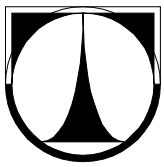
TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky a mezioborových inženýrských studií



BAKALÁŘSKÁ PRÁCE

Liberec 2007 **Tomáš Hlava**

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky a mezioborových inženýrských studií

Studijní program: B2612 – Elektrotechnika a informatika
Studijní obor: 2612R011– Elektronické informační a řídicí systémy

Implementace algoritmů hry dáma pro roboty ABB

Implementation algorithm game checker for robot ABB

Bakalářská práce

Autor: **Tomáš Hlava**
Vedoucí práce: Ing. Tomáš Pluhař
Konzultant: Ing. Tomáš Martinec

V Liberci 18. 5. 2007

TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií

Katedra softwarového inženýrství Akademický rok: **2006/2007**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Jméno a příjmení: **Tomáš Hlava**

studijní program: B 2612 – Elektrotechnika a informatika

obor: 2612R011 - Elektronické informační a řídicí systémy

Vedoucí katedry Vám ve smyslu zákona o vysokých školách č.111/1998 Sb. určuje tuto bakalářskou práci:

Název tématu: **Implementace algoritmů hry dáma pro roboty ABB**

Zásady pro vypracování:

1. Seznamte se s možnostmi řízení robotů ABB po sériové lince
2. Seznamte se se základními metodami rozpoznávání obrazu
3. Naprogramujte algoritmus hry „dáma“
4. Tento algoritmus implementujte robotům ABB a umožněte tak hru „robot-robot“ a „člověk-robot“
5. Vytvořte dokumentaci vámi vytvořených algoritmů a použitých postupů

Prohlášení

Byl(a) jsem seznámen(a) s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé diplomové práce a prohlašuji, že **s o u h l a s í m** s případným užitím mé diplomové práce (prodej, zapůjčení apod.).

Jsem si vědom(a) toho, že užít své diplomové práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Diplomovou práci jsem vypracoval(a) samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce a konzultantem.

Datum 18.5.2007

Podpis

Poděkování

Chtěl bych vyjádřit své poděkování vedoucímu mé bakalářské práce, Ing. Tomáši Pluhařovi, za pomoc a odborné vedení.

Dále bych chtěl poděkovat vedoucí katedry softwarového inženýrství, RNDr. Kláře Císařové, za ochotu a podporu při mé práci.

Abstrakt

Cílem této práce je vytvoření algoritmu pro hru dáma a jeho naprogramování v jazyce Delphi tak, aby si tuto deskovou hru mohli zahrát proti sobě dva roboti nebo člověk s robotem. To vše za předpokladu, že člověk bude roboty ovládat od počítače pomocí grafického rozhraní programu. K tomu je zapotřebí komunikace s robotem po sériové lince a rozpoznávání obrazu z kamery snímající scénu. Program přijímá data od kamery umístěné nad hrací plochou, která snímá rozložení kamenů na desce (stav hry). Tato data se zpracují, vyhodnotí a následně robot přijímá příkazy po sériové lince, jak má hrát.

Klíčová slova

Dáma

Algoritmus

Kamera

Robot ABB

Abstract

The aim of this work is to develop algorithms for the game Checker and its programming into the language Delphi to be able to play the table game Checker by two robots against each other or a human with a robot. The person controls it from a computer thanks to graphical interface of program. To that is needed communication with the robot on serial line and image recognition of camera. The program receives data from the camera that is placed over the game area. Those data are judged and then the robot receives order trough the serial line how it should play.

Keywords slova

Draughts

Checkers

Algorithm

Camera

Robot ABB

Obsah

Úvod

Deskovou hru dáma a její pravidla znají asi všichni. Pro jistotu je uvádím v části 3.1. Dámu mají hrát roboti ABB se šesti stupni volnosti. Pro komunikaci s počítačem využívá robot sériové linky a může mít jakýkoliv nástroj, který mu připevníme. Pro nás bude nejlepší přísavka, díky které může robot uchopit pingpongový míček. Ten bude reprezentovat daný kámen. Míčky budou umístěny na desce se 64 otvory, vyrobené z plexiskla.

