Jazyk pre špecifikáciu pravidiel stolných hier ako základ automatického inteligentného hracieho robotického systému

Pavel Petrovič

Comenius University in Bratislava
Mlynská dolina, 842 48 Bratislava
ppetrovic@acm.org

Abstrakt

Stolové hry sú ideálnou platformou pre štúdium rôznych aspektov umelej inteligencie. Predstavujeme systém pozostávajúci z robotického ramena, univerzálneho softvérového frameworku a jazyka na popis pravidiel stolových hier. Systém bol vyvinutý pre projekt na Letnej škole vedy v Požege v lete 2013. Predmetom záujmu je najmä deklaratívny jazyk s procedurálnymi prvkami. Pravidlá zapísané v tomto jazyku sú stručné, dobre čitateľné a efektívne interpretovateľné. Framework slúži na výskumné a výukové projekty zamerané na algoritmy pre hranie hier a špecifikačné deklaratívne jazyky.

1 Úvod

Jedným z centrálnych problémov v umelej inteligencii je adekvátna reprezentácia poznatkov. Neexistuje jediný vhodný, správny a najefektívnejší spôsob reprezentácie. Naopak, jej výber je podmienený konkrétnou množinou úloh, ktorú má daný inteligentný systém plniť a z nej vyplývajúcej sady požiadaviek na reprezentáciu: Bude potrebné vykonávať akcie v spojitom priestore? Bude potrebné prijímať logické závery? Bude potrebné, aby systém vedel okamžite reagovať na podnety z prostredia (reaktivita)? Bude potrebné pracovať s neurčitou informáciou? Bude treba riešiť nemonotónne usudzovanie? Bude potrebné modifikovať reprezentáciu učením sa na základe skúsenosti? Ak áno, v akom režime – s učiteľom, bez učiteľa, odmenou a trestom? Priamym zadaním inštrukcií? Bude potrebné reprezentovať procedurálne znalosti? Má ich systém byť schopný sám modifikovať? Bude treba riešiť ťažké problémy/úlohy? Má byť proces usudzovania a prijímania rozhodnutí dobre vysvetliteľný, zdôvodniteľný a človeku zrozumiteľný?

V snahe konštruovať umelé inteligentné pre človeka užitočné systémy, ktorou sa odlišujeme od práce Všemohúceho a od zvrátených snáh o zostrojenie človeka s dvojnásobne veľkou hlavou a mozgovou kapacitou, si volíme také reprezentácie, ktoré sú v danej čiastkovej alebo celkovej úlohe najprimeranejšie. Preto nadmieru nevyzdvihujeme reprezentácie podobné inteligentným biologickým systémom, hoci sa nimi v prípade zjavnej užitočnosti bez zdráhania inšpirujeme a nebiologická plausibilita nám väčšinou pripadá vhodnejšia ako plausibilita biologická, keďže naše umelé systémy majú s biológiou a v prírode prebiehajúcimi metabolickými, reprodukčnými, evolučnými, enzymatickými a inými procesmi (aby sme pomenovali aspoň niektoré z mnohých) a ich magnitúdou spoločné asi toľko, ako majú príbehy o včielke Maji spoločné s výrobou plošných spojov do medzihviezdnych vesmírnych sond. Tým samozrejme nijako neznevažujeme vedeckú prácu kolegov biológov, lekárov a niektorých psychológov, ktorí skúmajú procesy v človeku a prípadne v ostatných živých tvoroch, ale predstavu o zostrojení niečoho mäkkého, príjemného, mysliaceho, chodiaceho po dvoch nohách ako my a komunikujúceho s nami rovnocenným spôsobom v tomto tisícročí, alebo prinajmenšom v tomto storočí najmä z pragmatických dôvodov nezdieľame. Nachádzame sa v ére drsných, tvrdých silikónových tvorov, ktoré nás porážajú svojou výpočtovou silou, presnosťou a spoľahlivosťou senzorov, množstvom naraz spracovávanej informácie, obrovskou fyzickou silou a rýchlosťou, pevnosťou, pružnosťou a odolnosťou materiálov. Sú našimi bezpodmienečnými otrokmi, ku ktorým sa správame chladnokrvne a neprisudzujeme im žiadne slobody a práva. Tieto tvory si preto vyžadujú celkom iné postupy a prístupy k návrhu, konštrukcii i riadiacej logike ako doposiaľ objavila príroda. Aj preto aj v dnešnej dobe pre pokrok prináša viac matematika ako konekcionizmus, napríklad.

V tejto práci sa venujeme návrhu inteligentného systému, ktorý hrá stolové hry proti ľudskému hráčovi – v prípade hier pre dvoch hráčov, alebo sám – hry pre jedného hráča. Hlavným cieľom je vytvorenie systému, ktorý je univerzálny, čiže nie je predurčený na jednu konkrétnu hru alebo skupinu hier. Nové hry je možné systém naučiť hrať špecifikáciou pravidiel hry, ktorá sa zapisuje v našom vlastnom deklaratívnom jazyku s procedurálnymi prvkami a ktorý je jadrom tejto práce. Deklaratívne jazyky sú veľmi vhodné pre systémy umelej inteligencie, kde znalosti vkladá (alebo aspoň programuje) človek vďaka dobrej prehľadnosti a symbolickej povahe, ktorá je človeku zrozumiteľná a je

tiež formálne analyzovateľná – na rozdiel od subsymbolických distribuovaných reprezentácií. Čiastočne obmedzujúce sa nám zdá byť ich časté jednostranné zameranie na niektorý inferenčný mechanizmus alebo výpočtový model: napríklad jazyk Prolog, pracujúci s Hornovými klauzulami využíva SLD rezolvenciu, alebo jazyk Scheme teoreticky vybudovaný na jedinom operátore lambda bez akýchkoľvek bočných efektov. Na jednej strane sú vhodnými prostriedkami pre formálne metódy analýzy programov, na druhej strane tieto zaväzujúce predpoklady môžu viesť k nepraktickým scenárom použitia a implementáciám. Myslíme si, že reprezentačný jazyk by mal dobre umožňovať nielen deklarovanie bázy poznatkov, ale poskytovať vyjadrovacie prostriedky pre ich procedurálne realizácie a to flexiblne – nielen jedným vyhraneným štýlom interpretácie programu. Na druhej strane, miera všeobecnosti každého nástroja je ohraničená cieleným záberom a každá aplikácia má svoje hranice. Vytvorený jazyk je teda akýmsi hybridným diskusným príspevkom do tejto témy medzi mantinelmi všeobecnej a špecifickej vyjadrovacej sily, deklaratívneho a procedurálneho štýlu zápisu, formálnej čistoty a ad-hoc riešeniami s bočnými efektami. Je kompromisom vyplývajúcim zo série pragmatických rozhodnutí, ale z nášho hľadiska plní svoj účel dobre.

