Středoškolská odborná činnost

Umělá inteligence pro hru 3D piškvorky

Martin Beneš

Brno 2014

Středoškolská odborná činnost

Obor 18 – Informatika

Umělá inteligence pro hru 3D piškvorky

Artificial intelligence for 3D tic-tac-toe

Autor: Martin Beneš
Škola: Gymnázium Brno - Řečkovice, Terezy Novákové 2, 621 00
Kraj: Jihomoravský
Konzultant: Mgr. Jan Herman

Prohlášení

Prohlašuji, že jsem svou práci vypracoval samostatně, použil jsem pouze podklady (literaturu, SW atd.) uvedené v přiloženém seznamu a postup při zpracování a dalším nakládání s prací je v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) v platném znění.

V ..... dne ..... podpis: .....

Poděkování

Děkuji všem pedagogům a studentům za věcné připomínky a cenné rady, které mi poskytovali během vytváření práce a programu, především pak svému internímu konzultantu Mgr. Janu Hermanovi, bez jehož ochotné pomoci by celý projekt nemohl vzniknout.

Anotace

Tato práce se věnuje tvoření umělé inteligence pro hru 3D piškvorky. V první části práce je rozebrána hra piškvorky, historie a podtypy. Poté je obdobně analyzována i hra 3D piškvorky. V této části jsou také vysvětleny některé pojmy, které jsou užity v praktické části práce, kde je pomocí nich popsán mnou vytvořený program. Závěr práce obsahuje souhrn poznatků a náhled budoucího vývoje.

Práce by měla posloužit těm, které zajímá problematika umělé inteligence deskových her. Po přečtení by měl být každý obeznámen s programem, principy použité umělé inteligence a pravidly hry 3D pišvorky.

Klíčová slova: 3D piškvorky; algoritmus; umělá inteligence

Annotation

This paper describes the process of creating an artificial intelligence for the 3D tic-tac-toe. In the first part, there is an analysis of the game 5-in-row, as same as 3D tic-tac-toe. Also, certain terms from branch of an algorithmization, which are used to decribe of program in the second part, are defined here. The second part, as it has been already said, consist description of the program I created myself. Finally, all the findings are summed up and plans for the future development are set.

The paper should help people, who are interested in implementation of artificial intelligence for board games. After reading the paper, everyone should be familiar with the program, principals of used artificial inteligence and the rules of the 3D tic-tac-toe.

Key words: 3D Tic-Tac-Toe; algorithm; artificial intelligence

Obsah

Úvod

Piškvorky. Není snad nikdo, kdo by někdy navštěvoval školu a neznal piškvorky. V této práci bude řeč o speciálním typu piškvorek, o trojdimenzionálních piškvorkách, zkráceně 3D piškvorkách, konkrétně o vytvoření algoritmu pro počítačovou inteligenci při hraní 3D piškvorek. *„3D piškvorky jsou zajímavá strategická hra, která prověřuje prostorové vnímání hráčů, zejména schopnost představit si přímky v různých prostorových konfiguracích.“* Cituji z časopisu Mozkolam. Hraní piškvorek tedy rozhodně není žádná oddechová hra. Kromě orientace v prostoru je třeba také všímavost, předvídavost, strategie a samozřejmě štěstí.

Spousta programátorů se v minulosti snažila vymyslet jednoduchý algoritmus, který by vytvořil dokonalého soupeře skrze počítačový program. Zatím se to ale nikomu úplně nepodařilo. Vymyslet algoritmus pro 3D piškvorky dobře hrající umělou inteligenci by bylo samozřejmě nesčetněkrát jednodušší, ale dalo by se to provést na podobném základu, jaký zatím existuje pro šachy. Moje myšlenky při tvoření se tedy budou ubírat po stopách Turinga a Shannona.

Na internetu je zveřejněno mnoho verzí 3D piškvorek. Existuje spousta flashových her, např. hra FourSight – 3D Tic Tac Toe na serveru *http://wwwmathisfun.com*. Nejstarší je ale verze, vydaná v roce 1980 firmou Atari pro osmibitový počítač Atari 2600, obal této hry vidíte na obrázku 1. Tuto verzi se mi podařilo opatřit, pochopitelně se mi ji ale nepovedlo spustit na mém 64 – bitovém počítači, musel jsem tak učinit v emulátoru DOSBox.

Obrázek 1: Hra 3D Tic-Tac-Toe pro Atari.

Cíl práce

Hlavním cílem mé práce bylo vytvořit program, obsahující umělou inteligenci, schopnou hrát proti lidskému soupeři. Do tohoto programu jsem si usmyslel umístit základní položky, aby program fungoval. Rozhodl jsem se nabídnout dvě úrovně obtížnosti, výběr symbolu a jednoduché uživatelské prostředí. Zmíněné úrovně obtížnosti byly vzaty z algoritmů z oboru počítačových šachů, pro potřeby 3D piškvorek zjednodušeny a upraveny. Celý program včetně techto algoritmů bude samozřejmě v práci detailně analyzován.

V teoretické práci jsem si vytyčil objasnit problematiku piškvorek a 3D piškvorek. Budou stanoveny základní pojmy a vysvětleny pravidla. U 3D piškvorek bude objasněno zařazení mezi ostatní deskové hry. Práce také měla obsahovat stručný přehled podtypů piškvorek pro úplnost informací, jelikož 3D piškvorky vycházejí z několika modifikací této starobylé stolní hry.

Část I
Teoretická část

1 Piškvorky

Piškvorky jsou obecně známá strategická hra, kde spolu na čtverečkovaném papíře soupěří dva hráči. Ti se střídají v kreslení symbolů, jeden má křížek, druhý kolečko. Hráč, který jako první vytvoří řadu x značek, vyhrává.

