Egyetemi doktori (Ph.D.) értekezés tézisei

Korszerű információtechnológiai módszerek bevezetése a mesterséges intelligencia oktatásába

Introduction of Modern Information Technology Methods to the Teaching of Artificial Intelligence

Kósa Márk Szabolcs

Témavezető: Dr. Várterész Magda

Debreceni Egyetem
Természettudományi Doktori Tanács
Matematika- és Számítástudományok Doktori Iskola

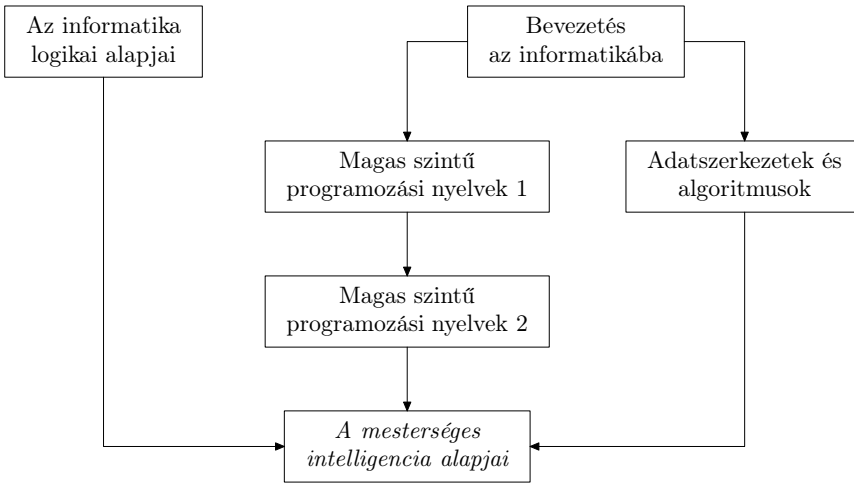
Debrecen, 2009

Tartalomjegyzék

1. FEJEZET

Bevezetés és célkitűzések

Az elmúlt évtizedben az objektumorientált szemléletmód széles körű elterjedését figyelhettük meg az informatikai világban. E technika elterjedésének szükségszerűen meg kellett jelennie az informatika oktatásában is. A dolgozatomban bemutattam azt, hogy milyen lehetőségeket rejt az objektumorientált programozási szemlélet térnyerése az informatikus egyetemi hallgatók oktatásban és tehetséggondozásában. Vizsgálódásaim terepét a Debreceni Egyetemen a BSc szintű programtervező informatikus képzés egyik kötelező tárgyának, *A mesterséges intelligencia alapjai* című kurzusnak a gyakorlati foglalkozásai jelentették.



1.1. ábra. *A mesterséges intelligencia alapjai* című tárgy alapozó tárgyai

A Debreceni Egyetemen a BSc szintű programtervező informatikus kép-

zésben *A mesterséges intelligencia alapjai* című kurzus négy olyan tantárgy ismereteire épül, amelyekkel hallgatóink tanulmányaik korábbi féléveiben már találkozhattak. Ezek kapcsolatát szemlélteti az 1.1. ábra.

A tárgyat a képzési struktúra negyedik félévében vehetik fel először a hallgatók. Ekkor már – szerencsés esetben – túl vannak a két féléves programozás képzésen, ahol megismerkedhettek a C programozási nyelvvel és egy objektumorientált nyelv (Java vagy C#) alapjaival, valamint túl vannak az informatika logikai alapjait tárgyaló kurzuson is. Mindemellett hallgathattak előadást a különböző adatszerkezetekről és a hozzájuk kapcsolódó algoritmusokról is. Így aztán fontosnak tartom kiemelni, hogy a kurzust látogató hallgatók nem ezen a kurzuson kerülnek először kapcsolatba az objektumorientált világgal, már szerezhettek tapasztalatokat valamilyen objektumorientált programozási nyelvvel kapcsolatosan.

A tárgy előadásain ismerkedhetnek meg a hallgatók az MI klasszikus kutatási területeivel, alapvető módszereivel és eszközeivel, legfontosabb eredményeivel. Az alapvető módszerek közül a gráfokban megoldást kereső eljárásokat tanulják részletesen ebben a félévben.

A gráfban megoldást kereső eljárások tanításához először reprezentáljuk a problémát állapottéren vagy problémaredukcióval. Mindkét esetben megadjuk a megfelelő gráfreprezentációt. A megoldást kereső rendszerek felépítésének elemzése után rátérünk a konkrét eljárások tanítására. Az algoritmusainkat különböző szempontok alapján csoportosíthatjuk (nem módosítható – módosítható; nem informált – heurisztikus stratégiák). Elsősorban a módosítható kereső eljárásokkal (backtrack algoritmus; keresőgráffal történő keresések: szélességi, mélységi, optimális gráfkereső; heurisztikus gráfkeresők: best-first eljárás, A-algoritmusok) foglalkozunk. Vizsgáljuk ezen eljárások algoritmikus tulajdonságait is (teljesség, megfelelőség, bonyolultság).

A problémaredukciós feladatmegoldásnak megfelelő ÉS/VAGY gráfokban való megoldáskereséseket mint az eddigi algoritmusok általánosításait tárgyaljuk. Megadjuk a kétszemélyes, zérusösszegű, teljes információjú játékok nyerő stratégiájának fogalmát, és módszert nyújtunk legjobbnak tűnő következő lépés kiválasztására (minimax, alfabéta algoritmusok).

A mesterséges intelligencia tárgy gyakorlati kurzusának anyagát két nagy témakör: a gráfokban való keresések, valamint a kétszemélyes játékok algoritmusainak témaköre köré lehet csoportosítani. Mindkét témakör kiválóan alkalmas objektumorientált módon történő megközelítésre, ugyanakkor számtalan olyan optimalizálási lehetőséget is nyújt, amelyeket az

objektumorientált megközelítés általánosításai elrejtenek a felületes szemlélő elől. Ez az utóbbi tény inspirált arra, hogy bizonyos problématípusok (problémaosztályok) esetén ne csak a magas szintű absztrakciót nyújtó objektumorientált lehetőségeket vizsgáljam meg, hanem más módszerekkel is megoldásokat adjak a kitűzött feladatokra.

Az egyes problémaosztályok optimalizálási lehetőségeinek feltérképezése azért is fontos, mert a tárgy keretében elsajátított algoritmusok az informatika más területein is visszaköszönnek a programozókra, hol látványosan (web- és mobilalkalmazások), hol kevésbé látványosan (adatbázisalkalmazások). Napjainkban ugyan az erőforrások jelentősen megsokszorozódtak, de megfontolt felhasználásukat szintén ezeken a kurzusokon sajátíthatják el a hallgatók.

Mindezen tényekből kiindulva dolgozatom megírásakor célkitűzéseim a következők voltak:

- javaslat arra, hogy hogyan közelítsük meg a mesterséges intelligencia alapvető algoritmusait az informatikában ma elterjedt objektumorientált szemlélettel;
- a Debreceni Egyetem Informatikai Karán, a mesterséges intelligencia tárgy bevezető kurzusának gyakorlatain a hallgatóknak bemutatott és tőlük számonkért állapottér-reprezentációs technikának az integrálása az objektumorientált modellbe;
- az objektumorientált megközelítésből származó előnyök és hátrányok felmérése;
- a „hagyományos” technikák (elemi kereső és rendező algoritmusok, mohó algoritmusok, dinamikus programozás) alkalmazhatóságának vizsgálata speciális problématípusok esetén;
- az ACM nemzetközi programozó versenyek néhány feladatának elemzése és felhasználása egyrészt a gyakorlati kurzusokon történő oktatás, másrészt a programozói versenyeken induló hallgatók felkészítésének eredményesebbé tétele érdekében;
- olyan ellenőrző szoftver kifejlesztése és használata, amellyel a bemutatott algoritmusok hatékonysága mérhetővé és összehasonlíthatóvá válik.