V nasledujúcich statiach príspevku vysvetlíme špecifikáciu problému, ktorý sme riešili, opíšeme fyzický hardvér systému, softvérový framework pre experimenty so systémom, ktorý sme navrhli a implementovali v jazyku Java, definujeme jazyk na reprezentáciu pravidiel hier, uvedieme a analyzujeme príklady niekoľkých hier, prediskutujeme prepojenie jazyka s frameworkom, zmienime sa o alternatívnom prístupe, ktorý je predmetom aktuálne riešenej diplomovej práce a myšlienky na záver zhrnieme.

2 Problém

Pôvodnou motiváciou pre túto prácu bola ústna komunikácia s kolegom Ľuborom Illekom, ktorý našej skupine vyčítavo pripomenul – prečo už dávno nemáme na „Matfyze“ robota, proti ktorému by bolo možné hrať dámu, človeče, prípadne šach a iné stolové hry? Túto myšlienku sme si osvojili a keďže sme získali jednoduché robotické rameno, zadali sme diplomovú prácu na túto tému. Okrem toho sme boli neskôr pozvaní ako projektoví vedúci na vyše týždňové intenzívne medzinárodné sústredenie pre stredoškolákov, Summer School of Science (S3), Požega, Chorvátsko 2013. Keďže rýchlosť postupu na diplomovej práci bola „štandardná“, rozhodli sme sa nezávisle od diplomanta radšej pripraviť vlastný framework, na ktorom by študenti na letnej škole mohli realizovať sadu experimentov o využití umelej inteligencie na hranie hier. Vychádzali sme z nasledujúcich predpokladov:

- systém bude hrať hry pre jedného alebo dvoch hráčov, ktorí sa v ťahoch striedajú
- hra pozostáva z postupnosti ťahov
- hra má cieľ, ktorému spravidla zodpovedá nejaká konfigurácia hry alebo jej parametre
- jeden ťah pozostáva z jedného alebo viacerých preložení „kameňov“ - figúrok, ktoré sa smú klásť len na definované miesta
- ťahy by mali byť realizovateľné robotickým ramenom v štýle akcií pick & place
- situáciu hry a najmä jednotlivé kroky ťahov dokáže rozpoznať kamera
- hru je možné hrať aj virtuálne – na simulovanej hracej ploche
- na každom mieste môže byť naraz iba jedna figúrka – ostatné prípady sa riešia zmenou stavu figúrky, prípadne duplicitnými miestami
- každý „kameň“ aj miesto, kam sa kamene ukladajú, majú svoj jednoznačný identifikátor a typ

Pred príchodom na letnú školu sme museli vyriešiť nasledujúce úlohy:

- navrhnúť vhodný reprezentačný formalizmus pre popis pravidiel hry
- navrhnúť a implementovať parser pre tento formalizmus, ktorého výstupom je vnútorná reprezentácia stavu hry a pravidiel
- zabezpečiť kontrolu správnosti ťahov, generovanie možných ťahov v danej hernej konfigurácii a testovanie konca hry
- implementovať softvérové riadenie robota a spracovanie obrazovej informácie z kamery
- implementovať virtuálnu hraciu plochu a umožniť hráčovi hrať v grafickom simulátore
- pripraviť modulárny framework pre testovanie rozličných algoritmov umelej inteligencie, ktorých výstup by riadil robotické rameno a vstup by bol získavaný analýzou obrazu zachytávajúceho aktuálnu hernú situáciu, alebo ktoré môžu riadiť hráča v simulovanej hre – vrátane vzájomných turnajov jednotlivých algoritmov, ktoré by mali byť spracované automaticky a vygenerovať logy vhodné na ďalšie štatistické spracovanie
- implementovať prvé príklady algoritmov umelej inteligencie do nášho frameworku ako východisko pre projekt na S3
- vytvoriť niekoľko ukážkových špecifikácií hier.

Všetky uvedené úlohy sme pred začiatkom S3 splnili a na S3 sme s tromi študentami vo veku okolo 16 rokov realizovali úspešný a pre nás v mnohom inšpiratívny projekt. Výstupom ich práce bol článok prezentovaný na konferencii ELMAR, Cvijović, Unger, Corazza (2013). V ďalšom opíšeme stručnú

motiváciu k stolovým hrám a komponenty vytvoreného systému.

3 Stolové hry

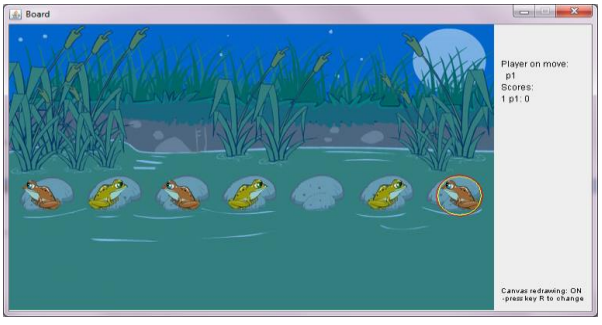
Stolové hry sú tradičná oblasť výskumu (i výuky) v oblasti umelej inteligencie. Na ich úspešné hranie často nestačia bežné deterministické algoritmy. Pri ich hraní sa možno zdokonaľovať rôznymi spôsobmi učenia sa, možno využívať usudzovanie a uvažovanie, sú ideálnou platformou pre skúmanie tvorby analógií, induktívneho i deduktívneho uvažovania, reprezentácie poznatkov, plánovania a modelovania, generalizácie, či učenia sa z príkladov, Petrovič (1997). Niekto by dokonca mohol tvrdiť, že umelá inteligencia na úrovni základného výskumu iné problémy ako hry ani nepotrebuje, všetko ostatné sú buď aplikácie alebo niečo iné. Ľudia sa hrali stolové hry ešte keď stôl ani nevedeli vyrobiť. Stolové hry sa tešili popularite krížom skrz celú históriu ľudstva. Formovali inteligenciu, kombinačné schopnosti, úsudok ľudí a poskytovali potrebnú zábavu a rozptýlenie.

4 Systém

Základ robotického herného systému tvorí rameno Lynxmotion AL5D so 6 stupňami voľnosti zakončené chápadlom otvoriteľným do šírky 3,175cm, obr.1. Sevromotory ramena sú riadené rozhraním SSC-32U, ktoré prijíma a interpretuje príkazy na sériovej linke pripojiteľnej cez adaptér do bežného portu USB a z programovacieho jazyka Java dostupného napríklad pomocou knižnice RXTX. Všetky odkazy možno nájsť na stránke Manipulátory na KAI (2015).

Obr. 1: Robotické rameno Lynxmotion AL5D s rozhraním SSC-32U.