Algoritmus pro hraní dámy může být úplně jednoduchý. Snadno vybereme figurku, která se může hnout, a táhneme s ní. Pakliže musí skočit, tak skočí. Tento algoritmus ovšem nereaguje na tahy soupeře a „nepřemýšlí“ dopředu. Rozhodně by s tímto počítač na žádného člověka nevyzrál. V praxi se používají různé modifikace metody minimaxu. Tento algoritmus vypočítá všechny možné tahy a ohodnotí jejich přínos pro hráče. Poté zjistí všechny možné tahy soupeře a opět je ohodnotí. Toto provádí do hloubky, kterou si zvolíme. Tato metoda je základem většiny deskových her hraných na počítači. Když budeme mít velmi výkonný počítač, který nám umožní prohledávání do co největší hloubky, je úspěch zaručen. V dnešních podmínkách by výsledný program této práce mohl pracovat s hloubkou 6.

Kamera, umístěná nad scénou, může být jakákoliv barevná kamera připojená k počítači. Důležitý je výběr barev, se kterými budeme pracovat. Ideální jsou barvy lehce rozpoznatelné kamerou. Problém by mohl nastat se změnou osvětlení hrací plochy během hry. Dále je nutné ošetřit případné pohnutí (vyosení) kamery během hry.

1. Komunikace s robotem

Komunikace s robotem probíhá po sériové lince. Do robota nahrajeme program (robot.prg), který v nekonečné smyčce čeká na naše příkazy. Musíme znát port, po kterém můžeme s robotem komunikovat. Tento port otevřeme a posíláme příkazy.

1.1 Komunikace s robotem v prostředí Delphi 7.0

Pro komunikaci s robotem byla vytvořena jednotka „robot.dcu“, kterou je nutné zmínit v úvodní části vytvářeného programu, v oddílu USES.

Část kódu může vypadat např. takto:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
Controls, Forms,
  Dialogs, StdCtrls,robot;

type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
```

Práce s touto jednotkou je naprosto stejná jako práce s každou jinou jednotkou.

Tato jednotka obsahuje několik konstant a typů, které jsou dále využívány v jediném objektu. Následuje výpis konstant a typů:

```
const
  v10=0;
  v20=1;
  v30=2;
  v40=3;
  v50=4;
  v60=5;
  v80=6;
  v100=7;
  v200=8;
  v300=9;
  v400=10;
```

```
  v500=11;
  v600=12;
  v800=13;
  v1000=14;
  v1500=15;
  v2000=16;
  v3000=17;

  zfine=0;
  z0=1;
  z1=2;
  z5=3;
  z10=4;
  z15=5;
  z20=6;
  z30=7;
  z40=8;
  z50=9;
  z60=10;
  z80=11;
  z100=12;
  z150=13;
  z200=14;

type tpos=record x,y,z:single; end;
type trobtarget=record x,y,z,a,b,c:single; end;
type tjointtarget=record a,b,c,d,e,f:single; end;
```

Konstanty „vxx“ mají stejnou funkci jako stejnojmenné konstanty na „straně robota“ tzn. představují konstanty rychlosti. Číslo v názvu znamená rychlost pohybu v [mm/s].

Konstanty „zxx“ mají stejnou funkci jako stejnojmenné konstanty na „straně robota“ tzn. představují konstanty zón. Číslo v názvu znamená velikost zóny v [mm].

Problematika nastavování rychlostí a zón je poněkud rozsáhlejší, ale pro naše potřeby plně postačí toto vysvětlení.

Vytvořené typy také kopírují existující typy, které jsou používány robotem. Tzn. po řadě

tpos – určuje polohu bodu v prostoru v kartézských souřadnicích (využívá se k navádění robota)

trobtarget – obsahuje také souřadnice bodu v prostoru (podobně jako tpos), navíc ale obsahuje ještě složky a,b,c. Tyto složky jsou Eulerovy úhly a

umožňují nastavit orientaci v prostoru. Těchto šest proměnných (vlastně jedna proměnnou typu `trobtarget`) reprezentuje polohu robota v prostoru (jak souřadnice příruby tak úhel jejího natočení)

tjointtarget – obsahuje šest složek typu **single**, které představují natočení jednotlivých ramen robota

A konečně objekt s názvem „**trobot**“.

Konstruktor robota obsahuje jediný parametr a tím je název portu ke kterému je připojen tzn. např. „COM2“.

Dále obsahuje několik funkcí, které jsou svými názvy totožné s funkcemi, které má sám robot. Jsou to: movel, movej, moveabs,waittime,setdo. Některé z funkcí jsou z důvodu širších možností přetíženy. Výpis ze zdrojového kódu vypadá následovně:

```
type trobot=class
     constructor create(s:string);
     procedure movel(rob:trobtarget);overload;
     procedure movel(x,y,z,a,b,c:single);overload;
     procedure movel(p:tpos);overload;
     procedure movel(x,y,z:single);overload;
     procedure movej(rob:trobtarget);overload;
     procedure movej(x,y,z,a,b,c:single);overload;
     procedure movej(p:tpos);overload;
     procedure movej(x,y,z:single);overload;
     procedure moveabs(joint:tjointtarget);overload;
     procedure moveabs(a,b,c,d,e,f:single);overload;
     procedure setdo(stav:byte);
     procedure waittime(t:byte);

     procedure speed(s:byte);
     procedure zone(z:byte);
      end;
```

Protože rychlosti a zóny jsou v robotovi parametry jednotlivých příkazů a příkazy samotné, bylo nutné vytvořit další dvě funkce tj. funkce:

```
     procedure speed(s:byte);
     procedure zone(z:byte);
```

Jako parametr se jim předává některá z konstant „vxx“ nebo „zxx“ (popsáno již dříve).

Následuje výpis z programu, který demonstruje použití popsaných struktur a objektů:

```
unit Unit1;
interface
uses …..,robot;
type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
……
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
…….
  private
  public
    r:trobot;
  end;

var
  Form1: TForm1;
implementation

//otevře port COM1 a připraví komunikaci
procedure TForm1.FormCreate(Sender: TObject);
begin
r:=trobot.create('COM1');
end;

// zavře port
procedure TForm1.FormDestroy(Sender: TObject);
begin
r.destroy;
end;

// přesune robota na dané souřadnice
procedure TForm1.Button1Click(Sender: TObject);
begin
r.movej(400,0,20,-180,0,180);
end;

// přesune robota na dané souřadnice
procedure TForm1.Button2Click(Sender: TObject);
begin
```

```
r.movel(400,-300,20,-180,0,180);
end;

// přesune robota na dané souřadnice
procedure TForm1.Button4Click(Sender: TObject);
begin
r.moveabs(0,0,0,0,0,0);
end;

// zapne přísavku
procedure TForm1.Button5Click(Sender: TObject);
begin
r.setdo(1);
end;

// nastaví rychlost na 1000mm/s
procedure TForm1.Button7Click(Sender: TObject);
begin
r.speed(v1000);
end;

// nastaví zónu na 150mm
procedure TForm1.Button9Click(Sender: TObject);
begin
r.zone(z150);
end;

// vykoná složitější pohyb
procedure TForm1.Button13Click(Sender: TObject);
var x:integer;
begin
for x:=0 to 30 do
 begin
 r.movel(400,-300+x*10,20,-180,0,180);
 r.movel(400,-300+x*10,40,-180,0,180);
 r.movel(400,-300+(x+1)*10,40,-180,0,180);
 end;
end;
end.
```

K tomu, aby tato jednotka společně s výše uvedenými metodami fungovala, je nutné na straně robota spustit příslušný program, který v nekonečné smyčce čeká na příkazy. Tento program se nazývá „ROBOT.PRG“.

2. Použití kamery

Při použití kamery musíme mít na počítači nainstalované ovladače naší kamery. Dále musíme mít nainstalovanou knihovnu DirectX pro pracování s multimédii.

2.1 Využívání kamer v prostředí Delphi 7.0

Pro využívání kamer v prostředí Delphi 7.0 byla napsána knihovna s názvem „kamera“, tuto knihovnu je nutné (pokud ji chceme využívat) připsat do části USES, která je uvedena v téměř každém programu, který je v tomto prostředí vytvořen. Část kódu vypadá např. takto:

```
uses
  …,Dialogs,kamera, ExtCtrls, …;
```

Tato jednotka obsahuje definici tří typů, jedné funkce a dvou objektů resp. objektových tříd.

typy