2. FEJEZET

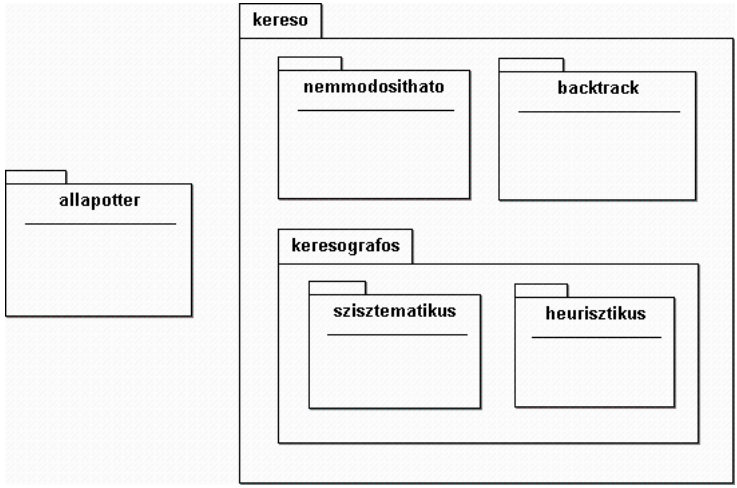
Új eredmények

2.1. Állapottér-reprezentáció és keresőalgoritmusok objektumorientált megközelítésben

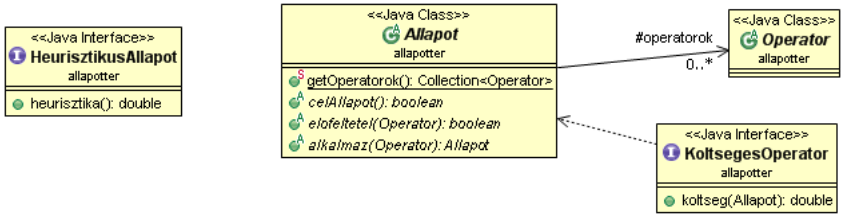
Az állapottérgráfokban történő megoldáskereséshez az elmúlt időszakban kifejlesztettünk két olyan Java csomagot, amelyek ügyesen felparaméterezve lehetőséget biztosítanak tetszőleges probléma bármely kívánt algoritmussal történő megoldására. *A mesterséges intellgiencia alapjai* tárgy gyakorlati kurzusainak követelményeként a hallgatóknak olyan programokat kell készíteniük, amelyek felhasználhatják ezen csomagokat. Az általunk javasolt két Java csomag egyike az állapottér-reprezentációval, a másik pedig a megoldáskereső algoritmusokkal kapcsolatos osztályokat tartalmazza (2.1. ábra).

Az állapottér-reprezentáció csomagjában két osztályt található (2.2. ábra). Az `Operator` osztály az egyes problémákhoz tartozó operátorok, míg az `Allapot` osztály a problémákhoz tartozó állapotok absztrakt ősosztálya. Mivel egy állapot jellemzői mindig a konkrét problémától függnek, ezért ez utóbbi osztályban csak egyetlen adattagot definiáltunk, amely nem más, mint a probléma állapotaira alkalmazható összes operátor halmaza. Természetesen – függetlenül a problémától – minden állapotnak meg kell tudnia mondani magáról, hogy célállapot-e, hogy alkalmazható-e rá egy adott operátor, illetve hogy egy operátor alkalmazásával milyen új állapot jön létre belőle.

A heurisztikus keresők olyan állapotokat használnak, amelyek a fentebb felsorolt jellemzőkön kívül még egy heurisztikaértéket is tartalmaznak. Ehhez nyújt segítséget a `HeurisztikusAllapot` interfész azáltal, hogy az ezt megvalósító állapotok rendelkezni fognak egy heurisztikus értéket szolgáltató módszerrel. Ehhez hasonlóan a `KoltsegesOperator` interfészt azok az operátorok fogják implementálni, amelyeket az operátorok költségét figyelembe vevő keresőalgoritmusok használnak.



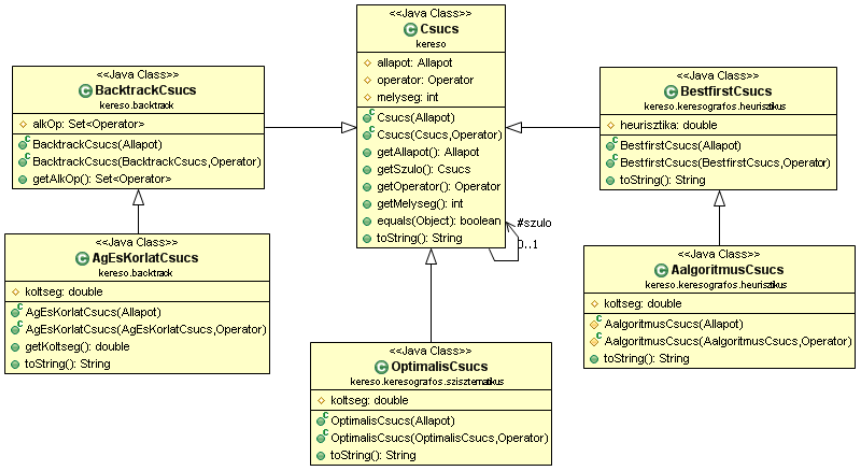
2.1. ábra. A csomagstruktúra



2.2. ábra. Az `allapotter` csomag osztályai és interfészei

A megoldáskereső algoritmusok csomagjában lévő osztályok két különálló hierarchiába rendeződnek. Az egyik (2.3. ábra) az állapottérgráfok csúcsainak, a másik (2.4. ábra) a különböző keresőalgoritmusoknak a modellezésére szolgál.

A csúcsokra vonatkozó csaknem minden információ a különböző csúcstípusok közös ősosztályában, a `Csucs` osztályban található. Itt tároljuk a csúcs által szemléltetett állapotot, a csúcsnak a startcsúcstól mért távolságát (mélységét), a szülő csúcsát, illetve azt az operátort, amellyel a szülőjében tárolt állapotból az adott csúcsban tárolt állapotot előállítottuk.

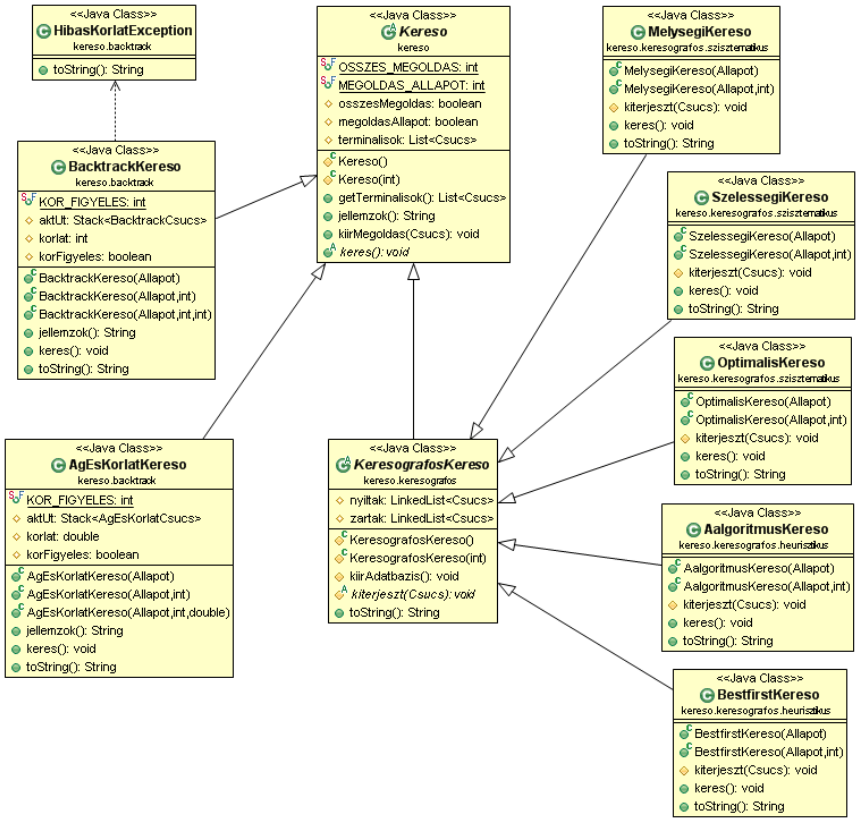


2.3. ábra. A reprezentációs gráf csúcsainak osztályai

A `Csucs` osztály leszármazottai ezeken az attribútumokon túl a konkrét keresőalgoritmusok által igényelt további adattagokat tartalmaznak.