1.1 Historie hry

Původ deskových her vznikl už před několika tisíci lety, pravděpodobně nezávisle na sobě v různých částech světa. První zmínka pochází z Číny z 2. tisíciletí př. Kristem. Již v roce 1400 př. n. l. byla obdoba této hry známa v Egyptě. Z této původní hry se vyvinuly hry jako go nebo alquerque[1].

1.2 Podtypy piškvorek

Uložení kamenů v jedné řadě bylo inspirujícím motivem pro vznik mnoha her, v nichž soupeří dva hráči. V některých hrách je hráč omezen, například gravitací, nebo naopak dostává k dispozici třetí rozměr, čímž se otevírá řada nových možností. 3D piškvorky jsou nejmladší typ piškvorek, čerpají spoustu věcí od jiných deskových her, bude jim věnována samostatná sekce. Stručný přehled podtypů piškvorek naleznete v tabulce 1. Každému z podtypů poté věnuji jeden paragraf, kde krátce objasním blíže pravidla.

Název	**Pole**	**Vítězství**	**Poznámka**
Piškvorky	Neomezená	5 a víc	-
Gomoku	15x15	právě 5	-
Rendžu	15x15	právě 5	Fault: 2x 3, 2x 4, přesah
Irensei	15x15	7 (začínající právě 7)	Fault: přesah
Tic-tac-toe	3x3	3	-
Šestvorky	Neomezená	6 a víc	Dva symboly najednou
Gravitační piškvorky	6x7	4 nebo 5	Gravitace
Pentago	4x(3x3)	5	Otáčení poddesek
Kvantové piškvorky	3x3	-	Model kvantové fyziky
3D piškvorky	4x4x4	4	3D kombinace

Tabulka 1: Stručný přehled podtypů piškvorek.

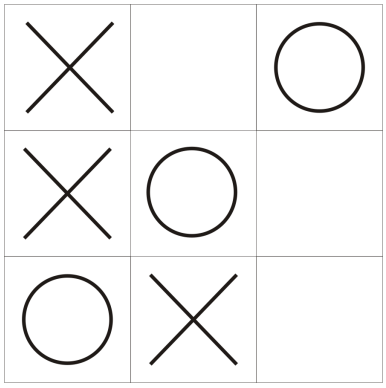
Gomoku Gomoku je předchůdce všech deskových her. Hraje se desce 15x15 políček, nekreslí se typické křížky a kolečka, ale na hrací pole se pokládají různobarevné kameny a výherná kombinace je právě 5 (tedy při překročení hráč nevyhrává, hra pokračuje dál).

[1] Alquerque, neboli el-Quirkat je obdoba dámy. Z této arabské hry se vyvinuly hry jako mlýn, dáma a zřejmě i šachy.

Renju Podobné gomoku, obsahuje pravidla znevýhodňující začínajícího hráče. Ten nesmí během hry vytvořit (záměrně, omylem, ani v obraně) vidličku, dvakrát nezablokovanou trojičku nebo čtveřičku. Navíc nesmí přesáhnout. Druhý hráč znevýhodněn není, přesah se mu počítá jako vítězství.

Irensei Irensei je hra, velmi podobná gomoku. Vítěznou kombinací je 7, začínající hráč nesmí vytvořit přesah.

Tic-tac-toe Velmi oblíbené jsou piškvorky typu tic-tac-toe, také známé pod názvem Noughts and Crosses, nebo Xs'n'Os, zejména tedy v USA, Velké Británii, ale i ve Francii. Hraje se na poli 3x3, vítězí 3 symboly v řadě. Částečně z nich vychází i 3D piškvorky. Hru můžete vidět na obrázku 2.



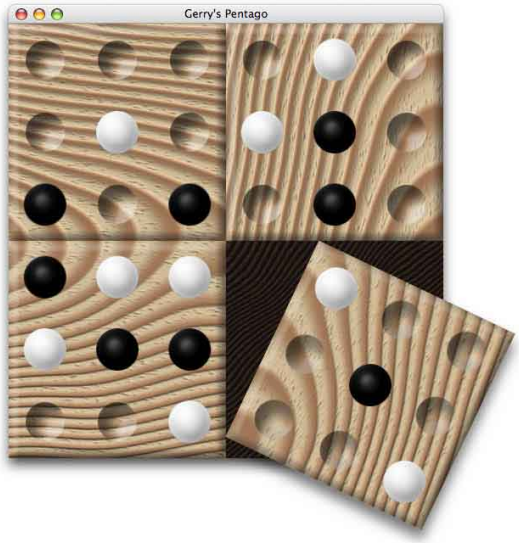
Obrázek 2: Hra Tic-Tac-Toe.

Šestvorky Vítězí 6 symbolů v řadě, během kola hráč zakresluje dva symboly najednou. U této modifikace piškvorek je znevýhodnění začínajícího hráče jen málo zasahující do samotné hry – začínající hráč zakreslí jednu značku, druhý dvě. Dále již každý zakresluje ve svém tahu dva symboly najednou.

Gravitační piškvorky Connect4, u nás též gravitační piškvorky, lze hrát na papíře, ale efektnější jsou se speciálním vertikálním herním „plánem", postihujícím prvkem je zde, jak název napovídá, gravitace. Byly proslaveny firmou Milton Bradley Company[2]. Jak anglický název prozrazuje, je třeba 4 (a víc) symbolů k vítězství.

Pentago Další z novějších podtypů piškvorek se jmenuje pentago. Pro hraní pentaga potřebuje hráč speciální otočnou hrací desku. Hraje se na 6x6, na 4 poddeskách (3x3), jak si můžete všimnout na obrázku 3. Hráč má ve svém tahu možnost kromě položení kamene i pootočení jedné ze čtyř poddesek.

[2]Milton Bradley Company (zal. 1860) byla americká společnost, vydávající deskové hry. Mezi nejznámější hry patřily Twister, Jenga nebo Connect4. Firma byla sloučena s Hasbro (1984).