Herná plocha veľkosti približne jedného hárku A4 sa ukladá na drevenú podložku, nad ktorou je na hliníkových profiloch upevnená bežná USB webkamera. Figúrky do hier sú vyrezané z drevených profilov, alebo vypožičané z bežných stolových hier. Systém ich rozlišuje na základe farby, sú určené intervalmi hodnôt vo farebnom modeli HSV a intervalom prípustnej veľkosti farebného segmentu v obraze. Vizuálnu informáciu spracováva samostatná za behu konfigurovateľná aplikácia vytvorená v C++. Pre jednoduchosť využíva na získavanie obrazu knižnicu OpenCV. S hlavným frameworkom komunikuje cez svoj štandardný vstup a výstup, ktoré sú frameworkom presmerované do komunikačnej rúry (pipe). Namiesto inverznej kinematiky pre určenie konfigurácií ramena, ktorú nám neskôr dodá diplomant, keď bude hotový, využívame jednoduchší spôsob. Pri definovaní pravidiel hry používateľ pomocou klávesnice riadi rameno robota a označí cieľové pozície, ktoré sa uložia do súboru. Tie sa potom využívajú počas bežného hrania hry na takto definovanej hracej ploche. Herná plocha je vždy pripevnená na rovnakej pozícii vzhľadom na súradnicový systém ramena robota. Priblíženie sa k figúrkam pred uchopením sa vykonáva v štyroch krokoch: rameno sa najskôr presunie nad danú pozíciu, potom sa priblíži kolmým pohybom nadol tak, aby bola figúrka medzi prstami chápadla, ktoré sa vzápätí uzavrie a v poslednom kroku sa opäť vráti na miesto niekoľko centimetrov nad figúrkou. Preto systém eviduje o každej pozícii dve sady nastavení pre servomotory: jednu pre pozíciu nad figúrkou a druhú pre jej pozíciu vhodnú na uchopenie. Problém hier, kde sa kamene umiestňujú do hry postupne, sme vyriešili množstvom prekrývajúcich sa políčok, na ktorých sú na začiatku uložené „všetky“ figúrky a tieto políčka majú v špecifikácii hry atribút *relevant=no*, čo znamená, že ich obsah sa pri kontrole korektnosti ťahu nekontroluje striktne, ale vezme sa ľubovoľné z prekrývajúcich sa políčok, ktoré ešte malo obsahovať presunutý kameň. V praxi teda na toto prekrývajúce miesto robotovi pred uskutočnením jeho ťahu vždy podáme ďalší kameň.



Obr. 2: Príklad hry pre jedného hráča (Frogs). Cieľom je vymeniť pozíciu zelených a hnedých žiab, ktoré môžu robiť iba krok alebo skok vpred.

5 Softvérový framework

Framework s3games je vytvorený v jazyku Java v prostredí Netbeans. **Z používateľského hľadiska** ide o

grafickú aplikáciu s hlavným riadiacim oknom, oknom na vizualizáciu pohľadu z kamery a oknom na vizualizáciu herného plánu s aktuálnym stavom hry. V ňom ľudský hráč robí svoje ťahy, ak si zvolil simulovanú verziu hry. Napokon aplikácia disponuje riadiacim oknom pre robotické rameno – s funkciami na pripojenie a inicializáciu robota, potvrdenie konca ťahu, ak je to potrebné a na priame riadenie ramena v režime definovania pozícií pre nový herný plán.

Z vývojárskeho hľadiska je framework rozdelený na 1) modul komunikácie s robotom a C++ aplikáciou na spracovanie obrazu kamery, 2) modul grafického používateľského rozhrania, 3) modul riadenia hry (základný engine) vrátane parsera jazyka a operátorov nad vnútornou reprezentáciou hry a 4) modul s rozličnými stratégiami hry, algoritmami prehľadávania stavového priestoru a heuristikami.

Špecifikácia hry je uložená v textovom súbore vo formáte Windows-like INI-súborov, rozdelenom na [*Pomenované Sekcie*] a dvojicami *premenná=hodnota* a časťou s voľným textom obsahujúcim definície funkcií a logických podmienok. Parsovanie premenných je priamočiare (nejde tu o nejaké premenné programovacieho jazyka, ale nastavenia rozličných parametrov a atribútov popisujúcich špecifikáciu hry). Definície funkcií a logických podmienok sú spracované dvojfázovo – najskôr sú vytvorené postupnosti lexém, ktoré sú následne interpretáciou gramatiky popisujúcej jazyk prevedené do internej reprezentácie výrazov, ktorá podstatne urýchľuje ich vyhodnocovanie počas hry, keďže nie je potrebné ich za behu interpretovať z podoby textových reťazcov. Z hľadiska umelej inteligencie je najzaujímavejší základný engine, ktorý v krátkosti vysvetlíme.

Hra je definovaná ako n-tica (*ET, LT, L, E, EX, SC, EG, GR*), pričom význam komponentov je nasledujúci:
ET – množina typov kameňov (element types)
LT – množina typov miest, kam je možné kameňe umiestňovať (location types)
L – množina miest, kde je možné kamene umiestňovať
E – množina kameňov a ich začiatočná poloha
EX – pomocné funkcie, ktoré sa môžu využívať v pravidlách, podrobnejšie ich vysvetlíme v stati o jazyku
SC – pravidlá o získavaní skóre v rozličných situáciách
EG – pravidlá o ukončení hry v rozličných situáciách
GR – pravidlá hry, podľa ktorých je možné ťahať.
Okrem toho k definícii hry patrí: parametre figúrok na ich rozpoznanie v obraze pre každý typ kameňov, obrázky, ktorými sa jednotlivé typy kameňov a typy políčok vizualizujú vo virtuálnej verzii hry, tvar a rozmery klikateľnej oblasti a relatívny posun každého zobrazeného obrázka, nastavenia servomotorov ramena pre dosiahnutie každého políčka a obrázok pozadia virtuálnej hracej plochy. Takto definovaná hra sa pri štarte hry inicializuje podľa začiatočných polôh všetkých kameňov a počas hry pozostáva **aktuálny stav hry** z n-tice (*ES, EL, P, O, CP, W, SC, ZI, K*), kde:
ES je funkcia *E->N* priraďujúca každému kameňu jeho číslo stavu
EL: *E->L* priraďuje každému kameňu miesto, na ktorom je práve položený
P: *L->E* je opačné zobrazenie, každému miestu priraďuje kameň, ktorý je na ňom práve položený (alebo null, ak tam žiaden nie je),
O: *E->N* priraďuje každému kameňu číslo hráča, ktorému daný kameň aktuálne patrí
CP je číslo hráča na ťahu
W je číslo vyhrávajúceho hráča (alebo -1 kým hra beží)
SC: *N->N* je dosiahnuté skóre každého hráča
ZI: *E->N* je zIndex jednotlivých figúrok a má zmysel len kvôli vizualizácii vo virtuálnej verzii hry, voliteľne možno jeho východziu hodnotu zadať aj v definícii hry
K je kontext hry, kde sú uložené hodnoty všetkých globálnych premenných, ktoré boli nastavené počas vyhodnocovania pravidiel.