A keresőalgoritmusok ősosztálya az absztrakt `Kereso` osztály, amely éppen a legfontosabb metódusát nem implementálja, nevezetesen azt, hogy milyen stratégiával kell működnie a megoldáskeresés vezérlőjének. Ennek a metódusnak a kódját a leszármazott osztályok tartalmazzák majd. Ugyanakkor lehetőséget biztosít arra, hogy tároljuk a terminális csúcsokat, kiírjuk a megoldásokat, illetve hogy beállítsuk az egyes keresők algoritmustól független jellemzőit.

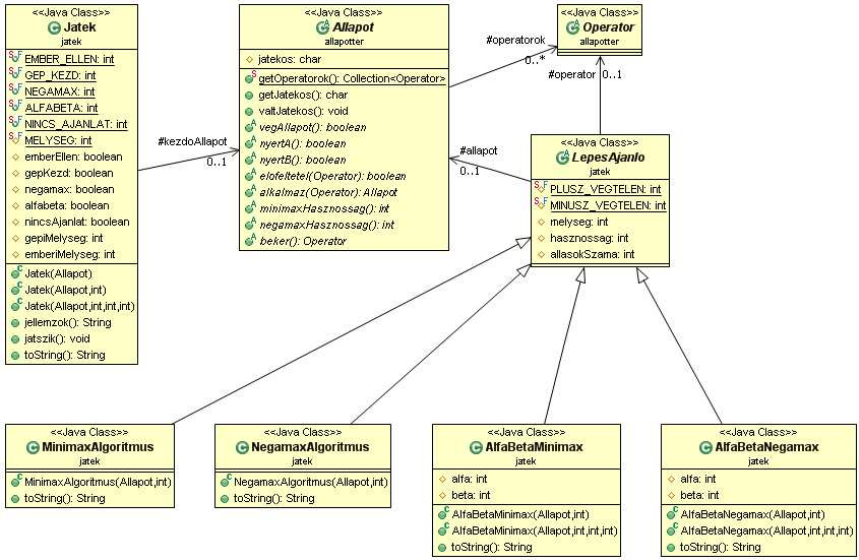
A `BacktrackKereso` osztály új jellemzői között találjuk az aktuális utat, valamint a körfigyelés és az úthosszkorlát opcionális lehetőségét. Az `AgEsKorlatKereso` osztály annyiban tér el ettől, hogy az úthosszkorlát helyett költségkorláttal dolgozik, amely itt nem opció, hanem az algoritmus szerves részét képezi. A `KeresografosKereso` osztály a kereső adatbázisában tárolt csúcsok nyilvántartásához két listát használ. Ezen osztály leszármazottai (a konkrét gráfkereső algoritmusok) egyrészt a keresési stratégiában, másrészt pedig az adatbázisukban tárolt csúcsok osztályában térnek el egymástól.



2.4. ábra. A keresőalgoritmusok osztálydiagramja

2.2. Kétszemélyes, zérusösszegű, teljes információjú játékok objektumorientált megközelítésben

A mesterséges intelligencia alapjai tárgy gyakorlati kurzusain a kétszemélyes, determinisztikus, véges, zérusösszegű és teljes információjú játékokkal is foglalkozunk. Kifejlesztettünk két Java csomagot, amelyek segítségünkre vannak e játékok implementálásánál. Ezek a csomagok az állapottér-reprezentációval és a lépésajánló algoritmusokkal kapcsolatos osztályokat tartalmazzák (2.5. ábra).



2.5. ábra. A lépésajánló algoritmusok osztálydiagramja

Az `allapotter` csomagban lévő `Operator` osztály az egyes játékokhoz tartozó operátorok, míg az `Allapot` osztály a játékokhoz tartozó állapotok absztrakt ősosztálya. Mivel egy állapot jellemzői mindig a konkrét játéktól függnek, ezért ez utóbbi osztályban mindössze két adattagot definiáltunk. Az egyik a játék állapotaira alkalmazható összes operátor halmaza, a másik pedig az aktuális állapotban lépni következő játékos. A konkrét játéktól függetlenül minden állapot el tudja dönteni, hogy végállapot-e, és ha igen, akkor milyen eredménnyel ért véget a játék (melyik játékos nyert, ha nem döntetlennel végződött a játszma). Szintén minden állapotnál vizsgálható, hogy alkalmazható-e rá egy adott operátor, illetve megadható, hogy egy operátor alkalmazásával milyen új állapot jön létre belőle. A `beker` absztrakt metódus implementálásával valósíthatjuk meg az emberi játékos lépéseinek a beolvasását.

A játékokhoz kötődő algoritmusokat megvalósító osztályokat a `jatek` csomagban helyeztük el. A lépésajánló algoritmusok ősosztálya az absztrakt `LepesAjanlo` osztály, amelynek az attribútumai tárolják azt az *állapotot*, amelyhez a legjobb lépést szeretnénk meghatározni, a lépéskeresés

mélységét, a keresés végrehajtását követően javasolt *operátort*, az ezzel az operátorral az adott mélységig előretekintve elérhető legjobb állapot *hasznosságát*, illetve a keresés során kiértékelt állapotok *darabszámát*. Ezeket a jellemzőket minden leszármazott osztályban kiszámítják a konkrét lépésajánló algoritmusok, melyeket a kiértékelt állapotok számának a segítségével tudunk majd összehasonlítani egymással.

A konkrét lépésajánló algoritmusok a legjobb lépés keresését rekurzívan, újabb és újabb lépésajánló algoritmusok példányosításával végzik el. Emiatt a keresések megvalósításához nem volt szükségünk más metódusok használatára a konstruktorokon kívül. Az alfabéta vágást végző algoritmusok osztályait kiegészítettük két olyan adattaggal, amelyek a játékfa egyes csomópontjaihoz tartozó aktuális *alfa* és *béta* értékeket tárolják.

A játék vezérlését a `Jatek` osztályban definiált `jatszik` metódus valósítja meg. Ez a metódus egy ciklust futtat mindaddig, míg az aktuális állapot példány végállapotba nem kerül. Mindeközben lépésenként megjeleníti az aktuális állást, és eldönti, hogy ki következik lépni. Ha interaktívan játszunk, és az emberi játékos lép, akkor beolvassa a játékos lépését, egyébként példányosít egy lépésajánlót, és annak segítségével meghatározza, hogy a gép mit lépjen. A lépés ismeretében mindkét esetben alkalmazza a lépésnek megfelelő operátort, és frissíti az aktuális állapotot.