Obrázek 3: Hra pentago.

Kvantové piškvorky Kvantové piškvorky jsou verzí piškvorek, podobnou tic-tac-toe s tím rozdílem, že tahy jsou suprapozicemi[3]. Hra slouží pro znázornění některých jevů v kvantové fyzice.

Qubic Qubic v sobě slučuje trojrozměrnost a gravitaci gravitačních piškvorek. Je hrán na speciálním dřevěném modelu, obsahujícím do čtverce 4x4 uspořádané tyčky, připevněné na destičce, na které se navlékají dřevěné kuličky, které zde nahrazují křížky a kolečka. Ty se při navléknutí dostanou vlivem gravitace na nejnižší možnou pozici na dané tyčce. Výherní kombinace jsou 4 kuličky, uspořádané v řadě v povoleném směru[4].

1.3 Soutěžní piškvorky

Piškvorky se hrají i soutěžně, například v rámci školní týmové soutěže pIšQworky. Minulý rok hrálo pIšQworky 27 000 studentů z 516 škol po celé České republice a na Slovensku. Soutěž pořádá občanské sdružení Student Cyber Games.

[3] V daném poli potencionálně najednou je i není soupeřova značka.

[4] Vítězná kombinace 3D piškvorek je hrana, stranová i tělesová úhlopříčka, vertikálně, horizontálně i diagonálně, podobně jako u 3D piškvorek.

2 3D piškvorky

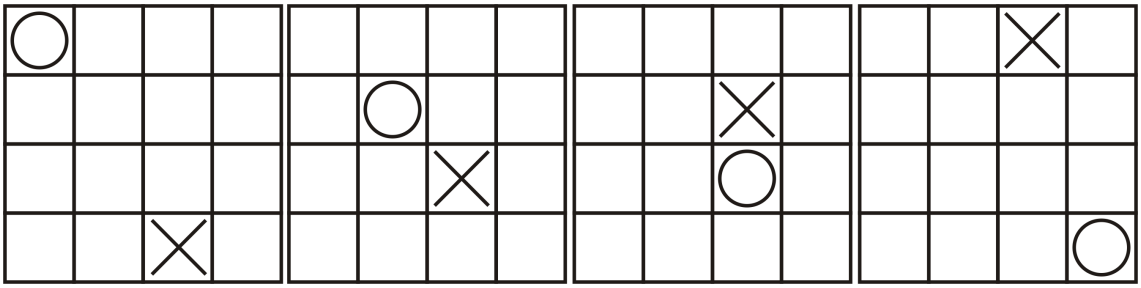
3D piškvorky jsou abstraktní hra s tužkou a papírem pro dva, ale i více hráčů, která se $N \times N \times N$ polích, kde N je větší než 3 a sudé, a zároveň ústřední motiv této práce. Existuje i úprava 3D piškvorek pro liché N, minimálně pak 3, kde středové pole *(N + 1) / 2][(N + 1) / 2][(N + 1) / 2]* neexistuje, resp. se na něj nesmí zapsat žádný symbol. Tím se ale výrazně sníží počet možných tahů. V této práci budu ale mluvit výhradně o 3D piškvorkách na poli 4x4x4, což je nejvhodnější pole na hru 2 hráčů.

Tyto pole jsou seřazeny nad sebou, tvoří krychli, kde jsou povolené tahy i skrze vrstvy. Na obrázku 4 můžete vidět vzhled pole a ukázku možných výherních kombinací skrz krychli.

3D piškvorky jsou vcelku nová hra, jako takové kombinují několik prvků z již existujících modifikací, čímž pádem se od původního gomoku celkem podstatně liší. Celkový nápad hry je samozřejmě postaven na piškvorkách a gomoku. Základní odlišující prvek je zde trojrozměrnost, tu si propůjčují z qubicu. Dále je zde již zmíněná omezenost plánu, kterou zase berou z tic-tac-toe. Celkově jsou nová nadstavba na starší piškvorky, třetí rozměr dodává nebývalé kouzlo, hra má jinou atmosféru. Oproti některým jiným modifikacím piškvorek nejsou třeba žádné další pomůcky, jen tužka a papír. Hra má velmi odlišnou atmosféru od piškvorky, vše je nahuštěno na malém prostoru a to může hráče zmást. A to vše činí z 3D piškvorek hru třetí generace a staví je vysoko nad úroveň spousty stolních her, piškvorky nevyjímaje.

2.1 Výherní kombinace

Hra přináší mnohem více druhů výherních kombinací. Oproti původním čtyřem v gomoku je jich zde třináct[5]. Celý systém hry je o to lepší, že pole je omezeno na 4 čtverce o 4 řádcích a 4 sloupcích, takže celých 72 možných výher[6] je na minimálním prostoru, což snižuje přehlednost celého plánu a nutí hráče více přemýšlet.



Obrázek 4: Hra 3D piškvorky.

[5] 3 po hranách, 6 po stranových a 4 po tělesových úhlopříčkách.
[6] $(3 \times 16) + (6 \times 4) + (4 \times 1)$

3 Herní algoritmy

3.1 Základní pojmy

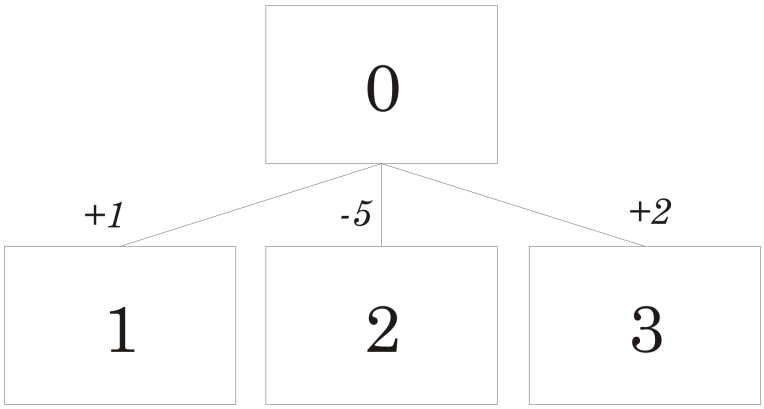
Algoritmus Algoritmus je přesný návod, kterým lze vyřešit daný typ problému. Musí splňovat 5 podmínek – musí být přesně definovaný, obecný, s konečným počtem jednoduchých kroků a musí z něj vystupovat výsledek.