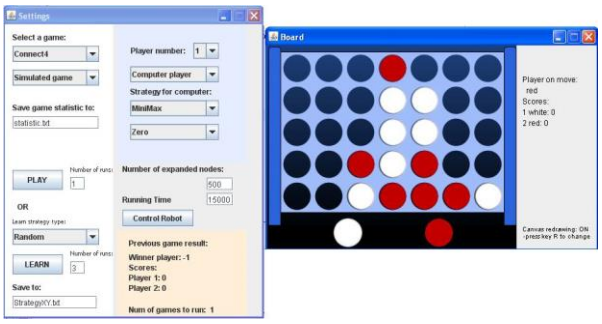
V mnohých hrách majú všetky kamene jediný povolený stav, možnosť zmeny stavu zodpovedá napríklad umiestneniu dekorácie na figúrku, prípadne slúži na zapamätanie stavovej informácie o priebehu hry, ktorá nie je vyjadrená iba pozíciou figúrky, napríklad: figúrka už bola v cieli a musí sa dostať naspäť do domčeka. Okrem stavu hry (trieda *GameState*) framework definuje triedu *Game*, ktorá riadi všeobecný priebeh hry: zabezpečuje striedanie hráčov v ťahoch a aktualizáciu stavu hry a abstraktnú triedu *Player* – ktorej konkretizáciami sú buď jednotlivé algoritmy umelej inteligencie, ktoré dokážu podľa aktuálneho stavu hry určiť nasledujúcu akciu, alebo triedy zastupujúce ľudského hráča – či už klikajúceho myšou vo virtuálnej verzii hry (*MousePlayer*), alebo presúvajúceho figúrky na reálnom hernom pláne rozpoznávanom kamerou (*CameraPlayer*). V prípade virtuálnej hry sa navrhované ťahy zobrazujú v okne a v prípade reálnej hry realizujú aj robotickým ramenom: zabezpečujú to metódy *GameWindow.setState(state)* a *Robot.moveRobot(nextMove)*. Podtrieda abstraktnej triedy *Player* musí definovať najmä metódu *Move move(GameState state, ArrayList<Move> allowedMoves)*, ktorá na základe aktuálneho stavu hry vyberie z množiny možných prípustných ťahov nasledujúci. Môže tiež definovať metódu *void otherMoved(Move move, GameState newState)*, pokiaľ chce byť detailne informovaná o ťahu súpera. Framework získava **množinu prípustných ťahov** tak, že sa pokúsi aplikovať každé pravidlo o ťahaní kameňov na každý kameň aktuálneho stavu a na každú voľnú cieľovú pozíciu na ploche. Tento operátor realizuje metóda *allPossibleMoves()* triedy *GameState*. Výsledný zoznam je potrebné odfiltrovať od množstva duplicitných stavov, ktoré sa líšia len na políčkach označených ako *relevant=no*. Ak daný triplet (kameň, odkiaľ, kam) pravidlu vyhovuje, cieľovú pozíciu pridá do množiny. Pre každý ťah hráča navrhovaný metódou prekrývajúcou abstraktnú metódu *Player.move()* sa podobným spôsobom overuje jeho korektnosť

approved = state.moveAllowed(nextMove). Ak nedovolený ťah vykoná ľudský hráč, je informovaný a má možnosť svoj ťah opraviť, v prípade automatického hráča je hra zastavená a hráč diskvalifikovaný z hry. Trieda *Player* obsahuje aj metódy na sledovanie maximálneho povoleného času na jeden ťah, ktorý je možné nastaviť v grafickom riadiacom okne aplikácie. **Pridanie nového typu algoritmu** teda znamená zadefinovanie novej triedy odvodenej od triedy *Player* (a zodpovedajúcej triedy odvodenej od triedy *Strategy*, ktorá je len adaptérom) a pridanie novej stratégie do zoznamu stratégii v *Strategy.availableStrategies()*. Okrem toho je možné **definovať heuristiky**, ktoré musia implementovať metódu určujúcu odhad kvality *double heuristic(GameState gameState, int forPlayer)* stavu hry pre hráča – napr. v prípade algoritmu A* ide o odhad vzdialenosti do cieľovej/výhernej konfigurácie z aktuálneho stavu. Heuristika je odvodená od triedy *Heuristic* a stačí ju pridať do zoznamu podporovaných heuristík v *Heuristic.availableHeuristics()*. **Pridanie nového typu hry** znamená len vytvorenie textového súboru s popisom hry a jej pravidiel a pridanie zodpovedajúcich obrázkov pre virtuálnu verziu hry, nastavenie pozícii v kamere a na servomotoroch ramena, zadefinovanie rozpoznávaných farieb pre jednotlivé typy figúrok a doplnenie do zoznamu hier *Controller.getGameNames()*. Vidíme, že navrhnutý a implementovaný framework je dostatočne všeobecný a okrem didaktickej hodnoty o hrách ako takých a prehľadávacích algoritmoch umožňuje experimenty s rôznymi typmi hier: za účelom testovania vyjadrovacej sily špecifikačného jazyka, alebo pre porovnávanie úspešnosti, prípadne časovej náročnosti jednotlivých prehľadávacích algoritmov a heuristík. Framework má predpripravenú podporu pre učiace algoritmy (napr. pre Reinforcement Learning), ale jej realizácia zatiaľ čaká na nasledujúce použitie tohto projektu.

6 Experimenty na S3

Pred začiatkom S3 sme do frameworku naimplementovali niekoľko základných algoritmov: *RandomGeneralPlayer* – ktorý pre ľubovoľnú hru generuje náhodné ťahy; *DepthFirstSearchPlayer* – pre hry pre jedného hráča, prehľadáva stavový priestor do hĺbky a vyberá ťah, ktorý smeruje k prvej nájdenej víťaznej konfigurácii; *BreadthFirstSearchPlayer* – pre hry jedného hráča, prehľadáva stavový priestor do šírky a vyberá ťah, ktorý vedie smerom k víťaznej konfigurácii, ktorá je najbližšie; *AStarPlayer* – ako BFSP, ale priestor prehľadáva efektívnejším algoritmom A* pomocou dodanej heuristiky, *MiniMaxPlayer* – pre hry pre dvoch hráčov, pričom vyberá ťah podľa algoritmu MINIMAX v strome konfigurácii hry ohodnoteného podľa toho, či hráč v danej konfigurácii vyhral, prehral, alebo remizoval. V našej modifikácii zvýhodňuje tie uzly, ktoré sú bližšie k východzej situácii. Tento algoritmus tiež dokáže využiť heuristiku ohodnocujúcu stav hry z pohľadu hráča na ťahu vtedy, keď ohodnotenie stavu hry v danom uzle nie je zistiteľné. Algoritmus prehľadá len takú časť priestoru, akú mu stanovený čas na jeden ťah umožní, potom urobí spätné ohodnotenie od listov ku koreňu a vyberie najlepší známy ťah; *MonteCarloPlayer* – vychádza z predpokladu, že prehľadávací priestor je natoľko rozsiahly, že ho aj tak nestačí celý ohodnotiť. Preto pre každý z nasledujúcich možných ťahov odohrá N kompletných náhodných hier a pozrie sa akú v nich má priemernú úspešnosť. Potom vyberie taký nasledujúci ťah, ktorý mal najlepšiu priemernú úspešnosť. Jeho výhodou je, že nepotrebuje konštruovať kompletný strom, ale vždy iba jednu cestu v prehľadávanom grafe a teda v porovnaní s MiniMax výrazne šetrí pamäť a teda potenciálne stihne preskúmať väčší prehľadávaný priestor, lebo vytváranie a uvoľňovanie rozsiahlych pamäťových štruktúr v jazyku Java nie je z časového hľadiska zadarmo. Naviac každú hru zahrá až do konca, takže používa informácie s vyššou vierohodnosťou.