A játékok és a lépésajánló algoritmusok testreszabását a játék példányosításakor végezhetjük el. A játék jellemzőjeként állíthatjuk be azt, hogy ember ember ellen, vagy ember gép ellen játsszon-e. Utóbbi esetben azt is megadhatjuk, hogy ki kezdje a játékot. Eldönthetjük, hogy a lépésajánlást minimax vagy negamax algoritmussal végezze-e a program, illetve hogy használjanak-e alfabéta vágást az algoritmusok. Az emberi játékos számára is kérhetünk lépésajánlatot, amelyet a gépi lépésajánlathoz hasonlóan egy lépésajánló példányosításával határoz meg a program. Ezt az ajánlatot aztán az emberi játékos vagy elfogadja, vagy elutasítja. Ráadásul külön szabályozhatjuk azt, hogy milyen mélységben tekintsen előre a program a gép és az ember lépésajánlatának a kiszámításakor.

A lépésajánlatok jóságát (erősségét) két tényező befolyásolja. Az egyik az előretekintés mélysége, a másik pedig az egyes állapotokra alkalmazott kiértékelő függvény megbízhatósága. Látni kell, hogy a játékfa elágazási tényezőjétől függően az előretekintés mélységének a növelésével a lépésajánlat ideje exponenciálisan nőhet. Érdemes tehát minél jobb kiértékelő függvényt írni a konkrét játék osztályában. Amennyiben olyan kiértékelő függvényt tudunk írni, amely a játék minden állapotáról biztosan el tudja

dönteni, hogy az mennyire jó az egyes játékosok számára, akkor az előretekintés mélységét nem szükséges 1-nél nagyobb értékre állítani. Az éremnek persze két oldala van: egy-egy állapot kiértékelése is hosszú időt vehet igénybe, minimalizálni tehát valójában a kiértékelt állapotok számának és az egyes állapotok kiértékelésére fordított időnek a szorzatát kell.

2.3. PCRM: kiértékelő szoftver programozói versenyekhez

A hallgatók felé átadott ismeretek számonkérésének sokféle módja létezik. Egy lehetőség, amivel az elmúlt években többször is éltünk, a különféle versenyek szervezése. A mesterséges intelligencia kurzusokon elsajátított ismereteiket hallgatóink egyrészt a tárgy keretein belül meghirdetett háziversenyeken, másrészt pedig az Informatikai Kar[1] által szervezett programozói versenyeken tehetik próbára. Utóbbiak esetében ACM stílusú programozó versenyekről van szó, amelyeken az 5 óra programozási idő alatt megoldandó 7–10 feladat között sokszor szokott előfordulni olyan feladat, amelynek a megoldásához felhasználható a mesterséges intelligencia témaköreihez köthető algoritmus. Mindkét versenyfajtánál kulcsfontosságú a feladatok megoldásainak pontos ellenőrzése.

Egy programozói verseny folyamán a versenyzőknek (akik csapatok is lehetnek) feladatokat kell megoldaniuk. A versenyzők az egyes feladatok megoldását valamilyen programozási nyelven megírt forráskód formájában készítik el. A lefordított forráskódból előállított futtatható program a bemeneti adatokból egy kimeneti adathalmazt hoz létre.

A célunk egy olyan szoftver fejlesztése volt, amely nagy mennyiségű megoldást képes feldolgozni mind versenykörülmények között, mind pedig off-line módon.

A Programming Contest Result Manager (PCRM) egy olyan program, amely segíti a programozói versenyek zsűrizését valós időben, de a megoldások off-line kiértékelésére is alkalmas. A versenyzőknek a feladatokat számítógépen kell megoldaniuk, majd a megoldásokat el kell küldeniük az on-line zsűrinek. A beérkező forráskódokat a program lefordítja, a futtatható kódot végrehajtja, majd ellenőrzi a kimenetet, amely vagy egy állományban, vagy a szabványos kimeneten áll elő.

Amikor a zsűri megkapja egy feladat megoldását egy versenyzőtől, lefordítja azt a megfelelő fordítóprogrammal, és futtatja különböző tesztese-

[1] 2004 előtt a Természettudományi Kar (Matematikai és) Informatikai Intézete.

tekre. Minden feladathoz tartozik legalább egy olyan állomány, amely a teszteseteket tartalmazza. Ezeket az állományokat kell a beküldött programoknak feldolgozni, és a helyes kimenetet előállítani. A kimenet akkor helyes, ha vagy megegyezik egy előre generált, tárolt állománnyal, vagy ha egy külső ellenőrző program szerint helyes.

A PCRM program tehát úgy segíti a zsűrizést, hogy a beküldött megoldásokat kiértékeli, majd naplózza és megjeleníti az eredményeket. A beérkező megoldásokat a PCRMPOP3 program fogadja e-mailek formájában. Ez a program biztosítja a zsűrizés teljes automatizálását azzal, hogy meghatározott időközönként feldolgozza az e-maileket, és a forráskódokat átadja kiértékelésre a PCRM programnak.

A program három különböző módban futtatható: *feladat alapú*, *pontszám alapú* és *ACM módban*.

Feladat alapú módban az egyetlen lényeges értékelési szempont a helyesen megoldott feladatok száma. A feladat csak akkor tekinthető megoldottnak, ha a beküldött program minden tesztesetre helyes eredményt ad.

Pontszám alapú módban a versenyzők sorrendjét azoknak az állományoknak a darabszáma határozza meg, amelyekre a beküldött program helyes eredményt ad (függetlenül az állományban lévő tesztesetek számától és nehézségi fokától).

A PCRM programot eredetileg ACM-versenyek zsűrizésére fejlesztettük ki. *ACM módban* az a versenyző nyer, aki a legtöbb helyes megoldást küldte be. Ha két vagy több versenyző azonos számú feladatot oldott meg helyesen, akkor az nyer, akinek kisebb a pontszáma. A pontszámot a következőképpen kell kiszámítani: Minden helyes megoldás esetén a versenyző annyi pontot kap, ahány perc telt el a verseny kezdetétől a helyes megoldás beküldéséig. Ehhez a pontszámhoz hozzá kell adni annyiszor egy előre meghatározott büntetőpontot, ahány hibás megoldást küldött be az adott (helyesen megoldott) feladatra.

A PCRM program segítségével 2002-től bonyolítottuk le az Informatikai Kar programozó versenyeit. A tesztelések során bebizonyosodott, hogy a program kiválóan alkalmazható néhány tantárgy (*Magas szintű programozási nyelvek*, *A mesterséges intelligencia alapjai*) házi feladatainak ellenőrzésére is.

Tudományos közlemények

Az értekezés témájához kapcsolódó referált közlemények

1. **Kósa Márk**, Pánovics János, Gunda Lénárd: An evaluating tool for programming contests, *Teaching Mathematics and Computer Science* (2005) **3** (1), p. 103–119.

   Ref. szám: Mathematics Didactics Database ME 2005f.02643. (http://www.zentralblatt-math.org/matheduc/)

2. **Kósa Márk**, Nagy Benedek: Logical puzzles (truth-tellers and liars), *Proceedings of the 5th International Conference on Applied Informatics* (2001) p. 105–112.

   Ref. szám: Zentralblatt MATH Zbl 1103.68826. (http://www.zentralblatt-math.org/zmath/)

Az értekezés témájához kapcsolódó tankönyv

1. Juhász István, **Kósa Márk**, Pánovics János: *C példatár*, Panem, Budapest, 2005.

Konferenciakiadványban megjelent dolgozatok

1. Várterész Magda, Nagy Benedek, **Kósa Márk**, Pánovics János: *A Mesterséges intelligencia tárgy bevezető kurzusának gyakorlatai a Debreceni Egyetemen*, Informatika a Felsőoktatásban 2002 Konferencia, Debrecen, 1103–1109. oldal.

2. **Kósa Márk**: *A kiszolgálási elvek hatása a Markov-vezérelt véges forrású sorbanállási rendszerek teljesítmény-mérőszámaira*, Informatika a Felsőoktatásban 2002 Konferencia, Debrecen, 1146–1153. oldal.

3. **Kósa Márk**, Pánovics János, Gunda Lénárd: *An evaluation tool for programming contests*, 6th International Conference on Applied Informatics, Eger, Vol. I., p. 163–172.