Časová složitost algoritmu Jak bylo řečeno, algoritmus musí být konečný. Jenže to by samo o sobě nestačilo. Proto se definuje tzv. časová složitost algoritmu, která udává vztah mezi počtem vložených výsledků (n) a přibližným počtem potřebných operací k tomu, aby algoritmus mohl odevzdat výsledek.

Rekurze Programátorská technika, která definuje objekt pomocí sebe sama. Ve sledu kroků, který jsme pojmenovali X se rekurze projeví příkazem „proveď X“.

3.2 Naive

Jednoduchý algoritmus, který prohledá všechny možné tahy v dané pozici a vybere ten, který má nejlepší hodnocení. Na obrázku 5 je vidět, že problém, před kterým počítač stojí, je velmi snadno řešitelný. Jde tu o srovnání několika čísel. V realite se může jednat ale až o několik desítek čísel (podle uplynulých kol hry). Výsledkem samozřejmě bude vykonání tahu na pozici 3.



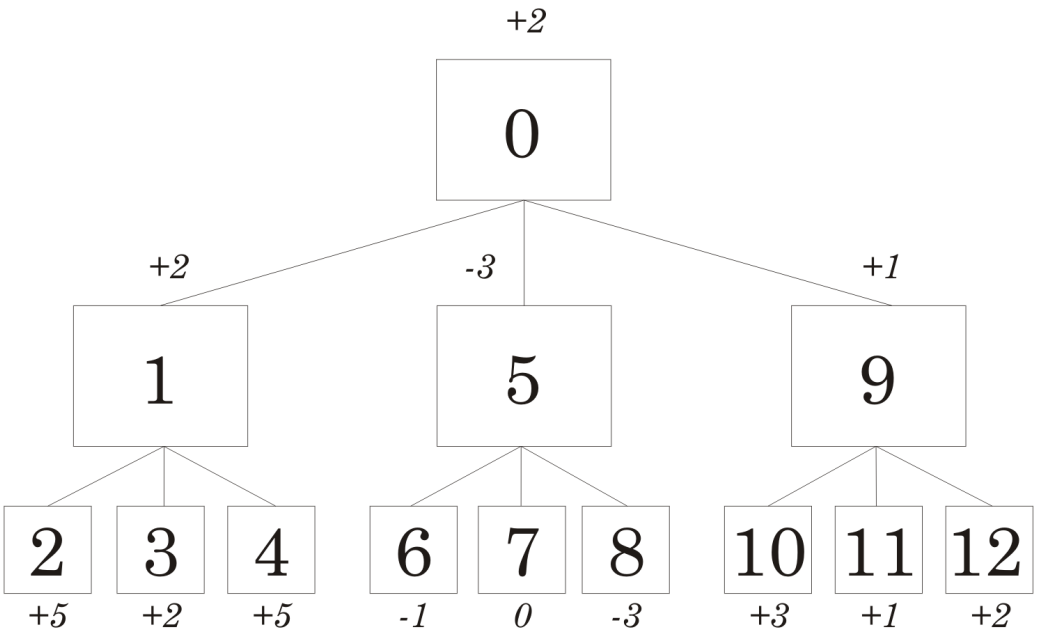
Obrázek 5: Grafické znázornění vyhledávacího stromu Naive.

3.3 Minimax

Někdy se může stát, že je výhodnější zahrát tah, který není v následujícím tahu výhodnější než jiný, ale v budoucnu bude lepší, k tomu nám dopomáhá algoritmus Minimax. Minimax je tedy algoritmus, který umožňuje počítači nahlédnout do příštích možných tahů a vybrat tah, který má nejvýhodnější možné další pokračování hry.

V Minimaxu je velmi efektní rekurze, tedy hlavně co se časové složitosti týče. V našem případě, tedy 3D piškvorkách, nám časová složitost n^2 (konstantní větvící faktor n, hloubka prohledávání 2) postačuje, u jiných her, například u šachu se používá vylepšení Minimaxu,

tzv. alfa-beta odřezávání, kde se rovnou při prohledávání zanedbávají = odříznou ty větve MiniMaxu (tj. možné pozice), které počítač určí jako ty, které nenabydou tak dobré pozice, jako některé, kterou už prohledal předtím. To sníží podstatně časovou složitost algoritmu, může se ovšem stát, že pokud půjdou výsledky od nejlepšího po nejhorší, nebude odříznuto nic. Musíme si ale uvědomit že i s dobrou situací, co se týče pořadí větví, které prohledáváme kvůli alfa-beta odřezávání, bude mít Minimax vždy vyšší časovou složitost (delší čas při počítání) než při použití Naive.



Obrázek 6: Grafické znázornění vyhledávacího stromu Minimax.

Část II
Praktická část

4 Pojmy

Pojem	Vysvětlení
Jazyk	Též programovací jazyk. Způsob zadávání příkazů počítači.
IDE	Též vývojové prostředí. Program, umožňující vytváření programu.
Proměnná	Místo v paměti, obsahuje něco – číslo, text, pravdivostní hodnotu...
Pole	Proměnné, seřazené za sebe, určují se názvem pole a pořadím (indexem).
Metoda	Několik příkazů, sjednocených pod jeden celek. Může vracet hodnotu.

Tabulka 2: Tabulka pojmů, uživaných v části II.