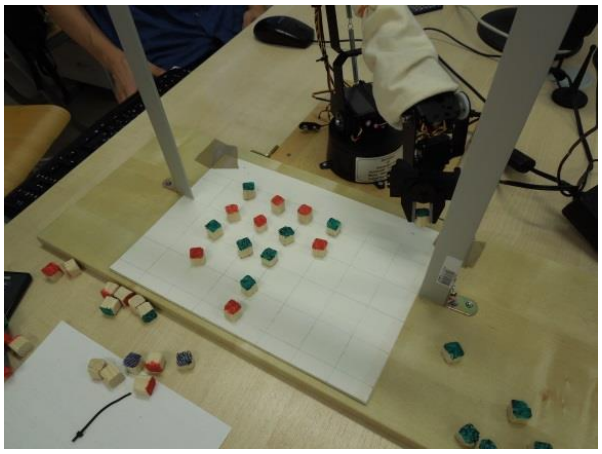
Študenti na S3 kompletne navrhli a vyšpecifikovali pravidlá hry *Connect4* (ľudovo známa ako padacie piškvorky) použitím nášho špecifikačného jazyka, zostrojili herný plán a naučili robota hrať proti človeku algoritmami, ktoré už vo frameworku boli implementované. Potom sa zamerali na ich modifikáciu a experimentálne porovnanie rozličných verzií, zbehnutie štatisticky relevantného počtu hier a vyhodnotenie a zdokumentovanie výsledkov.



Obr. 3: Riadiace okno aplikácie a virtuálna hracia plocha pre hru Connect4.

Testovali rôzne modifkácie algoritmov MiniMax a MonteCarlo. *MiniMax Stochastic*, kombinoval presnosť odhadu MiniMax s výhodami stochastického prehľadávania. Algoritmus sa vzdal ambície úplného prehľadania priestoru za cenu neotvárania niektorých ťahov. Táto modifikácia však viedla k fatálnym chybám. V prípadoch, keď algoritmus neotvoril práve taký ťah, ktorý bol pre neho prehrávajúci, tradičný MiniMax svoju výhodu dôsledného prehľadávania vždy využil. Keďže táto hra pozostáva z dlhej postupnosti ťahov a množstva otvorených hrozieb, pravdepodobnosť, že stochastický algoritmus bude nachytaný, bola priveľká.

Modifkácie algoritmu MonteCarlo vychádzali z toho, že algoritmus nebral do úvahy pravdepodobnosť dosiahnutia započítavaných výsledkov hier – čiže počet celkovo uskutočnených ťahov v danej náhodnej hre, kým niektorý hráč nevyhral a mieru vetvenia v jednotlivých stavoch počas hry. Preto sme vyskúšali deliť hodnotu výsledného uzla jeho „viditeľnosťou“ z koreňa – čiže pravdepodobnosťou, že hra sa z koreňa



Obr. 4: Robotické rameno hrá Connect4 proti človeku.

dostane do tohto cieľa pri náhodnom výbere hrán, berúc do úvahy stupeň vetvenia v súperových uzloch. V tomto prípade však boli príspevky víťazných hier hlbšie v strome s veľkým vetvením príliš malé, čo zvýhodňovalo horšie ťahy s inak vetvenou štruktúrou stromu. Iná testovaná modifikácia nebrala do úvahy stupeň vetvenia v súperových uzloch, ale iba hĺbku víťazneho uzla a so vzdialenosťou od koreňa ohodnotenie neklesalo tak prudko. Ako vidno z tabuľky 1, druhá metóda (MCR2) dosahovala v hrách proti prvej (MCR) lepšie výsledky.

	MCR	MCR2
MCR		31,93
MCR2	68,07	

Tab. 1: Percentá vyhraných hier medzi jednotlivými verziami algoritmov modifikovaných algoritmov MonteCarlo, priemer z 52-100 hier, každý hráč začínal v polovici hier. Podľa Cvijović, Unger, Corazza (2013).

	MM	MMS	MC	MCR2
MM		82,93	41,30	48,08
MMS	8,54		11,67	3,19
MC	54,35	88,33		51,72
MCR2	44,23	96,81	46,55	

Tab. 2: Percentá vyhraných hier medzi jednotlivými verziami algoritmov, priemer z 52-100 hier, každý hráč začínal v polovici hier. Podľa Cvijović, Unger, Corazza (2013).

Ďalej sme teda využili iba algoritmus MCR2 a porovnanie celkových výsledkov odohraných hier je v tabuľke 2. Z výsledkov vidno, že úspešnosť MCR2 voči stochastickému MiniMaxu bola významne lepšia ako pôvodného MonteCarlo, hoci jeho úspešnosť voči podstatne kvalitnejšiemu pôvodnému MiniMax bola o niečo nižšia. Podstatnejší (a prirodzený) záver tejto etudy je, že pokiaľ nepoužijeme priamo víťaznú stratégiu, každá stratégia bude mať rozličnú úspešnosť v závislosti od jej ad-hoc schopnosti nájsť slabé miesta v stratégii súpera. Na rozličných typoch hier a teda prehľadávacích stromoch s inými vlastnosťami budeme dostávať odlišné výsledky.

Vo frameworku boli špecifikované nasledujúce hry pre jedného hráča:

Frogs – pozri obr.1, toto bola kompletne realizovaná východisková hra pre jedného hráča pred S3

Puzzle8 – klasický puzzle na gride 3x3

RiverCrossing – hádanka o prechádzaní rieky cez deravý most

MasterMind – klasická hra Logik pre jedného hráča – cieľová konfigurácia je určená a dá sa zmeniť v špecifikácii. Toto je príklad chýbajúceho operátora generátora náhodných čísel, ktorý by umožnil vytvorenie inej náhodnej konfigurácie v každej hre, aktuálna verzia pravidiel je čisto deterministická.

