb gombbal rákattintunk a JADEX-es ágensnek megfelelő XML-re, és az `EJADE/Deploy Agent(s)` menüpontot választjuk. Ennek hatására feljön a szokásos ablak, ahol megadhatjuk az ágens lokális nevét, és paramétereinek értékét.

NAGYON FONTOS (!!!): Amennyiben a fentiek közül szeretnénk megadni valamelyik paramétert (mindegyik opcionális), úgy a paraméter-lista elejére oda kell írnunk a JADEX-es ágens konfigurációjának nevét (alapból `default`), és csak utána sorolhatjuk fel a paraméternév-érték párokat. Pl. `default myType=6 max_reproduction_num=3 reproduction_cost=100`

Az Eclipse-es indítás hatására lényegében a `JadeAgentAdapter` ágens hívódik meg, és ez fogja lényegében megvalósítani/interpretálni az általunk indítani kívánt ágenst. Az Eclipse-es indítás hatására tehát nagyjából a következő utasítás kerül végrehajtása.

Példa egy `Player` ágens indítására:

```
java                                jade.Boot                                -container
pa:jadex.adapter.jade.JadeAgentAdapter(milab.lab06.PlayerAgent.Player     "default
myType=6 max_reproduction_num=3 reproduction_cost=100")
```

Ebből tehát jól látszik, hogy az ágenst valójában a `jadex.adapter.jade.JadeAgentAdapter` ágens valósítja majd meg, melynek valójában az első paramétere az ágenst leíró XML fájl elérése (`milab.lab06.PlayerAgent.Player`), majd idézőjelek között, a konfiguráció megnevezését követően, a paraméternév=érték párok felsorolása.