```
type tbod=record b,g,r:byte; end;
            pbod=^tbod;

        type tprahujproc=procedure(var b:tbod);
```

typ **tbod** reprezentuje barevný bod, který je složen ze tří složek r,g,b (red, green, blue – červená, zelená, modrá)

typ **pbod** reprezentuje ukazatel na typ tbod

typ **tprahujproc** reprezentuje proceduru kterou je možno využít k prahování nebo jakékoli úpravě obrazu (blíže vysvětlíme později)

funkce

function bod(r,g,b:**byte**):tbod;

tato funkce umožňuje konvertovat tři hodnoty typu byte na jednu hodnotu typu tbod.

Objekty

type tmask=**class**

tento objekt reprezentuje masku, kterou je možné použít např. při „efektu“ dilatace, erose nebo jiné obecnější funkce.

type tkamera=**class**

tento objekt je stěžejní, zabírá 90% kódu a veškerá funkcionalita je ukryta právě zde.

Nyní se podrobněji podíváme na každou objektovou třídu zvlášť. Je nutné zdůraznit, že výpisy zdrojových kódů nejsou úplné a to z důvodu přehlednosti celého dokumentu. Jsou však uvedeny všechny podstatné proměnné, funkce i procedury.

Objektová třída „tmask“

Def.:

```
type tmask=class
      constructor create(wi,he,ox,oy:integer)
      public
      data:array of boolean;
      end;
```

Objektová třída obsahuje konstruktor, jehož parametry jsou pořadě:

- šířka masky
- výška masky
- x-ová souřadnice středu masky
- y-ová souřadnice středu masky

Tato objektová třída dále obsahuje datovou proměnnou data, která je jednorozměrným polem a po řádcích obsahuje hodnoty typu BOOLEAN, které reprezentují „tvar“ masky. Dodatečnou změnou této proměnné je možné vytvořit z původní obdélníkové masky masku jakéhokoli tvaru. Hodnota TRUE znamená, že příslušný bod v masce se bere v úvahu a hodnota typu FALSE znamená, že příslušný bod se ignoruje.

objektová třída Kamera

Def.:

```
type tkamera=class
     constructor create;
     function readBMP(Bitmap: TBitmap):boolean;
     procedure SetFormat(Width,Height:integer);
```