Vo frameworku boli špecifikované nasledujúce hry pre dvoch hráčov:

Alquerque – historická verzia hry dáma – táto hra demonštruje vyjadrovaciu silu špecifikačného jazyka

TicTacToe – klasické piškovorky 3x3 pre 2 hráčov toto bola kompletne realizovaná východisková hra pre dvoch hráčov pred S3

Nim – odoberanie zápaliek – tradičná hra s optimálnou stratégiou založenou na zvyškových triedach

Reversi – kde kamene neustále menia farbu, čo sa realizuje zmenou ich stavu a hra má zmysel len vo virtuálnej verzii

Connect4 – kompletne realizovaná hra na S3

Squares – v dvoch verziách, hra kde používatelia striedavo spájajú mrežové body a každý, kto vytvorí jednotkový štvorec, získava bod

Skipping – je podobná u nás známej hre Halme, hrá sa na šachovnici

Mill – známa hra Mlyn, pričom je špecifikovaná len prvá fáza hry.

Celý framework a vytvorené špecifikácie hier sú open-source, dostupné na stránke Manipulátory na KAI (2015) a autori radi poskytnú podporu v prípade využitia v študentských projektoch, výuke, alebo v závečných prácach.

7 Jazyk pre špecifikáciu hry

Jadrom tohto príspevku je analýza špecifikačného jazyka navrhnutého pre popis stolových hier a ich pravidiel. Ako sme uviedli vyššie, hra je určená n-ticou (*ET, LT, L, E, EX, SC, EG, GR*). Prvé štyri časti len

popisujú vyššie pomenované množiny a nevyhnutné a voliteľné atribúty ich prvkov – detaily sa nachádzajú na stránke s popisom špecifikácie hry. Tu sa zameriame na posledné štyri, resp. tri, keďže EX – množina používateľom definovaných výrazov (expressions) definuje výrazy, ktoré môžu byť použité v SC, EG, aj GR.

Bodovanie hráčov (SC – scoring) obsahuje niekoľko trojíc vo formáte:

situation=výraz
player=číslo_hráča
score=výraz

každá trojica sa aplikuje na výpočet aktuálneho prírastku skóre určeného hráča za podmienky, že výraz situácie je platný. Napríklad, ak biely hráč hrá proti čiernemu, typicky je biely hráč číslo jeden a čierny je číslo dva. V hre Reversi po každom kroku vyhodnotíme skóre oboch hráčov takto:

situation=true
player=1
score=Stones(2)-SCORE(1)
player=2
score=Stones(1)-SCORE(2)

Pričom *Stones($HRAC)* je používateľom definovaný výraz, ktorý vypočíta počet kameňov zadaného hráča a *SCORE($HRAC)* je súčet skóre, ktoré zadaný hráč získal vo všetkých predchádzajúcich ťahoch. Tento rozdiel teda určí o koľko sa zvýšil počet kameňov zadaného hráča a o túto hodnotu sa celkové skóre zvýši.

Koniec hry (EG – end of game) pozostáva z dvojíc:

situation=výraz
winner=číslo_hráča

a znamená ukončenie hry v prípade splnenia daného výrazu. Víťazom sa stáva určený hráč. SC aj EG môže obsahovať výraz na pravej strane každého z riadkov.

Pravidlá ťahania kameňmi (GR – game rules) je množina pravidiel typu čo – odkiaľ – kam, pričom každé z nich je podrobne určené nasledujúcimi položkami (*v* = voliteľná položka):

name=meno_pravidla
element=určenie_kameňa
v state=určenie_stavu_kameňa
v player=číslo_hráča
from=určenie_východzieho_miesta
to=určenie_cieľového_miesta
condition=výraz
v awardPlayer=číslo_hráča
v withScore=výraz
v followup=výraz

v riadku *condition* je možné uviesť podmienku, ktorá musí platiť, aby sa pravidlo mohlo použiť a môže to byť ľubovoľný výraz. Riadky *awardPlayer* a *withScore* určujú hráča, ktorému sa majú za vykonanie kroku podľa tohto pravidla pridať body. Môžu taktiež obsahovať ľubovoľný výraz. Ak matchuje viacero pravidiel, vždy sa vyberie to, ktoré pridelí najviac bodov. Ľubovoľný výraz *followup* sa vyhodnotí vždy po vykonaní kroku podľa tohto pravidla. Napríklad v prípade preskočenia figúrky v hre Dáma sa táto figúrka z ihriska odstraňuje. V reálnej hre ju teda rameno robota vezme a odstráni, t.j. presunie na určené nerelevantné miesto. Generalizácia v našom druhu špecifikácie pravidiel je skrytá v tom, že kameň, jeho stav, hráč, východzie aj cieľové miesto môžu byť zadané s indexom, čiže napriklad *figurka($I)*, prípadne s dvoma či viacerými indexami: *figurka($A,$B)*. Takéto pravidlo aplikované na kameň s názvom napr. *figurka(2)* automaticky do premennej *$I* namapuje hodnotu 2. Vo výrazoch použitých v ostatných riadkoch pravidla sa potom tieto premenné môžu použiť a budú v nich zodpovedajúce hodnoty, napr. *$I==2*. Scoping premenných je výlučne globálny, sú uložené v tzv. kontexte reprezentovanom triedou *Context*, ktorej inštancia sa vytvorí na začiatku hry a zotrvá do jej konca. Uveďme si dva príklady hier s vysvetlením pravidiel:

Toto sú pravidlá hry Frogs pre jedného hráča, v ktorej si žabky vymieňajú pozície, pozri obr.2:

name=greenStep
element=gf($J)
from=s($K)
to=s($L)
condition=$L==$K+1

name=greenJump
element=gf($J)
from=s($K)
to=s($L)
condition=StoneOccupied($K+1) AND ($L==$K+2)

name=redStep
element=rf($J)
from=s($K)
to=s($L)
condition=$L==$K-1

name=redJump
element=rf($J)
from=s($K)
to=s($L)
condition=StoneOccupied($K-1) AND ($L==$K-2)

Druhým príkladom je hra River Crossing pre jedného hráča, kde 4 ľudia, ale najviac dvaja naraz, musia prejsť cez most nad riekou pomocou jediného lampáša. Každý z nich potrebuje rozdielny čas: 1,2,5, alebo 10 minút. Časy interpretujeme ako záporné hodnoty a odrátavame ich od začiatočného skóre 50 bodov. Pri mostíku sú po dve miesta, kde sa zhromažďujú pred prejdením rieky: *leftstack(1-2)*, *rightstack(1-2)*. Prejdenie cez mostík zariadime symbolicky preložením lampáša, poprekladanie figúrok už zabezpečia výrazy v riadku *followup*:

name=toLstack
element=person($J)
from=left($K)
to=leftstack($S)
condition=NOT EMPTY("lightLeft")

name=toRstack

```
element=person($J)
from=right($K)
to=rightstack($S)
condition=NOT EMPTY("lightRight")

name=toLeft
element=lg
from=lightRight
to=lightLeft
condition=CheckStackR
awardPlayer=1
withScore=Min(timeOf("rightstack(2)"),timeOf("rightstack(1)"))
followup=crossToLeft

name=toRight
element=lg
from=lightLeft
to=lightRight
condition=CheckStackL
awardPlayer=1
withScore=Min(timeOf("leftstack(2)"),timeOf("leftstack(1)"))
followup=crossToRight
```

Keďže už máme aspoň približnú predstavu o špecifikácii pravidiel hry, môžeme sa konečne zamerať na jadro – **jazyk výrazov,** na ktoré sa v pravidlách odvolávame.