5 Úvodem

5.1 Výběr jazyka a vývojového prostředí

Pro tvorbu programu jsem si vybral jazyk *Java*, a to hned z několika důvodů:

- Java je velmi jednoduchá. V době, kdy jsem si vybíral, kterým jazykem program napíšu, jsem ovládal základy pythonu, který je javě v některých ohledech velmi podobný.
- Java je vhodná i pro úplné začátečníky. Chyby v programu se dají snadno vyhledat.
- Programy v javě jsou přenositelné, tj. lze je spustit na jakémkoli operačním systému. Javu lze spustit po úpravě i např. na mobilu nebo na tabletu.
- V porovnání s interpretovanými jazyky je Java velmi výkonná (php, ruby).
- Java je široce používaná, případě nesnází není problém najít někoho, kdo poradí.
- Java má velmi kvalitní vývojová prostředí.
- Po dotázání mi jej většina mých známých, kteří se v informatice a v programování orientují, doporučila. Můj interní konzultant také souhlasil.

Jako prostředí pro vývoj programu jsem si zvolil *Eclipse*, také k tomu bylo hned několik důvodů:

- Eclipse se dá rozšířit pomocí pluginů na znalost mnoha programovacích jazyků.
- Pluginy umí také rozšířit Eclipse například o grafické uživatelské rozhraní, ale i další.
- Eclipse je nejpoužívanější IDE pro vývoj Javy.

(a) Logo jazyku Java.

(b) Logo IDE Eclipse.

Obrázek 7: Využitý jazyk a IDE.

6 Algoritmus programu

Myšlenkou programu je ohodnocování pozic a následného výběru té, která má nejlepší hodnocení. Tabulka 3 obsahuje seznam metod programu.

Metoda	**Účel**
main()	Vyvolávání ostatních metod. Klíčový engine.
ohodnocovac()	Statické ohodnocení hracího pole.
strateg()	Algoritmus Naive.
strategminimax()	Engine metody *minimax()*.
minimax()	Algoritmus Minimax.
tahovac()	Vytvoření pole prázdných pozic.

Tabulka 3: Tabulka metod programu.

6.1 Ohodnocování pozice

Ačkoli se nejprve při spuštění programu spouští metoda *main()*, pro pochopení činnosti bude nejprve představena metoda *ohodnocovac()*, která činí statické ohodnocení pozice. Hodnotí se na základě počtu nezablokovaných čtveřic, trojic, dvojic a „jednic“ jedné a druhé strany. Jednice mají hodnotu 1, dvojice 10, trojice 100 a čtveřice 1 000 000, pro soupeře stejně, jen záporně. Toto hodnocení je umístěno v poli *hodnoty*, stejně tak je pro přehlednost v tabulce 4. Každé pole každé možné kombinace každého možného výherního směru se otestuje a v případě, že obsahuje značku, zvýší se o 1 buď neznámá *moje*, nebo *jeho*. Na konci každé čtveřice se neznámé zkontrolují. V případě, že jedna je nulová, pak se podle již zmíněného hodnocení pozmění celkové hodnocení pozice.

Myšlenka ohodnocování je velmi snadno realizovatelná, jak je vidět v listingu 1. Jediný zádrhel nastal v poli *smery*, které má velikost [72][4][3], pole obsahuje souřadnice všech čtyř políček kombinace všech 72 možných tahů. Pole jsem vytvořil v jiném programu. Kvůli tomu, že je definice pole dlouhá 1500 znaků, uvádím jen úplný počátek. Samotná metoda má jediný argument, a to intovou proměnnou *hrac*, která nabývá hodnot buď 1,

Počet symbolů	Hodnocení	
	Počítač	Člověk
0	0	0
1	1	- 1
2	10	- 10
3	100	- 100
4	1 000 000	- 1 000 000

Tabulka 4: Přehled hodnocení.

nebo -1. Díky tomu je možné ohodnotit pole jak z pohledu lidského hráče, tak z pohledu umělé inteligence počítače. To je velmi důležité pro metodu *minimax()*, ve které je tato metoda v obou možnostech proměnné *hrac* hojně volána.

Listing 1: Ohodnocovač.

```
final int[][][] smery = {{{0, 0, 0}, {0, 0, 1}, {0, 0, 2}, {0, 0, 3}}, {{0,
     1, 0}...;
int skore = 0;
for (int m = 0; m <= 71; m++) {
  int moje = 0;
  int souperovo = 0;
  for (int n = 0; n <= 3; n++) {
    if(pole[smery[m][n][0]][smery[m][n][1]][smery[m][n][2]] == -1) {
      moje++;
    }
    if(pole[smery[m][n][0]][smery[m][n][1]][smery[m][n][2]] == 1) {
      souperovo++;
    }
  }
  if(moje == 0) {
    skore -= hodnoty[souperovo];
  }
  if(souperovo == 0) {
    skore +=  hodnoty[moje];
  }
}
return skore;
```

6.2 Metoda *main()*

Metoda *main()* obsahuje kromě primitivního uživatelského rozhraní ve formě výpisu textu také jednu jednu z nejdůležitějších částí programu, while cyklus (můžete jej vidět v listingu 2), který volá všechny ostatní metody. Při vstupu do cyklu se nejprve spustí metoda *zadavac()*, do které hráč zadává svůj tah a ten je již v metodě zapsán do hrací desky. Následně se ohodnotí pole podle dříve popsané metody *ohodnocovac()* a zkontroluje se, zda hráč nevyhrál, v takovém případě klíčové slovo *break* osvobodí program z cyklu. Pokud ne, program pokračuje dál. Nyní program zkontroluje dříve uživatelem zadanou proměnnou *minimax* booleanového typu. Pokud je *true*, pak se spouští metoda *strategminimax()* s algoritmem Minimax. Pokud je hodnota proměnné *false*, pak se spouští metoda *strateg()* s algoritmem Naive. Obě metody vrací jednorozměrné pole se souřadnicemi dalšího tahu.