4. **Kósa Márk**: *Stochastic Simulation of Markov-Modulated Finite-Source Queues in Java Environment*, 6th International Conference on Applied Informatics, Eger, Vol. II., p. 369–377.

5. **Kósa Márk**, Nagy Benedek, Pánovics János: *Megoldáskereső algoritmusok hatékonyságának vizsgálata az állapottér-reprezentációk függvényében*, Számítástechnika az Oktatásban 2006, XVI. nemzetközi konferencia, Szováta, Románia, 76–81. oldal.

6. **Kósa Márk**, Pánovics János: *Keresőalgoritmusok objektumorientált megközelítése a* Mesterséges intelligencia *tárgy bevezető kurzusán*, Számítástechnika az Oktatásban 2007, XVII. nemzetközi konferencia, Nagyvárad, Románia, 94–97. oldal.

7. **Kósa Márk**, Pánovics János: *Szoftverfejlesztés a Qt keretrendszer használatával*, Számítástechnika az Oktatásban 2008, XVIII. nemzetközi konferencia, Csíksomlyó, Románia, 179–184. oldal.

Az értekezés témájához kapcsolódó konferencia előadások

1. Nagy Benedek, **Kósa Márk**: *Igazmondó-hazug fejtörők gráfelméleti megközelítésben*, XXV. Magyar Operációkutatási konferencia, Debrecen, 2001. október 17–20.

2. Várterész Magda, Nagy Benedek, **Kósa Márk**, Pánovics János: *A Mesterséges intelligencia tárgy bevezető kurzusának gyakorlatai a Debreceni Egyetemen*, Informatika a Felsőoktatásban 2002 Konferencia, Debrecen, 2002. augusztus 28–30.

3. **Kósa Márk**, Pánovics János, Gunda Lénárd: *An evaluation tool for programming contests*, 6th International Conference on Applied Informatics, Eger, 27–31 January 2004.

4. **Kósa Márk**, Nagy Benedek, Pánovics János: *Megoldáskereső algoritmusok hatékonyságának vizsgálata az állapottér-reprezentációk függvényében*, Számítástechnika az oktatásban 2006, XVI. nemzetközi konferencia, Szováta, Románia, 2006. május 25–28.

5. **Kósa Márk**, Nagy Benedek, Pánovics János: *Performance analysis of search algorithms depending on the state space representation*, 6th Joint Conference on Mathematics and Computer Science, Pécs, 12–15 July 2006.

6. **Kósa Márk**, Pánovics János: *Search algorithms at ACM contests*, 7th International Conference on Applied Informatics, Eger, 28–31 January 2007.

További konferencia előadások

1. Almási Béla, **Kósa Márk**, Sztrik János: *Nemmegbízható terminálrendszerek optimalizálási problémái a MOSEL segítségével*, XXV. Magyar Operációkutatási Konferencia, Debrecen, 2001. október 17–20.

2. **Kósa Márk**: *Markov-vezérelt véges forrású sorbanállási rendszerek szimulációja*, XXV. Magyar Operációkutatási konferencia, Debrecen, 2001. október 17–20.

3. Sztrik János, Oliver Möller, **Kósa Márk**: *The effects of service disciplines on the performance measures of Markov modulated finite-source queuing systems*, XII Seminar on Stability Problems of Stochastic Models, Varna, Bulgaria, 25–31 May 2002.

4. **Kósa Márk**: *A kiszolgálási elvek hatása a Markov-vezérelt véges forrású sorbanállási rendszerek teljesítmény-mérőszámaira*, Informatika a Felsőoktatásban 2002 Konferencia, Debrecen, 2002. augusztus 28–30.

5. **Kósa Márk**: *Stochastic Simulation of Markov-Modulated Finite-Source Queues in Java Environment*, 6th International Conference on Applied Informatics, Eger, 27–31 January 2004.

További tudományos közlemények

1. **Kósa Márk**, Pánovics János: *Programming Contest Result Manager*, Technical reports, 2009/9, Preprints No. 369, Faculty of Informatics, University of Debrecen.

Szoftvertermék

1. Programming Contest Result Manager, 2003.

3. CHAPTER

Introduction and Motivation

In the world of informatics, object-oriented approach has become very popular in the last decade. This method apparently must have come up also in the field of education. In my thesis, I presented the potential of teaching object-oriented aspect of programming at universities and in talent-care programs. During my researches, I was focusing on the practical courses of the subject titled *Introduction to Artificial Intelligence*. This subject is one of the compulsory subjects of the Software Engineering BSc level curriculum at the University of Debrecen.

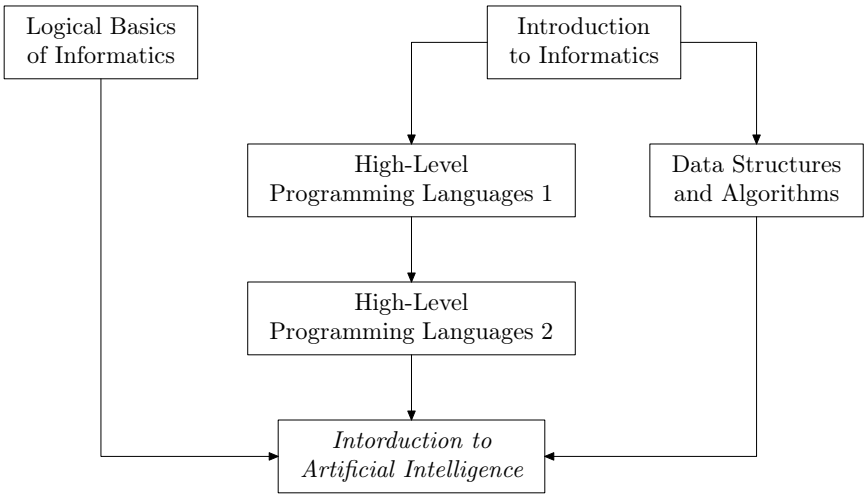


Figure 3.1. Prerequisite subjects of *Introduction to Artificial Intelligence*

The *Introduction to Artificial Intelligence* of the Software Engineering BSc level curriculum at the University of Debrecen is based on the knowl-

edge of four subjects which the students may have met during the preceding semesters. The relations of these subjects are shown in Figure 3.1.

The students may register to this subject in the fourth semester of the curriculum. Before this, they had fulfilled the two-semester programming course where they got acquainted with the C programming language and the basics of an object-oriented language (Java or C#), and they got over the course *Logical Basics of Informatics*. Apart from this, they could take part in lectures about data structures and algorithms. I would like to emphasize that students attending this course do not get in connection with the object-oriented world for the first time in this course, they already have experience with an object-oriented programming language.

The students learn about the classic research areas, basic methods and the most important results of AI in the lectures of this subject. Among the basic methods, they learn about the graph search algorithms in detail in the frame of this course.

For teaching graph search algorithms, first we have to represent the problem using state-space or problem-reduction representation. We give the appropriate graph representation in both cases. After analysing the structures of the search systems, we present the concrete procedures. We can classify the algorithms based on various aspects (non-modifiable—modifiable; non-informed—heuristic control strategies). We mainly deal with modifiable search algorithms (backtracking, graph search algorithms: breadth-first search, depth-first search, uniform cost search; heuristic graph search algorithms: best-first search, A-algorithms). We also examine the algorithmic properties (completeness, soundness, complexity) of these procedures.

We talk about search algorithms using AND/OR graphs as generalizations of the previously mentioned algorithms used for problem-reduction representation. We give the concept of winning strategy of two-player zero-sum games of perfect information, and introduce methods to determine the probably best next move (minimax method with and without alpha-beta pruning).