Výrazy sú buď jednoriadkové anonymné, alebo viacriadkové pomenované – s 0 alebo viac formálnymi parametrami. Výsledkom vyhodnotenia výrazov je nejaká hodnota. Hodnoty majú implicitný typ: **string/symbol**, ktorý môže byť aj indexovaný číslami, alebo premennými, napr. "hello", "hello(3)", "hello($I,$J)", **číselná hodnota**, napr. 3, **množina**, napr. {1,2,"hello"}, **logická hodnota** true/false, prípadne prázdny string, či prázdna množina – "" a {}.

Ak je výsledkom vyhodnotenia výrazu **false**, hovoríme, že výraz nie je splnený, inak je splnený – a to aj keď výsledkom nie je true, ale iná hodnota. Výsledkom vyhodnotenia viacriadkových výrazov je buď *false*, ak sa niektorý riadok vyhodnotí ako *false* – a vtedy sa nasledujúce riadky už nevyhodnotnia, alebo hodnota na ktorú sa vyhodnotí posledný riadok – z logického hľadiska tvoria viacriadkové výrazy konjunkciu medzi jednotlivými riadkami a na všetky výrazy sa možno dívať aj ako na logické formuly. Vďaka tomu dokážeme pohodlne kontrolovať sadu potrebných podmienok, ktoré majú platiť pri aplikácii pravidla bez potreby ďalšieho zbytočného syntaktického cukru bez straty čitateľnosti a zrozumiteľnosti. Na druhej strane zavádzame pomerne bohatú sadu operátorov a primitívnych funkcií jazyka, pričom niektoré z nich majú aj bočné efekty – je to až nevyhnutné, pretože v mnohých hrách je potrebné priebežne modifikovať stav hry aj inak, ako len čisto presunutím kameňa z iniciatívy hráča. To však nebráni písať pekné logické formuly, ak to jednoduchosť hry dovoľuje. Tabuľka 3 obsahuje stručný prehľad operátorov, tabuľka 4 zoznam preddefinovaných funkcií a tabuľka 5 zoznam preddefinovaných funkcií s bočným efektom. Pozrime sa na príklady viacriadkových výrazov použitých v pravidlách hier uvedených vyššie. V definíciach výrazov sa na jednotlivých riadkoch môžu nachádzať biele znaky (medzera, tab) na ľubovoľnom mieste.

```
FrogsAtHome
  FORALL($J,1,3,ELTYPE(CONTENT("s($J)")) == "redFrog")
  FORALL($J,5,7,ELTYPE(CONTENT("s($J)")) == "greenFrog")
END

StoneOccupied($POS)
  $POS > 0
  $POS <= 7
  NOT EMPTY("s($POS)")
END

CheckStackL
  (NOT EMPTY("leftstack(1)")) OR (NOT EMPTY("leftstack(2)"))
END

moveToRight($EL,$FROM)
   $I = INDEX($EL)
   MOVE($EL,$FROM,"right($I)")
END

crossToRight
  $M = CONTENT("leftstack(1)")
  $N = CONTENT("leftstack(2)")
 IF(NOT EMPTY("leftstack(1)"), moveToRight($M, "leftstack(1)"), true)
 IF(NOT EMPTY("leftstack(2)"), moveToRight($N, "leftstack(2)"),true)
END

Min($NUM1,$NUM2)
  IF($NUM1 < $NUM2, $NUM1, $NUM2)
END

timeOf($PLACE)
  $WHO=ELTYPE(CONTENT($PLACE))
  $PTS=0
 PtsMother($WHO) OR PtsFather($WHO) OR PtsEmily($WHO) OR PtsBob($WHO) OR true
  $PTS
END

PtsMother($W)
  $W == "mother"
  $PTS = (-10)
END

PtsBob($W)
  $W == "bob"
  $PTS = (-1)
END
```

Prvý výraz *FrogsAtHome* kontroluje cieľovú konfiguráciu, teda či sa na všetkých políčkach *s(1)..s(3)* nachádzajú červené žaby a na všetkých políčkach *s(5)..s(7)* sa nachádzajú zelené žaby. *StoneOccupied* overuje, či požadované miesto je na

Operator	**t1**	**t2**	**Description**
val1 == val2	a	l	values equal
val1 != val2	a	l	values not equal
constant	a	a	value of that constant
val1 < val2	i	l	less than
val1 > val2	i	l	more than
val1 <= val2	i	l	less or equal than
val1 >= val2	i	l	more or equal than
val1 + val2	i	i	add
val1 - val2	i	i	subtract
val1 * val2	i	i	multiply
val1 / val2	i	i	integer division
val1 % val2	i	i	integer quotient
ABS val	i	i	absolute value
s1 ISPARTOF s2	s	l	subset
val IN set	as	l	is member
s1 EXCEPTOF s2	s	s	set difference
s1 UNION s2	s	s	set union
s1 INTERSECT s2	s	s	set intersection
val1 AND val2	l	l	logical AND
val1 OR val2	l	l	logical OR
NOT val	l	l	logical NOT
IF(v1,v2,v3)	la	a	conditional, result is v2 if v1 is true, otherwise v3
FORALL ($X,X1,XN,val)	ia	a	val evaluates to true for all values of variable $X from X1 to XN
FORSOME ($X,X1,XN,val)	ia	a	val evaluates to true for at least one value of variable between X1 and XN incl.
EXPRNAME	-	a	evaluate the nuladic expression with the given name
EXPRNAME (v1,...,vN)	a	a	evaluate the expression with the given name and arguments, which are arbitrary expressions
$VAR = val	a	l	assignment, result is true
$VAR	-	a	value of the variable

Tab. 3: Zoznam operátorov jazyka výrazov. Typy sú i – celé číslo, a – ľubovoľný, l – logická hodnota, s – množina. Stĺpec t1 určuje typy argumentov, t2 je typ výslednej hodnoty.

LOCTYPE(loc)	g	type of location
ELTYPE(elmnt)	g	type of movable element
STATE(elmnt)	i	state of movable element
LOCATION(elmnt)	g	current location of element
CONTENT(loc)	g	current content of location
EMPTY(loc)	l	location is empty
INDEX(string)	i	first index in a given string "name(123)" -> 123
INDEXA(string)	i	i-th index in a given string
UNINDEX(string)	g	name without indexing "name(123)" -> "name"
OWNER(elmnt)	i	current owner of element
PLAYER	i	current player on move 1+
SCORE(player)	i	current score of the player
ZINDEX(elmnt)	i	current ZINDEX of the specified element

Tab. 4: Zoznam zabudovaných primitívnych funkcií. Typ g označuje string.