Metoda *main()* tedy pole vyplní značkou počítače, ohodnotí pozici a opět zkontroluje, zda právě proběhlý tah nebyl vítězný. Pokud ano, program se dostává z cyklu, pokud ne, pak se celý while cyklus zopakuje. Toto trvá, dokud není absolutní hodnota ohodnocení pole větší než 100 000, tedy dokud jeden hráč nevyhraje.

Listing 2: While cyklus v metodě main.

```
while (true) {
  zadavac();
  hodnoceni = ohodnocovac(-1);
  if (hodnoceni > 100000) {
    break;
  }
  if (minimax == true) {
    tah = strategminimax();
  }
  else {
    tah = strateg();
  }
  deska[tah[1]][tah[2]][tah[3]] = -1;
  hodnoceni = ohodnocovac(-1);
  if (hodnoceni < -100000) {
    break;
  }
}
```

6.3 Výběr pozice

Výběr pozice byl hlavní oříšek v mém programu, který jsem musel řešit. Je to v podstatě také hlavní problém u jakékoli hry, kdy má hráč více figurek, ale smí táhnout jen jednou, je více možností, kam položit kámen, ale hráč smí položit kámen jen na jedno místo, kdy může jet figurkou na různá místa, ale pouze na jedno v tahu, když si musí vybrat. V mé práci jsem umožnil hráči vybrat si, jestli má jeho protivník uvažovat na jedné úrovni tahů, nebo zda má použít algoritmus Minimax.

6.3.1 Metoda *strateg()* a algoritmus Naive

Jak již bylo zmíněno v sekci 3 na straně 12, Naive je algoritmus, který prohledá všechny možné pozice, které může z aktuální pozice počítač vykonat a vybere tu, která má nejlepší ohodnocení (o ohodnocení se můžete dočíst v předchozí kapitole). Metoda *strateg()* se vzájemně v programu vylučuje s metodou *strategminimax()*. Tyto metody jsou téměř stejné, ale účelem naprosto totožné. Algoritmus Naive je však oproti algoritmu Minimax méně propracovaný, jelikož se může stát, že přehlédne možnou vidličku (tedy že vzniknou dvě místa, kam může počítač v jednoum umístit značku a vyhraje).

Metoda *strateg()*, která obsahuje algoritmus Naive, nejprve dostane od metody *tahovac()* tabulku souřadnic polí, které nejsou v daném tahu obsazené. *strateg()* postupně zkouší do daných polí umístit svoji značku. Při každém umístění ohodnotí vzniklé pole. V případě, že má pole největší hodnocení, jaké v tomto tahu prozkoušel, zapíše údaje do tabulky o 4 políčkách, přepíše předchozí největší. První políčko (tedy *tah[0]*) je hodnocení,

zbylé tři souřadnice[7]. Právě toto první políčko je porovnáváno s aktuálním hodnocením. Na konci jsou tedy v tabulce souřadnice nejvýhodnějšího pole pro další tah. V metodě *main()*, kam se poté pole *tah* vrací již ale *tah[0]* s hodnocením neplní žádnou funkci.

Listing 3: Algoritmus Naive.

```
int[][] tahy = tahovac();
int[] tah = new int[4];
int hodnota;
tah[0] = -1000000;
for (int n = 0; n < pocitadlo; n++) {
  deska[tahy[n][0]][tahy[n][1]][tahy[n][2]] = -1;
  hodnota = ohodnocovac(1);
  if (hodnota > tah[0]) {
    tah[0] = hodnota;
    tah[1] = tahy[n][0];
    tah[2] = tahy[n][1];
    tah[3] = tahy[n][2];
  }
  deska[tahy[n][0]][tahy[n][1]][tahy[n][2]] = 0;
}
return tah;
```

6.3.2 Metody *strategminimax()* a *minimax()* a algoritmus Minimax

Listing 4: Algoritmus Minimax (metoda Strategminimax).

```
public static int[] strategminimax() {
  int[][] tahy = tahovac();
  int[] tah = new int[4];
  int hodnota;
  tah[0] = -1000000;
  for (int n = 0; n < pocitadlo; n++) {
    deska[tahy[n][0]][tahy[n][1]][tahy[n][2]] = -1;
    hodnota = minimax(0, 1);
    if (hodnota > tah[0]) {
      tah[0] = hodnota;
      tah[1] = tahy[n][0];
      tah[2] = tahy[n][1];
      tah[3] = tahy[n][2];
    }
    deska[tahy[n][0]][tahy[n][1]][tahy[n][2]] = 0;
  }
  return tah;
}
```

Podstatným vylepšením programu bylo nahrazení Naive algoritmem Minimax. Minimax je obecně algoritmus, který z možných kauzalit vybíra tu s nejslibnějším pokračováním. V mém programu pak porovnává tahy, které mohou být vykonány z aktuální pozice, a následně vybere ten, který je nejvýhodnější, tahy porovnává na základě ohodnocení pomocí metody *ohodnocovac()*.

Algoritmus jsem rozdělil do dvou metod, *strategminimax()* a *minimax()*. První metoda, *strategminimax()*, doplní vždy do desky tah počítače a vrací program do metody *main()*.

[7] *Tah[1]* je souřednice X (čtverec), *tah[2]* je Y (řádek) a *tah[3]* je Z (sloupec).

Druhá, *minimax()*, následně pokračuje ve větvení za pomocí rekurze až do stanovené maximální hloubky.

Nyní již k bližšímu popisu metod. Princip metody *strategminimax()* je stejný jako metody *strateg()*, stejně jako kód, kde jediná změna nastává při načítání proměnné *hodnota*. Místo přímého načtení z metody *ohodnocovac()* se její hodnota načítá nepřímo z metody *minimax()*, v rámci které ohodnocení ale také probíhá pomocí metody *ohodnocovac()*, jak bude zřejmé později.