The practical course of the subject *Introduction of Artificial Intelligence* covers two main topics: graph search algorithms and algorithms for two-player games. Both topics are very suitable for object-oriented approach and give numerous possibilities of optimization which are hidden by the generalizations of the object-oriented approach. This fact inspired me in case of certain problem classes to examine not only the object-oriented

approaches which give us high-level abstraction but to find solution to the given problems using other methods, too.

Discovering the possibilities of optimization of the different problem classes is important also because the programmers can meet the algorithms learned in the frame of the subject at other areas of informatics (like web, mobile and database applications). Although the resources have escalated very rapidly nowadays, students can also learn to use them in a careful manner in these courses.

There are many ways to test the students' skills. For this purpose, we have organised programming contests a number of times in the previous years. The students can put their knowledge learned in the Artificial Intelligence courses to the test at contests in the frame of the subject on the one hand, and at contests organised by the Faculty of Informatics[1] on the other hand. In the case of the latter, these are ACM style programming contests. Among the 7–10 problems to be solved during the five-hour programming time, the problem set often contains problems that can be solved using algorithms related to the topics of artificial intelligence.

Based on these facts, my aims were the following:

- to present the basic algorithms of artificial intelligence in the popular object-oriented approach;
- to integrate the state-space representation technique presented to the students into the object-oriented model;
- to discover the pros and cons of the object-oriented approach;
- to examine the applicability of "traditional" methods (the use of basic data structures, greedy algorithms, dynamic programming) in the case of special problem types;
- to analyse some problems of ACM international programming contests and to use them during the practical courses and for preparing students for programming contests;
- to develop an application which is capable of measuring and comparing the efficiency of the presented algorithms.

[1] Formerly Institute of (Mathematics and) Informatics of the Faculty of Natural Sciences.

4. CHAPTER

New Results

4.1 State-Space Representation and Search Algorithms Using Object-Oriented Approach

We have developed two Java packages for graph search algorithms which provide us with the possibility to solve any problem by using any search algorithm. As one of the requirements of the practical course of *Introduction to Artificial Intelligence*, the students have to write a program which may utilize these packages. One of the two Java packages contains classes regarding the state-space representation, the other of them contains classes regarding search algorithms.

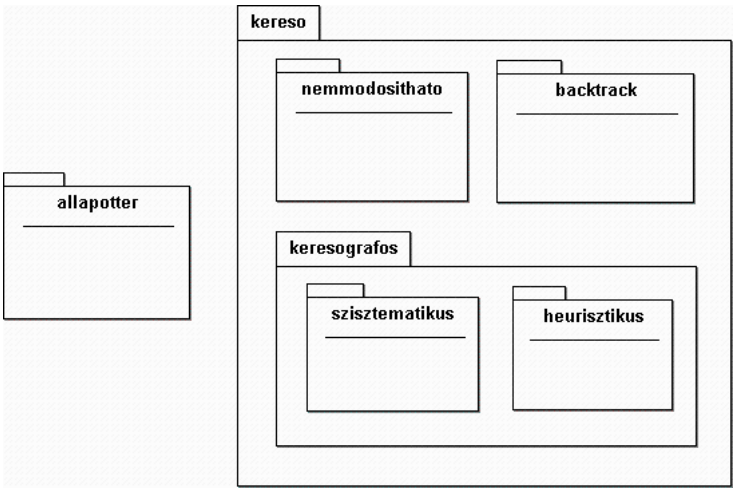


Figure 4.1. The package structure

In the package of the state-space representation two classes can be found (see Figure 4.2). The `Operator` class is the abstract superclass of the operators given in the various problems, while the `Allapot` class is the abstract superclass of the states of the problems. As the attributes of a state always depend on the actual problem, we define only one field in the latter class which is the collection of operators applicable to the states of the problem. Of course, independently of the problem, each state has to know if it is a goal state or not, if an operator is applicable to it or not, and the state which derives from it by applying an operator.

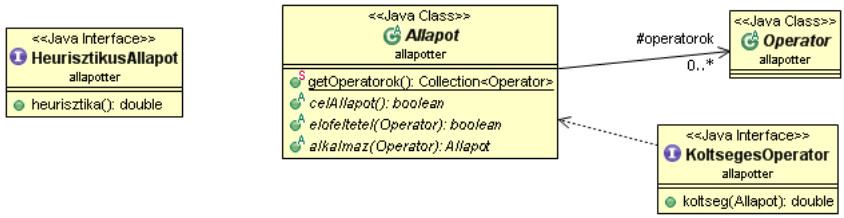


Figure 4.2. Classes and interfaces in the `allapotter` package

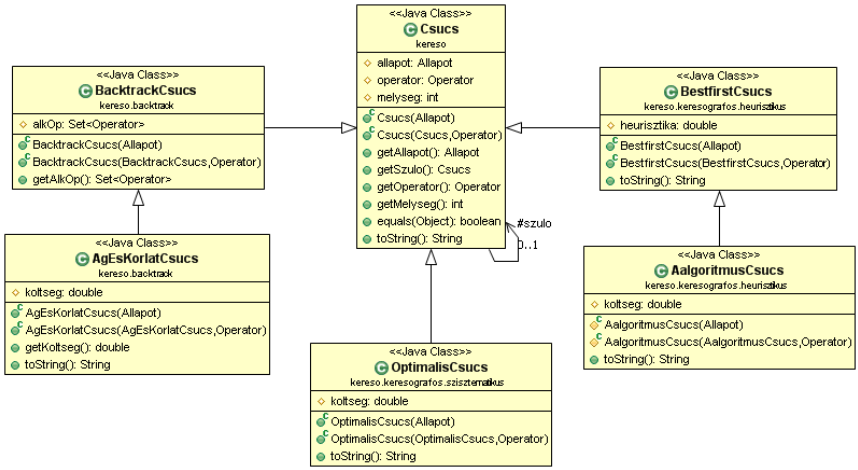


Figure 4.3. Class diagram of the representation graph nodes

The heuristic search algorithms require states that have a further heuristic value in addition to the above mentioned attributes. The `HeuriszticusAllapot` interface helps us realize this because states implementing this interface will possess a method which provides a heuristic value. Similarly, the `KoltsegesOperator` interface will be implemented by operators having cost. These operators are utilized, for example, by the uniform-cost search algorithm or the A algorithm.

Classes in the package of the search algorithms are organised in two different hierarchies. One of these hierarchies serves for modeling the nodes of the representation graphs (see Figure 4.3), while the other is modeling the different search algorithms (see Figure 4.4).

Almost all of the information concerning the nodes can be found in the common superclass of the various node types, the `Csucs` class. This class stores the state represented by the node, the distance between the start node and the current node (the depth of the node) as well as the operator with which the state stored in the current node has been generated from the parent state. The derivatives of the `Csucs` class contain, besides these members, further fields required by the different search algorithms.

The superclass of the search algorithms is the abstract `Kereso` class which does not implement its most important method, namely the one that controls the strategy of the search algorithm. This method will be implemented by the derived classes. At the same time, this class provides us with a means for storing the terminal nodes, printing the solutions, and setting the algorithm-independent properties of the search algorithms.

Among the new attributes of the `BacktrackKereso` class, we can mention the current path and options for cycle checking and path length bound. The `AgEsKorlatKereso` class differs from this only in that it uses cost bound instead of path length bound, and that this is no more an option but an integral part of the algorithm. The `KeresografosKereso` class uses two lists to register the nodes stored in the database. The derivatives of this class (the concrete graph-search algorithms) differ from one another in the control strategy and the classes of nodes stored in their databases.