```
    procedure prahuj(p:pointer);overload;
    procedure prahuj(r,g,b,o:integer);overload;
    procedure erose(m:tmask);
    procedure dilatace(m:tmask);
    procedure eroseEx(m:tmask);
    procedure dilataceEx(m:tmask);
    procedure inverze;
    procedure operace(m,m2:tmask;bol:boolean);
    procedure setregion(x1,y1,x2,y2:integer);
    function getpixel(x,y:integer):tbod;
    function getpixelv(x,y:integer):pbod;
    procedure setpixel(x,y:integer);overload;
    procedure setpixel(x,y:integer;c:tbod);overload;
    function inside(x,y:integer):boolean;
    function procent(c:tbod):integer;
    function isequal(a,b:pbod):boolean;
    function  getcolor:tbod;
    procedure setcolor(c:tbod);
    procedure horizontalline(x1,y1,x2:integer);
    procedure verticalline(x1,y1,y2:integer);
    procedure rectangle(x1,y1,x2,y2:integer);
    procedure copybuffer(d,s:pbod);
    procedure newbuffer(var d:pbod);
    procedure deletebuffer(d:pbod);
    procedure setbuffer(b:pbod);
    procedure union(d,s1,s2:pbod);
    procedure intersection(d,s1,s2:pbod);
  public
    ListDevice,ListFormat:tstrings;
    end;
```

A nyní si popíšeme jednotlivé procedury a funkce podrobněji.

constructor create;

Konstruktor vyhledá v systému nainstalované kamery a pokud nějaká existuje, tak ji začne využívat. Toto vyjádření je zavádějící, ale podrobnější popis je zbytečný. Je možné odkázat na dokumentaci DirectX konkrétně DirectShow. Pokud nedojde k nalezení kamery, tak není z pochopitelných důvodů možné využívat ostatní funkce.

function readBMP(Bitmap: TBitmap):boolean;

Tato funkce načte z kamery (konkrétně ze streamu vytvořeného konstruktorem) obrázek, který následně vloží do bitmapy, která je jejím jediným parametrem.

procedure SetFormat(Width,Height:**integer**);

Tato procedura nastaví rozlišení kamery na požadované parametry (pokud kamera toto rozlišení podporuje).

procedure prahuj(p:pointer);**overload**;

Tato procedura umožňuje prahovat - případně téměř jakkoli upravovat obraz. Jako parametr se jí předává proměnná typu TPrahujProc. Viz. výše.

procedure prahuj(r,g,b,o:**integer**);**overload**;

Tato procedura má již nastavený určitý konkrétní typ prahování. Parametry r,g,b reprezentují barvu, která se bude s odchylkou o prahovat. Pokud je odchylka menší, bude výsledná barva bílá, pokud ne, bude černá.

procedure erose(m:tmask);

Tato procedura dělá erozi (tato funkce je známá z jiných předmětů). Má jediný parametr. Tento parametr reprezentuje masku, která se při erozi využije. Popsáno výše.

procedure dilatace(m:tmask);

Tato procedura dělá dilataci (tato funkce je známá z jiných předmětů). Má jediný parametr. Tento parametr reprezentuje masku, která se při dilataci využije. Popsáno výše.

procedure inverze;

Tato procedura vytváří inverzní obrázek.

procedure setregion(x1,y1,x2,y2:**integer**);

Tato procedura umožňuje nastavení obdélníkové oblasti, na kterou se budou aplikovat VEŠKERÉ operace, které je možné nějakým způsobem spojit s konkrétní oblastí. Tzn. procedury prahuj, erose, dilatace,inverze, procent atd.

function getpixel(x,y:**integer**):tbod;

Tato funkce načte bod, který svým jménem vrátí.

function getpixelv(x,y:**integer**):pbod;

Tato funkce načte ukazatel na bod, který svým jménem vrátí.

procedure setpixel(x,y:**integer**);**overload**;

Tato procedura umožňuje nastavit určitý bod na obrazovce na konkrétní barvu.

function procent(c:tbod):**integer**;

Tato funkce vrátí obsazení bílou barvou v zadaném regionu v procentech. Této funkce je možné využít například po vhodném vyprahování.

function getcolor:tbod;

Tato funkce vrátí aktuální barvu, která bude použitá k vykreslování.

procedure setcolor(c:tbod);

Tato procedura nastaví barvu ke kreslení.

procedure horizontalline(x1,y1,x2:**integer**);

Tato procedura nakreslí horizontální čáru, parametry x1,x2 určují začátek a konec čáry a parametr y1 určuje y-ovou souřadnici čáry.

procedure verticalline(x1,y1,y2:**integer**);

Tato procedura nakreslí vertikální čáru, parametry y1,y2 určují začátek a konec čáry a parametr x1 určuje x-ovou souřadnici čáry.

procedure rectangle(x1,y1,x2,y2:**integer**);

Tato procedura nakreslí obdélník, přičemž první dva parametry určují levý horní a druhé dva parametry pravý dolní roh vykreslovaného obdélníku.

procedure newbuffer(**var** d:pbod);

Tato procedura vytvoří nový buffer, který je možné použít jako jakousi zálohu upraveného obrázku.

procedure deletebuffer(d:pbod);

Tato procedura smaže existující buffer.

procedure copybuffer(d,s:pbod);

Tato procedura umožňuje kopírovat buffery „bod po bodu“. První parametr je cílový a druhý zdrojový.

procedure setbuffer(b:pbod);

Tato procedura nastaví aktuální buffer na buffer předaný v parametru. Všechny operace, které pracují s obrazem budou od této chvíle pracovat s obrazem, který se touto procedurou nastaví jako aktuální.

procedure union(d,s1,s2:pbod);

Tato procedura vytváří sjednocení (logické plus).

procedure intersection(d,s1,s2:pbod);

Tato procedura vytváří průnik (logické krát)

public

ListDevice,ListFormat:tstrings;

ListDevice obsahuje seznam nainstalovaných kamer a ListFormat obsahuje seznam možných rozlišení, které konkrétní kamera podporuje.

Následuje zkrácený výpis z konkrétního programu, který tuto jednotku využívá.