Amennyiben több JADEX-es ágenst szeretnénk indítani, úgy célszerű a fentebbi utasítással dolgozni Eclipse-ben: `Run/Open Run Dialog…/Arguments/…` Miután egymás után felsoroltuk az ágenseket, végül egyetlen gombnyomással (`Run`) elindíthatjuk őket. Például az `Program arguments` részbe írhatjuk a következőt:

```
-container -port 1099 -host localhost
ga:milab.lab06.GameAgent.GameAgent(4 800 4)
g0:jadex.adapter.jade.JadeAgentAdapter(milab.lab06.PlayerAgent.Player     "default
myType=16" "max_reproduction_num=3" "G=4")
g1:jadex.adapter.jade.JadeAgentAdapter(milab.lab06.PlayerAgent.Player     "default
myType=16" "max_reproduction_num=3" "G=4")
g2:jadex.adapter.jade.JadeAgentAdapter(milab.lab06.PlayerAgent.Player     "default
myType=16" "max_reproduction_num=3" "G=4")
g3:jadex.adapter.jade.JadeAgentAdapter(milab.lab06.PlayerAgent.Player     "default
myType=16" "max_reproduction_num=3" "G=4")
g4:jadex.adapter.jade.JadeAgentAdapter(milab.lab06.PlayerAgent.Player     "default
myType=16" "max_reproduction_num=3" "G=4")
g5:jadex.adapter.jade.JadeAgentAdapter(milab.lab06.PlayerAgent.Player     "default
myType=16" "max_reproduction_num=3" "G=4")
g6:jadex.adapter.jade.JadeAgentAdapter(milab.lab06.PlayerAgent.Player     "default
myType=16" "max_reproduction_num=3" "G=4")
g7:jadex.adapter.jade.JadeAgentAdapter(milab.lab06.PlayerAgent.Player     "default
myType=16" "max_reproduction_num=3" "G=4")
```

```
g8:jadex.adapter.jade.JadeAgentAdapter(milab.lab06.PlayerAgent.Player      "default
myType=16" "max_reproduction_num=3" "G=4")
g9:jadex.adapter.jade.JadeAgentAdapter(milab.lab06.PlayerAgent.Player      "default
myType=16" "max_reproduction_num=3" "G=4")
g10:jadex.adapter.jade.JadeAgentAdapter(milab.lab06.PlayerAgent.Player     "default
myType=16" "max_reproduction_num=3" "G=4")
g11:jadex.adapter.jade.JadeAgentAdapter(milab.lab06.PlayerAgent.Player     "default
myType=16" "max_reproduction_num=3" "G=4")
g12:jadex.adapter.jade.JadeAgentAdapter(milab.lab06.PlayerAgent.Player     "default
myType=16" "max_reproduction_num=3" "G=4")
g13:jadex.adapter.jade.JadeAgentAdapter(milab.lab06.PlayerAgent.Player     "default
myType=16" "max_reproduction_num=3" "G=4")
g14:jadex.adapter.jade.JadeAgentAdapter(milab.lab06.PlayerAgent.Player     "default
myType=16" "max_reproduction_num=3" "G=4")
w0:jadex.adapter.jade.JadeAgentAdapter(milab.lab06.PlayerAgent.Player      "default
myType=17" "max_reproduction_num=3" "G=4")
w1:jadex.adapter.jade.JadeAgentAdapter(milab.lab06.PlayerAgent.Player      "default
myType=17" "max_reproduction_num=3" "G=4")
w2:jadex.adapter.jade.JadeAgentAdapter(milab.lab06.PlayerAgent.Player      "default
myType=17" "max_reproduction_num=3" "G=4")
w3:jadex.adapter.jade.JadeAgentAdapter(milab.lab06.PlayerAgent.Player      "default
myType=17" "max_reproduction_num=3" "G=4")
w4:jadex.adapter.jade.JadeAgentAdapter(milab.lab06.PlayerAgent.Player      "default
myType=17" "max_reproduction_num=3" "G=4")
w5:jadex.adapter.jade.JadeAgentAdapter(milab.lab06.PlayerAgent.Player      "default
myType=17" "max_reproduction_num=3" "G=4")
w6:jadex.adapter.jade.JadeAgentAdapter(milab.lab06.PlayerAgent.Player      "default
myType=17" "max_reproduction_num=3" "G=4")
w7:jadex.adapter.jade.JadeAgentAdapter(milab.lab06.PlayerAgent.Player      "default
myType=17" "max_reproduction_num=3" "G=4")
w8:jadex.adapter.jade.JadeAgentAdapter(milab.lab06.PlayerAgent.Player      "default
myType=17" "max_reproduction_num=3" "G=4")
w9:jadex.adapter.jade.JadeAgentAdapter(milab.lab06.PlayerAgent.Player      "default
myType=17" "max_reproduction_num=3" "G=4")
w10:jadex.adapter.jade.JadeAgentAdapter(milab.lab06.PlayerAgent.Player     "default
myType=17" "max_reproduction_num=3" "G=4")
w11:jadex.adapter.jade.JadeAgentAdapter(milab.lab06.PlayerAgent.Player     "default
myType=17" "max_reproduction_num=3" "G=4")
w12:jadex.adapter.jade.JadeAgentAdapter(milab.lab06.PlayerAgent.Player     "default
myType=17" "max_reproduction_num=3" "G=4")
w13:jadex.adapter.jade.JadeAgentAdapter(milab.lab06.PlayerAgent.Player     "default
myType=17" "max_reproduction_num=3" "G=4")
w14:jadex.adapter.jade.JadeAgentAdapter(milab.lab06.PlayerAgent.Player     "default
myType=17" "max_reproduction_num=3" "G=4")
```

Ennek hatására összesen 30 ágens fog létrejönni a 4-es számú, azaz Vezéürü játékot feltételezve. Ebből 15 lesz olyan, aki mindig a `GO` stratégiát játssza, és 15 olyan, amely a `WAIT` stratégiát.