4.2 Two-Player, Zero-Sum Games of Perfect Information Using Object-Oriented Approach

In the practical courses of *Introduction to Artificial Intelligence* we also talk about two-player, deterministic, finite, zero-sum games of perfect in-

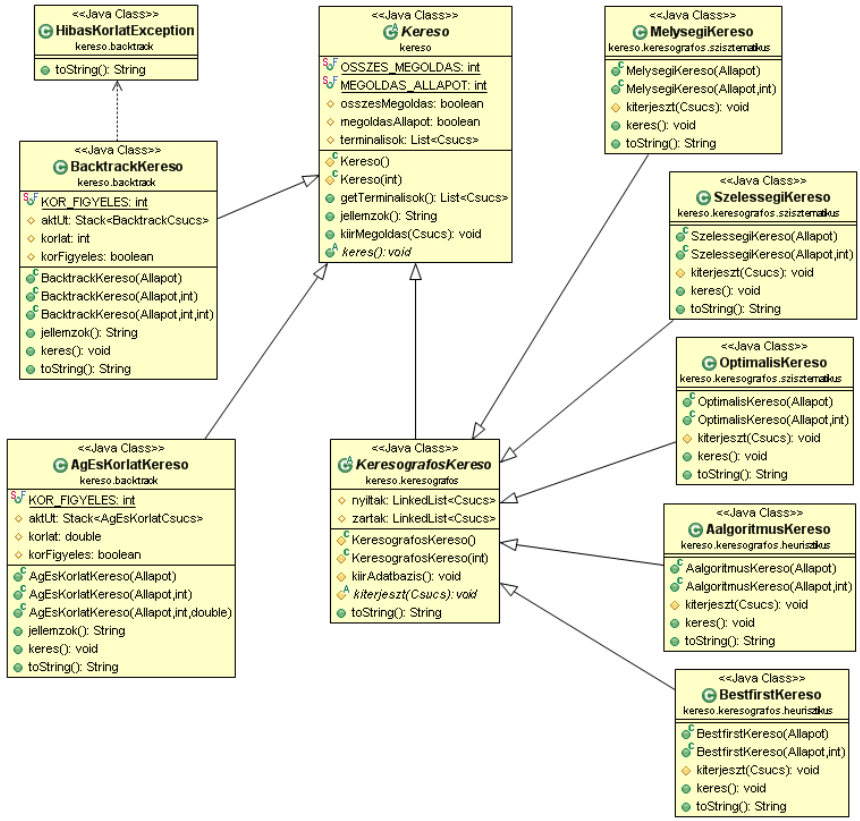


Figure 4.4. Class diagram of the search algorithms

formation. We have developed two Java packages which may be used to implement such a game. These packages contain classes regarding the state-space representation and algorithms for choosing the next move in a game (see Figure 4.5).

In the package of the state-space representation two classes can be found. The `Operator` class is the abstract superclass of the operators given in the games, while the `Allapot` class is the abstract superclass of the states of the games. As the attributes of a state always depend on the actual game, we define only two fields in the latter class. One of these fields

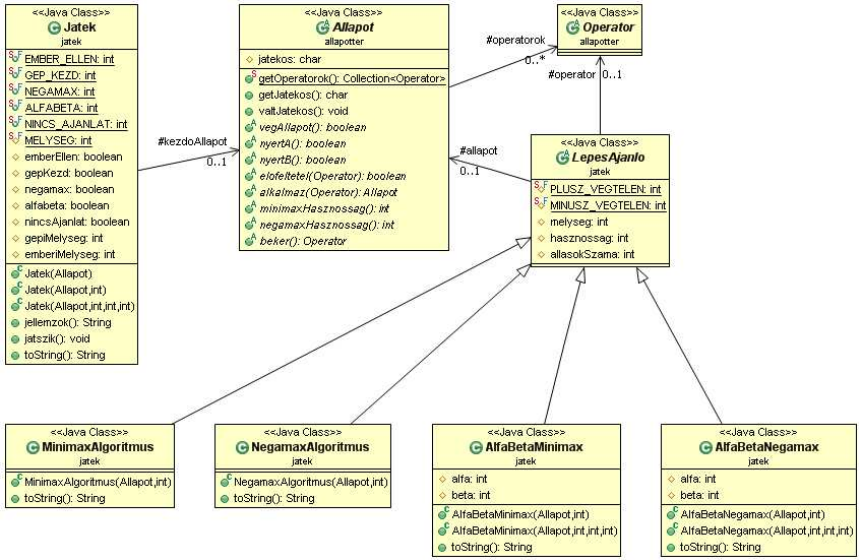


Figure 4.5. Class diagram of the algorithms for choosing the next move

is the collection of the operators applicable to the states of the game, and the other one represents the player who is in turn in the current state of the game. Independently of the game, each state has to know if it is a final state, and if so what the result of the game is (which player wins if the game is not a tie). I can also examine if an operator is applicable to a state or not, and we can give the state which derives from the state by applying the operator. We can realize the human players' input by implementing the abstract `beker` method.

Classes implementing the algorithms related to games are placed in the `jatek` package. The superclass of the algorithms choosing the next move is the abstract `LepesAjanlo` class. The fields of this class store the *state*, the *depth* of the search, the suggested *operator*, the *utility function's value* of the state reached by the suggested operator and the *number of states* evaluated during the search. These attributes will be computed by the concrete algorithms implemented in the derived classes. These algorithms can be compared to one another with the help of the number of evaluated states.

The concrete algorithms choosing the next move work recursively by instantiating objects of the contrete algorithm classes. Because of this, there was no need for other methods other than the constructors. Classes of algorithms using alpha-beta pruning have been extended by two fields which represent the *alpha* and *beta* values that belong to each node of the game tree.

The `jatszik` method defined in the `Jatek` class is responsible for controling the game. The body of this method contains a loop which runs until the current state instance is a final state. While in the loop, it displays the current state and determines which player is in turn. If we play interactively and the human player is in turn then it will read the player's move, otherwise it instantiates an algorithm object and determines the computer's move using this object. Afterwards it applies the operator representing this move in both cases and updates the current state.

We can customize the games and the algorithms choosing the next move at the time of the game's instantiation. We can choose whether a human player plays against another human player or a computer player. In the latter case, we can give the player who will make the first move. We can set either minimax or negamax algorithm as the algorithm choosing the next move, and we can decide whether or not these algorithms should use alpha-beta pruning. We may ask for a suggestion for the human player. Similarly to the case of the computer's move, this suggestion will be computed by instatiating an algorithm object. The human player may either accept or reject this suggestion. Additionally, we can give different depths of search for computing the computer's move and the human's suggestion.

The strength of the suggestion is affected by two factors. One of these factors is the depth of the search and the other is the efficiency of the utility function applied to each state. Depending of the branching factor of the game tree, the time required to compute the next move may increase exponentially by increasing the depth of the search. That is why it is worth to write as efficient utility function as we can in the class of the actual game. If we can write a utility function that can determine for every state of the game how much that state is good for each of the players then there is no point in setting the depth of the search to a value greater than one. However, the coin has two sides: it may take a very long time to evaluate a state so what we really have to minimize is the product of the number of the evaluated states and the time spent for evaluating each state.

4.3 PCRM: An Evaluation Tool for Programming Contests

Precise checking of the solutions of the problems is of key importance for contests organised both in the frame of the subject *Introduction to Artificial Intelligence* and by the Faculty of Informatics. We developed a software called Programming Contest Result Manager (PCRM) to automatically evaluate the solutions. PCRM is a program that can be used to help judge a programming contest where the contestants must solve problems on a computer and then send the solutions to the online jury. It can also be used to judge an offline competition as well. The PCRM program can automatically check the sent solutions, i.e. compile them, run them, and check their output. It can work with programs that produce an output file and also with programs that read from the standard input and write to the standard output.

When the jury receives the source file from a contestant for a certain problem, it compiles it with the appropriate compiler and runs the program with test cases. For each problem, there is one or maybe more (but at least one) test case file, which the program must process, and generate a correct output. An output is correct, if it is the same as a pre-generated one, or if a program that verifies outputs, finds that the output is correct.