```
unit Unit1;
interface
uses
…,kamera, …;
type
TForm1 = class(TForm)
Image1: TImage;
Timer1: TTimer;
procedure FormCreate(Sender: TObject);
procedure Timer1Timer(Sender: TObject);
k:tkamera;        // proměnná kamera
{ Public declarations }
end;
var
Form1: TForm1;
implementation

procedure TForm1.FormCreate(Sender: TObject);
begin
k:=tkamera.create;                // „vytvoří“-lépe řečeno zpřístupní kameru
k.setregion(50,50,200,200);     // nastaví výřez v obrazu se kterým se bude //pracovat
end;
procedure TForm1.Timer1Timer(Sender: TObject);
begin
k.readBMP(Image1.Picture.Bitmap);    //  načte  bitmapu  do  komponenty Image1
k.setbuffer(k.buffer);
k.prahuj(255,0,0,50);         //Provede prahování červené
k.inverze;                    //vytvoří inverzní obrázek
form1.Caption:=inttostr(k.procent(bod(255,255,255)));    //vypíše    do titulkového pruhu okna procentuelní zastoupení bílé barvy
end;

//tato procedura nastaví rozlišení 640x480 (pokud ho vybrané zařízení //podporuje)
procedure TForm1.Button1Click(Sender: TObject);
begin
k.SetFormat(640,480);
end;
//tato procedura nastaví rozlišení 320x240 (pokud ho vybrané zařízení //podporuje)
procedure TForm1.Button2Click(Sender: TObject);
begin
k.SetFormat(320,240);
end;
end.
```

2.2 DirectX

DirectX je programátorská knihovna obsahující nástroje pro tvorbu počítačových her a dalších multimediálních aplikací, vytvořená firmou Microsoft pro použití pod operačním systémem Windows. Aktuální verzí knihovny je momentálně (2007) DirectX 9.

DirectX se skládá z několika částí, rozdělených podle svého účelu:

- **DirectX Graphics** (a jeho části **DirectDraw** a **Direct3D**) – podpora grafiky, 3D vykreslování atd.,
- **DirectInput** – podpora vstupních zařízení jako např. myši, joysticky, gamepady apod., včetně podpory technologie force feedback,
- **DirectPlay** – podpora hry více hráčů po síti,
- **DirectSound** (dříve též spolu s DirectMusic označováno souhrnným názvem **DirectX Audio**) – podpora přehrávání a záznamu zvuků,
- **DirectMusic** – podpora přehrávání a zpracování hudby,
- **DirectShow** – podpora multimediálních aplikací, přehrávání a zpracování videa a zvuku,
- **DirectSetup** – jednoduchý nástroj umožňující instalaci knihovny DirectX na počítač,
- **DirectX Media Objects** – podpora pro tvorbu multimediálních efektů, kodeků apod.

Toto rozdělení je podstatné hlavně pro programátora, DirectX se šíří jako jediný balík obsahující všechny komponenty.

historie

DirectX verze	Číslo verze	Operační systém	Datum vypuštění
DirectX 1.0	4.02.0095		1995
DirectX 2.0 / 2.0a	4.03.00.1096	Windows 95 OSR2 a Windows NT 4.0	1996
DirectX 3.0 / 3.0a	4.04.0068 / 69	Windows NT 4.0 SP3 *Poslední podporovaná verze DirectX pro Windows NT 4.0*	1996

DirectX 4.0	Nikdy nebylo vypuštěno		
DirectX 5.0	4.05.00.0155 (RC55)	*Dostupné jako součást beta verze Windows NT 5.0*	1997
DirectX 5.0	4.05.01.1721 / 1998	Windows 98	1998
DirectX 6.0	4.06.00.0318 (RC3)	Windows 98 SE	1998
DirectX 6.1	4.06.02.0436 (RC0)		1999
DirectX 7.0	4.07.00.0700 (RC1)	Windows 2000 a Windows ME	1999
DirectX 7.0a	4.07.00.0716 (RC1)		1999
DirectX 8.0	4.08.00.0400 (RC14)	Xbox *Poslední podporovaná verze DirectX pro Windows 95*	2000
DirectX 8.1	4.08.01.0810 4.08.01.0881 (RC7)	Windows XP	2001
DirectX 9.0	4.09.0000.0900	Windows Server 2003	2002
DirectX 9.0a	4.09.0000.0901		2003
DirectX 9.0b	4.09.0000.0902 (RC2)		2003
DirectX 9.0c	4.09.0000.0904 (RC0)	Windows XP SP2	2004