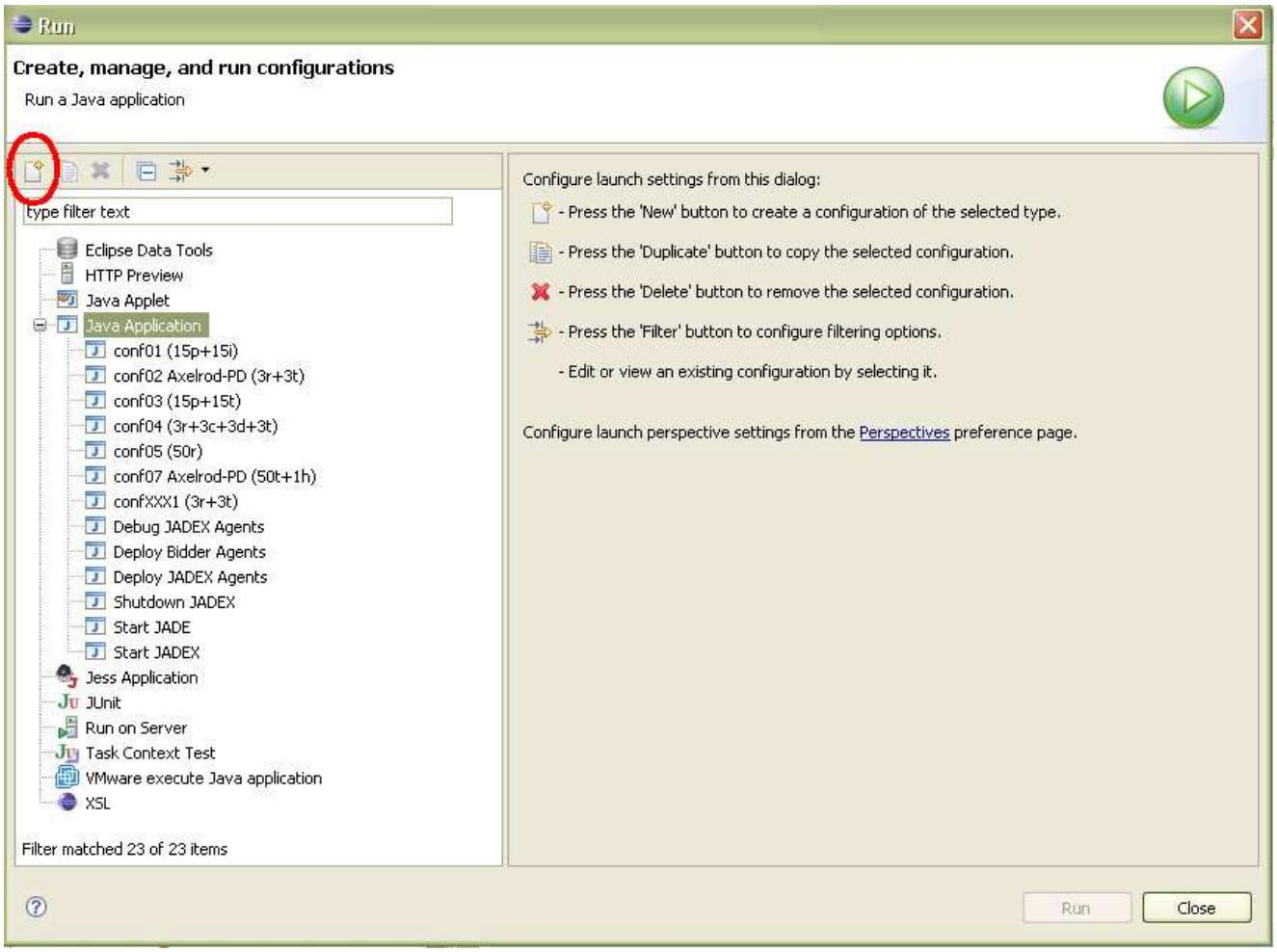
JADE platform beállítása

Ha több ágenst szeretnénk indítani, akkor, mint mondtuk, az csakis az Eclipse-es `Run/Open Run Dialog…/Arguments` dialógus ablak megfelelő beállítása (és esetleg mentése) mellett javasolt. Viszont ebben az ablakban nem csak a `Program arguments` részt kell pl. a fentiekhez hasonlóan kitölteni, hanem a `VM arguments` részt is. Be kell ugyanis állítani, hogy a Java virtuális gép, amely a platformot, vagy a konténert futtatni fogja, mennyi memóriával rendelkezzen. Alapértelmezésben ez túl kevés esetleg több száz ágens futtatásához (feltéve, hogy az ágensek elszaporodnak az evolúció során). Tehát a `VM arguments` részt a biztonság és jó teljesítmény kedvéért állítsuk a következőre:

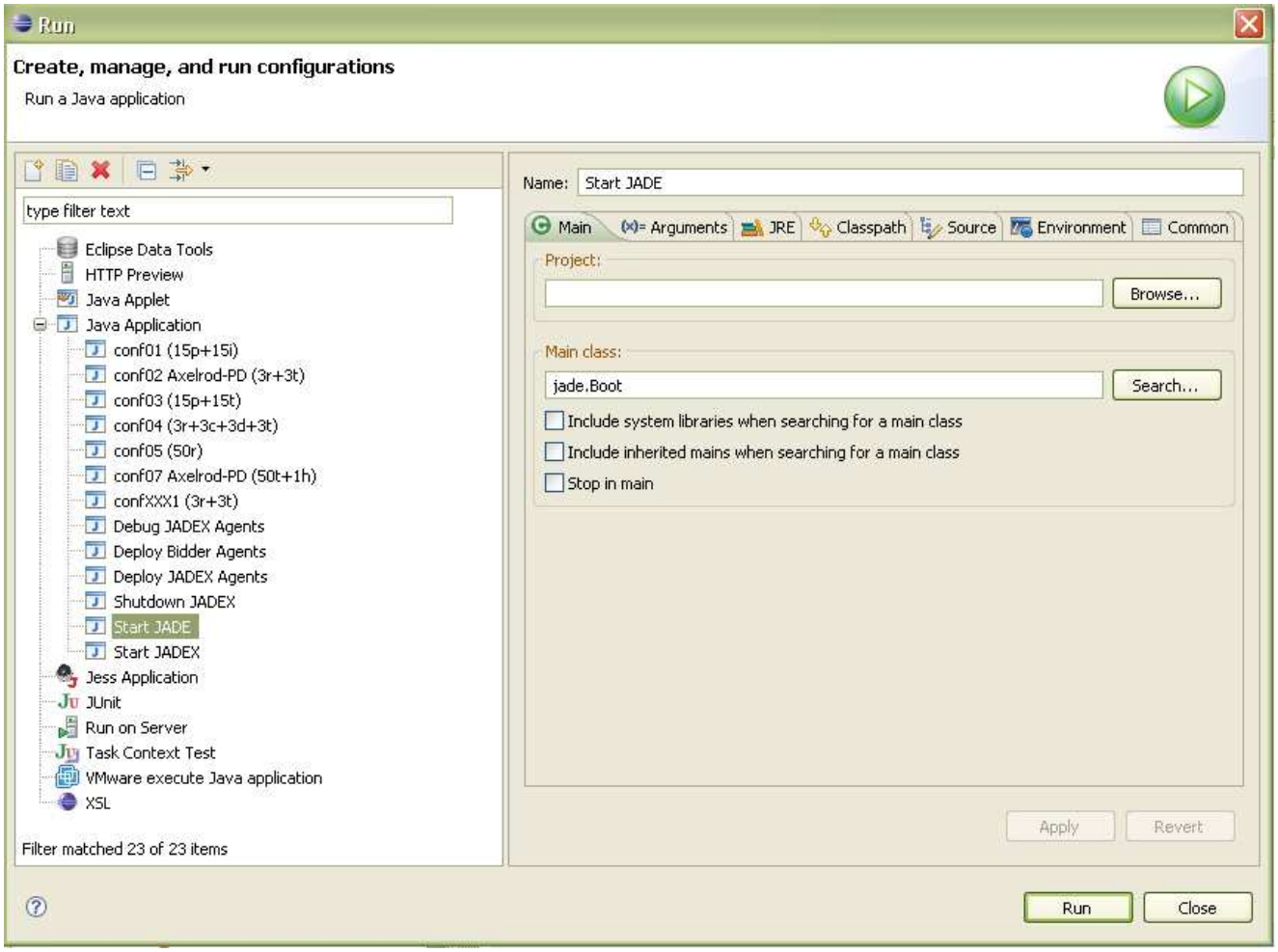
```
-Xms32m -Xmx512m -Dhttp.proxyHost= -Dhttp.proxyPort=
```

Ezen felül, ahhoz, hogy a `DF` ágens képes legyen a `GameAgent` ágensnek több, mint 100 találatot visszaadni akkor, amikor a `GameAgent` ágens `Player` ágenseket keres a platformon, magát a JADE platformot is megfelelően opcionálva kell elindítanunk. A legegyszerűbb, ha a következőket tesszük:

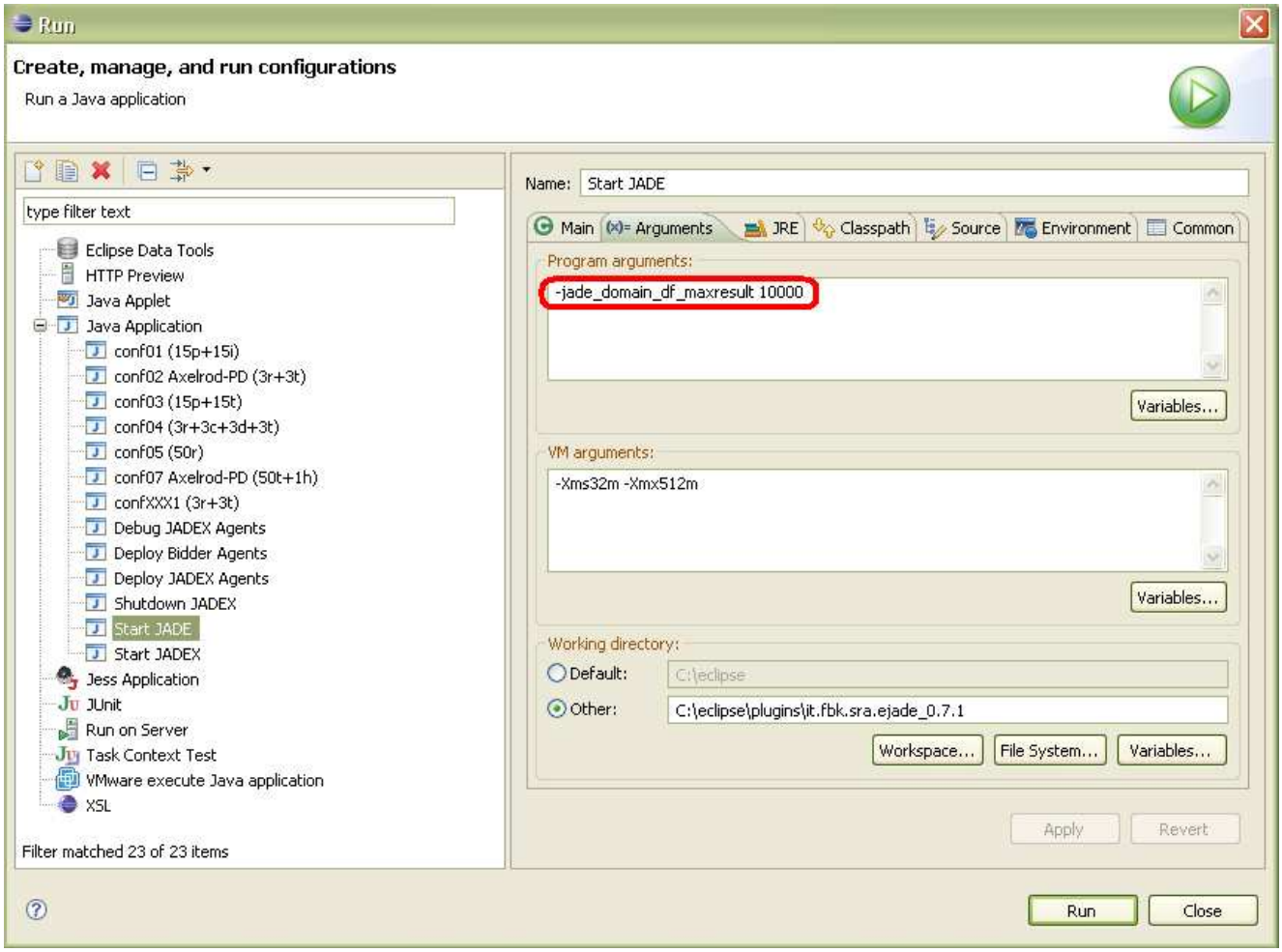
1) Válasszuk ki a `Run/Open Run Dialog…` menüben a `Java Application` elemet, és kattintsunk a `New launch configuration` gombra (lásd. alábbi ábra).



2) Adjuk meg a konfiguráció nevét, pl. `Start JADE`, és `Main` fülben az indítani kívánt főbb osztályt (`Main class = jade.Boot`) az alábbi ábrának megfelelően.



3) Ezek után az `Arguments` fülben adjuk meg a `Program arguments`, `VM arguments`, és `Working directory` részeket az alábbi ábrának megfelelően, majd kattintsunk a `Run` gombra a platform indításához.



A fentebbi ábrán a `Program arguments` részben látható, hogy a platformot úgy indítjuk, hogy a `DF` ágens alapértelmezésben adott 100-as limitjét átírjuk 10000-re (`-jade_domain_df_maxresult 10000`). Nagyszámú ágens esetén ez elengedhetetlen. Továbbá, a hatékonyság kedvéért a platformot GUI nélkül, azaz RMA ágens nélkül indítjuk úgy, hogy a Java virtuális gépnek legalább 32 megabájt, legfeljebb 512 megabájt memóriát engedélyezünk.

Az előbbiekhez hasonló módon különböző játékos ágensek indításához is létrehozhatunk egy-egy `Launch configuration`-t. Ennek haszna nyilván akkor jelentkezik, ha sok ágens van, különböző paraméterekkel, illetve esetleg többször egymás után is szeretnénk kísérletezni egy-egy adott konfigurációval.