When using PCRM, there are test case files for each problem. However, each physical file can of course contain more test cases – this is the case in ACM mode, where all test cases must actually be in one file. This usually means that the input is built so that it can contain several test cases following one another.

PCRM helps you in automating the judging process by receiving the solutions (automatically), compiling them, running them, evaluating the result, and keeping a track of events and generating (or showing) the complete result.

PCRM also includes a POP3 client program that can accept incoming solutions from the contestants via email and can fully automate the entire judging process.

The program can work in one of three modes: *problem based mode*, *score based mode* and *ACM mode*.

In *problem based mode*, the only measure is the number of problems solved. The individual test cases do not count, i.e. a problem is only solved if all test cases are solved correctly. You can have as many problems as you

wish and each problem can have as many test cases as needed.

In *score based mode*, the winner is determined by the number of test case files for which the output of the program is correct. Every test case file that is evaluated is equal to a score of 1 (or more, if a solution checking program is used. In this case, the solution checking program determines the score for a test case file). That is, if we have 5 problems and 3 test case files for each, the maximum score is 15 and the minimum score is 0. You can have as many problems as you wish and each problem can have as many test cases as needed.

PCRM was originally developed for ACM-style competitions. In *ACM mode*, the contestant with the highest number of correctly solved problems wins. If there are two or more contestants with the same number of solutions, a score is calculated, and the contestant with the least score wins. The score is calculated as follows: For each problem, the contestant gets as many points as the number of minutes passed since the beginning of the competition. For every unsuccessful submission (compile error, runtime error, wrong answer, etc.) they also get penalty points. But penalty is only given if the problem is accepted in the end. So, if I sent in a solution to problem A after 80 minutes, got a wrong answer, sent it in after 84 minutes again (I am fast at finding the error), then again at 92 minutes, which is finally accepted, I will get $92 + 2 \times$ penalty points (assuming penalty is 20, that sums up to 132). If someone else sent in a correct solution after me with no prior incorrect submissions (say at 118 minutes in the competition), that contestant would still beat me. In ACM mode, there can only be one test case file for each problem. This does not mean one test case because these problems usually have a test case format which allows several test cases to be put into a single file. Even if you specify more test cases, PCRM will only work with one.

We used this software from 2002 during contests organised by the Faculty of Informatics and also for testing the homework of some subjects (*High-Level Programming Languages*, *Introduction to Artificial Intelligence*).

List of Publications

Reviewed papers in dissertation topics

1. **Márk Kósa**, János Pánovics, Lénárd Gunda: *An evaluating tool for programming contests*, Teaching Mathematics and Computer Science (2005) **3** (1), p. 103–119.

   Ref. number: Mathematics Didactics Database ME 2005f.02643. (http://www.zentralblatt-math.org/matheduc/)

2. **Márk Kósa**, Benedek Nagy: *Logical puzzles (truth-tellers and liars)*, Kovács Emőd (ed.) et al., Proceedings of the 5th International Conference on Applied Informatics, Eger, Hungary, January 28–February 3, 2001, p. 105–112 (2001).

   Ref. number: Zentralblatt MATH Zbl 1103.68826. (http://www.zentralblatt-math.org/zmath/)

Book in dissertation topics

1. István Juhász, **Márk Kósa**, János Pánovics: *C példatár* (in Hungarian), Panem, Budapest, 2005.

Conference Proceedings

1. Magda Várterész, Benedek Nagy, **Márk Kósa**, János Pánovics: *A Mesterséges intelligencia tárgy bevezető kurzusának gyakorlatai a Debreceni Egyetemen*, Informatika a Felsőoktatásban 2002 Konferencia, Debrecen, p. 1103–1109.

2. **Márk Kósa**: *A kiszolgálási elvek hatása a Markov-vezérelt véges forrású sorbanállási rendszerek teljesítmény-mérőszámaira*, Informatika

a Felsőoktatásban 2002 Konferencia, Debrecen, p. 1146–1153.

3. **Márk Kósa**, János Pánovics, Gunda Lénárd: *An evaluation tool for programming contests*, 6th International Conference on Applied Informatics, Eger, Vol. I., p. 163–172.

4. **Márk Kósa**: *Stochastic Simulation of Markov-Modulated Finite-Source Queues in Java Environment*, 6th International Conference on Applied Informatics, Eger, Vol. II., p. 369–377.

5. **Márk Kósa**, Benedek Nagy, János Pánovics: *Megoldáskereső algoritmusok hatékonyságának vizsgálata az állapottér-reprezentációk függvényében*, 16th International Conference on Computer Science, Sovata, Romania, 2006, p. 76–81.

6. **Márk Kósa**, János Pánovics: *Keresőalgoritmusok objektumorientált megközelítése a* Mesterséges intelligencia *tárgy bevezető kurzusán*, 17th International Conference on Computer Science, Oradea, Romania, 2007, p. 94–97.

7. **Márk Kósa**, János Pánovics: *Software Development Using the QT Framework* (in Hungarian), 18th International Conference on Computer Science, Sumuleu-Ciuc, Romania, 2008, p. 179–184.

Conference talks in dissertation topics

1. Benedek Nagy, **Márk Kósa**: *Igazmondó-hazug fejtörők gráfelméleti megközelítésben*, XXV. Magyar Operációkutatási konferencia, Debrecen, October 17–20, 2001.

2. Magda Várterész, Benedek Nagy, **Márk Kósa**, János Pánovics: *A Mesterséges intelligencia tárgy bevezető kurzusának gyakorlatai a Debreceni Egyetemen*, Informatika a Felsőoktatásban 2002 Konferencia, Debrecen, August 28–30, 2002.

3. **Márk Kósa**, János Pánovics, Lénárd Gunda: *An evaluation tool for programming contests*, 6th International Conference on Applied Informatics, Eger, January 27–31, 2004.

4. **Márk Kósa**, Benedek Nagy, János Pánovics: *Megoldáskereső algoritmusok hatékonyságának vizsgálata az állapottér-reprezentációk függvényében*, 16th International Conference on Computer Science, Sovata, Romania, May 25–28, 2006.

5. **Márk Kósa**, Benedek Nagy, János Pánovics: *Performance analysis of search algorithms depending on the state space representation*, 6th Joint Conference on Mathematics and Computer Science, Pécs,July 12–15, 2006.

6. **Márk Kósa**, János Pánovics: *Search algorithms at ACM contests*, 7th International Conference on Applied Informatics, Eger, 28–31 January, 2007.

Further conference talks

1. Béla Almási, **Márk Kósa**, János Sztrik: *Nemmegbízható terminálrendszerek optimalizálási problémái a MOSEL segítségével*, XXV. Magyar Operációkutatási Konferencia, Debrecen, October 17–20, 2001.

2. **Márk Kósa**: *Markov-vezérelt véges forrású sorbanállási rendszerek szimulációja*, XXV. Magyar Operációkutatási konferencia, Debrecen, October 17–20, 2001.

3. János Sztrik, Oliver Möller, **Márk Kósa**: *The effects of service disciplines on the performance measures of Markov modulated finite-source queuing systems*, XII Seminar on Stability Problems of Stochastic Models, Varna, Bulgaria, 25–31 May, 2002.

4. **Márk Kósa**: *A kiszolgálási elvek hatása a Markov-vezérelt véges forrású sorbanállási rendszerek teljesítmény-mérőszámaira*, Informatika a Felsőoktatásban 2002 Konferencia, Debrecen, August 28–30, 2002.

5. **Márk Kósa**: *Stochastic Simulation of Markov-Modulated Finite-Source Queues in Java Environment*, 6th International Conference on Applied Informatics, Eger, January 27–31, 2004.

Further

1. **Márk Kósa**, János Pánovics: *Programming Contest Result Manager*, Technical reports, 2009/9, Preprints No. 369, Faculty of Informatics, University of Debrecen.

Software products

1. Programming Contest Result Manager, 2003.