MOVE (elem,loc1,loc2)	l	moves element from loc1 to loc2, false if occuppied or not present
SETOWNER (elem,int)	t	set the owner of the element to given player number
SETSTATE (elem,int)	t	set the state of the elment
SETZINDEX (elem,int)	t	set the zIndex of the element
NEXTPLAYER	t	finish the move of this player

Tab. 5: Zoznam zabudovaných primitívnych funkcií s bočným efektom. Typ t označuje logickú hodnotu true.

CheckStackL je nutná podmienka na prenesenie lampáša z ľavého brehu rieky na pravý, lebo aspoň jedna z pozícií, kde sa postavičky pripravujú na cestu cez most, musí byť obsadená. *MoveToRight* zabezpečí prechod jednej postavičky cez most a jej umiestnenie na jej cieľové vyhradené miesto (s rovnakým indexom ako je v jej mene) na pravej strane rieky. *CrossToRight* zabezpečuje prechod všetkých pripravených postavičiek na druhú stranu pomocou výrazu *MoveToRight*. *Min* je príklad používateľom definovanej aritmetickej funkcie a využitia podmienkového príkazu. *timeOf* a zostávajúce dve

funkcie zisťujú čas, ktorý potrebuje na pechod cez rieku postavička, ktorá stojí na určenom políčku, využívajúc, že vyhodnocovanie argumentov operátora *OR* sa zastaví akonáhle sa jeden z nich nerovná *false*.

8 Prepojenie jazyka s frameworkom

Špecifikácia každej hry je uvedená v jedinom textovom súbore. Parserom je načítaná a prevedená do vnútornej reprezentácie, ktorá je efektívne interpretovaná počas hrania hry. V aktuálnej verzii sa rameno po vykonaní každého ťahu vráti do pohotovostnej polohy – jednak aby uvoľnilo priestor a svoj ťah mohol vykonať človek a jednak preto, aby kamera mala netienený výhľad na celú scénu. Obrovskou výzvou bolo zladiť s reálnym svetom tie typy hier, kde nie sú všetky figúrky naraz na ploche tak, aby to výrazne nepoškodilo prehľadávacie algoritmy zbytočnou kombinatorickou explóziou. Atribút políčka *relevant=no* teda spôsobí nielen to, že si dané miesto kamera nevšíma. Okrem toho sú ťahy z nerelevantných políčok rovnakého typu na rovnaké cieľové políčko s kameňom rovnakého typu (hoci iného mena) považované za ekvivalentné. Problém nastáva, keď potreujeme zadefinovať ťahy na rôzne nerelevantné políčka rovnakého typu, ktoré nemajú byť považované za rovnaké, alebo keď treba vygenerovať cieľové políčka, ktoré kamera už nevidí alebo ignoruje – a to sú len ukážky príkladov, kedy nami zvolená abstrakcia (ako každá iná) naráža na svoje hranice vyjadrovacej sily, hoci vždy existuje nejaký spôsob, ako sa s problémom vyrovnať.

9 Alternatívny prístup

Prebiehajúca diplomová práca Petra Pukančíka, ktorej pôvodným cieľom bolo zostrojiť framework nie nepodobný nášmu, sa postupne zamerala na alternatívny spôsob špecifikácie pravidiel. Náš spôsob uvažovania bol nasledujúci: ak hráč vykonal ťah, postupne prejdeme cez zoznam všetkých pravidiel a zistíme, či niektoré z nich vyhovuje. Ak áno, ťah akceptujeme, inak ho odmietneme. Viac práce však máme, keď z danej konfigurácie hry máme vygenerovať zoznam všetkých možných ťahov: vtedy skúšame aplikovať všetky pravidlá na všetky položené kamene a všetky možné voľné cieľové pozície.

Alternatívny prístup k tvorbe pravidiel je založený na ASP solveri logického programovania. Pre zadanú figúrku a jej pozíciu priamo vygeneruje všetky prípustné ťahy. Od tohto prístupu očakávame vyššiu efektivitu a zostávame v očakávaniach výsledkov porovnania oboch metód aj z hľadiska prehľadnosti a zrozumiteľnosti zápisu špecifikácie.

10 Záver

Práca je prvá podrobnejšia informácia o univerzálnom systéme na hranie stolových hier pozostávajúceho z robotického ramena, kamery a softvérového frameworku, ktorý sme vyvinuli na Katedre aplikovanej informatiky, FMFI UK. Systém dovoľuje zadefinovať nové hry pomocou deklaratívneho špecifikačného jazyka s procedurálnymi prvkami a len na základe tejto špecifikácie systém dokáže hrať zadefinovanú hru proti človeku na netriviálnej úrovni. Experimenty so živými subjektami na letnej škole vedy S3 v Chorvátskej Požege ukázali, že náš systém bez problémov dokáže poraziť priemerného ľudského hráča.

Na sade expriementov realizovaných na S3 sme demonštrovali, že navrhnutý framework je vhodný na porovnávanie algoritmov umelej inteligencie, výskum jazykov na špecifikáciu a interpretáciu hier, a výuku a študentské projekty a záverečné práce.

V ďalších projektoch s frameworkom S3games by sme radi vyskúšali učiace sa algoritmy, ktoré budú zlepšovať schopnosť systému hrať hry na základe získaných skúseností z hier. Plánujeme zaintegrovať modul inverznej kinematiky pre automatické zadefinovanie pozícií na hracom pláne z obrazu a pokračovať vo vylepšovaní impelemtnovaných algoritmov a ponuke hier, ktoré systém dokáže hrať.

Poďakovanie

Tento príspevok vznikol za podpory grantovej agentúry KEGA v rámci grantovej úlohy 076UK-4/2013. Hardvér pre robotické rameno bol zaobstaraný vďaka finančnému daru od Nadácie Alexandra von Humboldt v roku 2010. Úprimné poďakovanie patrí Mgr. Zuzane Koyšovej, ktorá sa na S3 podujala na prevzatie úlohy hlavného vedúceho projektu, podieľala sa na implementácii frameworku a vytvorení špecifikácie viacerých hier.

Literatúra

Cvijović, Unger, Corazza (2013). The Art of Playing Games. Student session at 55th International Symposium ELMAR-2013, Zadar.

Petrovič (1997). Design and Play in Comenius Logo: A microworld for designing and exploring games. V zborníku Sixth European Logo Conference EUROLOGO'97.

Manipulátory na KAI (2015). Odkazy na všetky relevantné knižnice, špecifikácie a dokumentáciu a zdrojové kódy impelemntovaného frameworku, Katedra aplikovanej informatiky, FMFI UK Bratislava, dostupné online: http://dai.fmph.uniba.sk/projects/manipulators