Listing 5: Algoritmus Minimax (metoda Minimax)

```
public static int minimax (int hloubka, int hrac) {
  int maxtah = -100000;
  if (hloubka == maxhloubka) {
    return -ohodnocovac(hrac);
  }
  else {
    int[][] tahy = tahovac();
    for (int n = 0; n < pocitadlo; n++) {
      deska[tahy[n][0]][tahy[n][1]][tahy[n][2]] = hrac;
      if(Math.abs(ohodnocovac(hrac)) > 20000) {
        deska[tahy[n][0]][tahy[n][1]][tahy[n][2]] = 0;
        return -ohodnocovac(hrac);
      }
      else {
        int ohodnoceni = -minimax(hloubka+1, -hrac);
        if (ohodnoceni > maxtah) {
          maxtah = ohodnoceni;
        }
      }
      deska[tahy[n][0]][tahy[n][1]][tahy[n][2]] = 0;
    }
  }
  return maxtah;
}
```

Nyní již ke klíčové metodě *minimax()*. Její argumenty jsou aktuální hloubka a hráč, který je na tahu. Nejprve algoritmus zkontroluje, zda již není v koncovém uzlu, tedy, že hloubka není již maximální hloubka, pak vrací ohodnocení situace aktuálního pole. Pokud není, pak načte všechny možné tahy pomocí metody *tahovac()* a do prvního pole položí značku aktuálně hrajícího hráče a ohodnotí pole. Pokud aktuálně hrající hráč vítězí, pak se vrací. Pokud ne, dále se větví. Nakonec se doplněné políčko vyprázdní a zkouší další možné. Takhle to pokračuje, dokud hodnota proměnné n, jakožto klíčové proměnné cyklu nedosáhne k hodnoty statické proměnné pocitadlo, která se pozmění vždy při vyvolání metody *tahovac()*. Více v listingu 5.

6.4 Metoda *tahovac()*

Aby mohl být výběr pole proveden, ať už Naive, nebo Minimax, je třeba získat seznam všech prázdných polí, tedy polí, kam může být příští tah soustředěn, aby program věděl, která pole má uvažovat. Možná jste si všimli, ať už v listingu 3, 4 nebo 5, že je v kódu zmíněna zatím nepopsaná metoda *tahovac()*. Ta právě pole souřadnic prázdných pozic vrací.

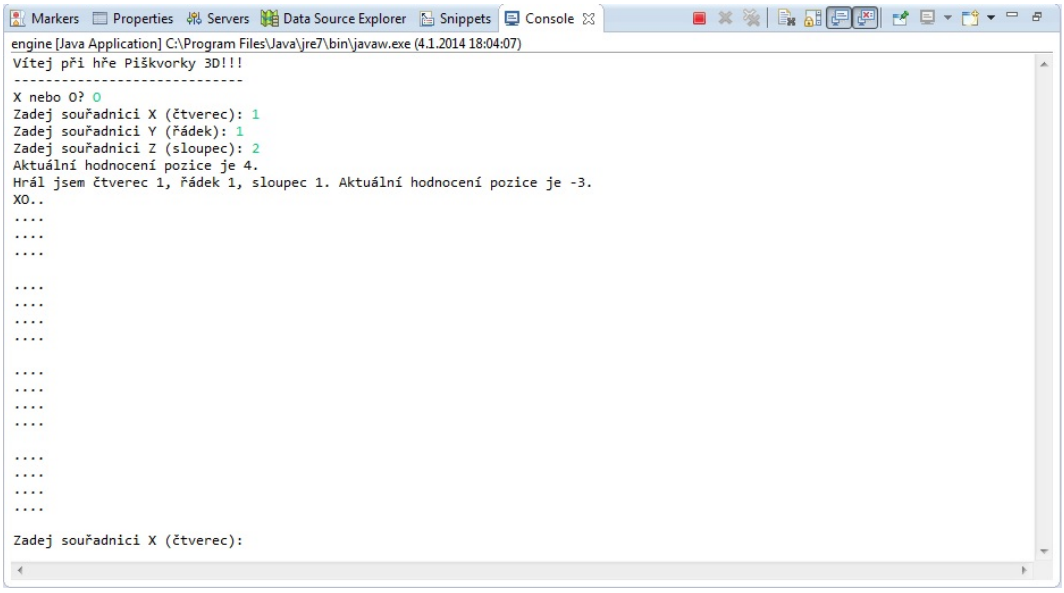
Metoda prohlédá každé políčko hracího pole a v případě, že je splněna podmínka, že je políčko prázdné, zapíše si jeho souřadnice do pole *tahy*. Toto pole má rozměry [64][3], tedy 3 souřadnice (čtverec, řádek, sloupec) 64 políček (celkový počet políček). Nakonec je pole tahy metodou vraceno. Také zde figuruje statická proměnná *pocitadlo* intového typu, která je využívána poté při vybírání pole, ať už přes Naive, nebo Minimax, kde ohraničuje for cyklus, který tam prochází pole tahy. Kód si můžete prohlédnout v listingu 6.

Listing 6: Tahovač.

```
int[][] tahy = new int[64][3];
    pocitadlo = 0;
    for (int a = 0; a <= 3; a++) {
      for (int b = 0; b <= 3; b++) {
        for (int c = 0; c <= 3; c++) {
          if (deska[a][b][c] == 0) {
            tahy[pocitadlo][0] = a;
            tahy[pocitadlo][1] = b;
            tahy[pocitadlo][2] = c;
            pocitadlo++;
          }
        }
      }
    }
    return tahy;
```

7 Vzhled

Prvotní a zatím prozatimní vzhled je v textu, pozice se zadává čísly, které zastupují souřadnice. Počítá se shora a zleva, jak je vidět na obrázku **??**, vzhled programu můžete vidět na obrázku 8.



Obrázek 8: Vzhled verze 1.0.

Diskuze

V budoucnu hodlám rozšířit tento program o některé další prvky. Předně bude dokončeno grafické prostředí pro hru. Předpokládám, že jej budu tvořit v Java GUI. Dále bude program obsahovat i položky jako alfa-beta odřezávání. Samotný program není až zas tak pomalý, ale přece jenom pořád není dost rychlý na to, aby se mohl měřit s výpočetní rychlostí jiných her tohoto typu. Dalšími nápady, které v budoucnu zahrnu do programu budou i načítání uložené pozice ze souboru *.txt* a vytvoření mobilní verze programu (pro Android a iOS). Dalším možným rozvojem je pak hra pro 2 hráče a uvažuji i o hru pro dva hráče přes LAN.

Zmíněné grafické prostředí by mohlo kromě poutavého vzhledu menu obsahovat samozřejmě vypsanou mřížku, kde by se tah zaváděl kliknutím myši, aktuální zadávání souřadnic je velmi nepohodlné. Ač začátečník, nebude problém vytvořit dobré grafické prostředí, jelikož se nejedná o 3D grafiku, ale o menu a v předchozí větě zmíněný způsob zadávání tahu.

Jak bylo řečeno v úvodu této práce, existují různé verze 3D piškvorek, zejména ve formě flashových her. Při srovnání jsou samozřejmě hry s grafickým rozhraním jednodušší na hraní, což je ale způsobeno absencí grafického rozhraní v mém programu. Po stránce obtížnosti však málokterá nabízí tak velkou obtížnost jako právě můj program. Z toho plyne, že algoritmy použité zde jsou velmi efektivní.

Můj program je tedy kvalitním soupeřem při hře 3D piškvorky. Proto kódy klíčových částí kódu (které jsou popsány i zde) zveřejním velmi brzy na *programujte.com*, současně i s mojí prací. Můj program po dokončení grafického prostředí (což udělám, jen co se naučím Java GUI, nebo uvažuji ještě o openGL) pak na některém herním serveru jako flashovou hru.

Závěr

Podařilo se mi vytvořit program, který je schopným digitálním soupeřem při hře 3D piškvorky. Obsahuje primitivní grafické rozhraní a umělou inteligenci. Splnil jsem cíle, které jsem si vytýčil o tom, co bude odevzdávaná verze programu mít.

K psaní práce jsem použil LaTeX který jsem se naučil vcelku slušně ovládat hlavně díky této práci, k vytvoření titulní strany pak Microsoft Office Word a k vytvoření prezentace Microsoft Office Powerpoint. Díky této práci a hlavně programu navíc ovládám dobře jazyk Java, což se na trhu práce velmi hodí.

Při psaní programu jsem se snažil používat pouze standardy jazyka a co nejvíce kód zjednoduššit. Práce mě také ještě více podnítila ke studiu informatiky. Ve vylepšování programu budu i nadále pokračovat, jelikož mne tématika této hry zaujala a navíc jsem pochopil, jaký potenciál 3D piškvorky mají.

Reference

[1] Hry: Trojrozměrné piškvorky. *Sbírka Mozkovna: Hrátky s logikou a matematikou.* 2004, č. 23, 15 - 16 s.

[2] RYBKA, Aleš. Zajímavosti ze světa piškvorek a renju: Historie gomoku a renju. ČESKÁ FEDERACE PIŠKVOREK A RENJU. *Piškvorky.cz.* Vyd. 2013, cit. 2013-11-25. Dostupné z: `http://www.piskvorky.cz/`.

[3] HANTL, Michal. Nápověda a pravidla: Pravidla piškvorek. VÝVOJOVÉ CENTRUM PIŠKVOREK A RENJU. *VCPR.cz.* Vyd. 2013, cit. 2013-11-25. Dostupné z: `http://www.vcpr.cz/`.

[4] BAŤHA, Matěj, Dan ZEMAN a Jakub TĚŠÍNSKÝ. Renju. *DeskovéHry.cz.* Vyd. 8-9-2009, cit. 2013-11-25. Dostupné z: `http://www.deskovehry.cz/`.

[5] Hry s tužkou a papírem: Piškvorky. *Wikipedia.cz.* Vyd. 2010-9-30, cit. 2013-11-26. Dostupné z: `http://cs.wikipedia.cz/`.

[6] Topics: Tic tac toe. *Cracked.com.* Vyd. 2013, cit. 2013-11-26. Dostupné z: `http://www.cracked.com/`.

[7] KOPŘIVA, Petr. O soutěži: O projektu. STUDENTS CYBER GAMES. *pIšQworky.cz.* Vyd. 2013, cit. 2013-11-26. Dostupné z: `http//www.pisqworky.cz/`.

[8] Historie teorie her. Vysoká Škola Finanční a Správní, o. p. s. Vyd. 2012, cit. 2013-12-10.

[9] ZAPLETAL, Lukáš. Software: Eclipse 2 - IDE na všechno. *root.cz.* Vyd. 2002-9-3, cit. 2013-12-28. Dostupné z: `http://www.root.cz/`.

[10] STEINWENDER, Dieter, Frederic A. FRIEDEL. Šachy na PC. Vyd. 1997, cit. 2014-1-1. 2013-12-28. 40-65 s.

[11] BENEDA, David. Počítačová hra 3D piškvorky v OpenGl. Zlín, 2010. bakalářská práce (Bc.). Univerzita Tomáše Bati ve Zlíně. Fakulta aplikované informatiky. Vyd. 2010-3-5, cit. 2014-1-12

[12] KOUBA, Petr. Umělá inteligence pro hru Pentago. Pardubice, 2010. Středoškolská odborná činnost. Gymnázium, Pardubice, Dašická 1083. Vyd. 2010, cit. 2014-1-12.

Seznam obrázků

Seznam tabulek

Seznam listingů

Seznam příloh