DirectX 9.0c	4.09.0000.0904	*První verze obsahující dynamickou knihovnu D3DX*	2005
Direct3D 10 (Windows Vista)	(?)	Nová verze Direct3D určená pouze pro Windows Vista	

3 Algoritmus

3.1 Pravidla hry Dáma

Česká dáma podle pravidel České federace dámy

- Hraje se na šachovnici 8x8.
- Soupeři mají na začátku po 12 kamenech stojících na protilehlých stranách v prvních třech řadách na černých políčkách.
- Kameny se pohybují po diagonálách po černých políčkách, pouze vpřed a nemohou přeskakovat kameny vlastní barvy.
- Pokud obyčejný kámen dojde na druhou stranu šachovnice, přemění se v dámu.
- Dáma se pohybuje diagonálně dopředu a dozadu o libovolný počet polí.
- Jestliže se kámen nachází na diagonále v sousedství soupeřovy figury, za kterou je volné pole, je povinen ji přeskočit, obsadit toto volné pole a odstranit přeskočenou soupeřovu figuru z desky.
- Skákání je povinné, může-li hráč skákat jak dámou tak obyčejným kamenem, musí skákat dámou. V případě, že může jedním kamenem provést více variant skoku, je na něm kterou si vybere, musí ovšem variantu doskákat (např. můžu jedním kamenem skočit jeden nebo tři kameny, skočím tedy jeden nebo tři. Nemohu skočit jen dva z druhé varianty.)
- Při vícenásobném skoku se kameny odstraní až po dokončení celé sekvence. Přes jeden kámen ovšem nelze skákat vícekrát.
- Hráč který je na tahu a nemůže hrát (nemá kameny, nebo má všechny zablokované), prohrál. Partie končí remízou tehdy, když je teoreticky nemožné vzít soupeři při pozorné hře žádnou další figuru.
- Jestliže některý z hráčů zahraje tah v rozporu z pravidly (např. opomenutí skákání), je na jeho protihráči, jestli bude vyžadovat opravu tahu nebo ne.

Žravá dáma

Varianta české dámy, při které vyhrává hráč, který nemá žádnou figuru. Pokud hráč může brát, tak brát musí.

Anglická dáma

Angličané této hře říkají draughts, Američané tuto dámu nazývají checkers. Odlišnost od české dámy:

- Na konci hracího pole se kameny povyšují na krále, který se může pohybovat pouze o jedno pole všemi směry

Mezinárodní dáma

Hraje se na šachovnici 10x10, figurky stojí v prvých čtyřech řadách. Platí pravidla jako v klasické dámě, ale kámen může brát i směrem dozadu Přednost při skákání má větší počet přeskočených kamenů a to i před dámou.

Brazilská dáma

hraje se na malé desce 8x8, kameny můžou skákat i dozadu a přednost při braní má většina.

Španělská dáma

Ve Španělské dámě se také hraje na desce 8x8, ovšem na bílých polích. přednost při braní má většina, a pokud hráč může vzít dáma nebo kámen, musí přeskočit dámu (počítá se jako 2 kameny)

3.2 Rekurze

Pro převedení algoritmu do programovacího kódu Delphi jsem použil tzv. rekurzi. Rekurzivní podprogram je podprogram, který volá sebe sama. Význam těchto funkcí je vidět především u problémů, které jsou „přirozeně“ rekurzivní. Jako příklad bývá často zmiňován algoritmus pro výpočet faktoriálu (i když rekurzi lze vypočítat i bez použití rekurzivního volání). Připomeňme, že faktoriál čísla n se značí n! a platí pro něj následující vztah (hvězdička znamená násobení):

```
n! = n * (n - 1) * (n - 2) * ... * 2
```

Příklad: faktoriál čísla 5:

```
5! = 5 * 4 * 3 * 2 = 120
```

Prohlédneme-li si vztah dobře, snadno zjistíme, že jej lze vyjádřit též prostřednictvím následujících dvou vztahů:

```
n! = n * (n - 1)!
0! = 1;
```

Jinak řečeno – faktoriál z čísla n počítáme jako (n * faktoriál_z_čísla_o_jedna_nižší). Faktoriál počítáme pomocí faktoriálu – typická situace pro využití rekurzivního podprogramu:

```
function Faktorial(n: integer): double;
begin
  if n = 0 then
    Faktorial := 1
  else
    Faktorial := n * Faktorial(n - 1);
end;
```

Funkce volá opakovaně sebe samu. Při každém novém volání sama sobě předá nižší číslo, z něhož se počítá faktoriál. Pozor je nutné dávat na správné stanovení "zastavující podmínky" rekurze. V předchozím příkladu byla touto "stopkou" podmínka if n = 0. Kdyby nebyla zastavující podmínka uvedena vůbec nebo by byla stanovena chybně, volala by funkce sama sebe donekonečna, což by dříve nebo později způsobilo zaplnění operační paměti a zatuhnutí programu.

Jedním z největších problémů rekurze je její značná paměťová náročnost. Každé volání funkce je spojeno s kopírováním mnoha informací do paměti a ani s tím spojené časové náklady nejsou zanedbatelné.

3.3 Princip metody minimaxu

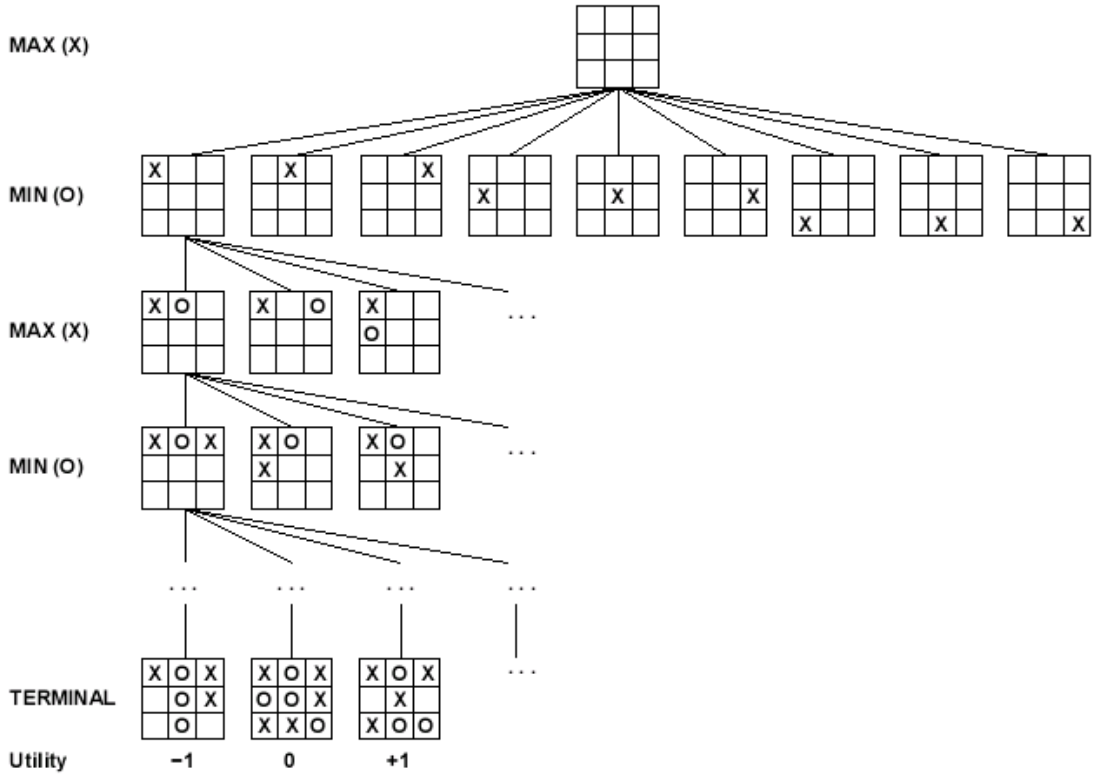
Nyní uvádím, pro lepší pochopení, několik pojmů, používaných u prohledávání při hraní dvou hráčových her.

Adversariální prohledávání (adversarialsearch) stavového prostoru (hraní her), implementuje inteligencí rozhodování v komutativním prostředí, kde dva nebo více agentů mají konfliktní cíle.

Teorie her – komplikovaná vědní disciplína, součást ekonomiky, která analyzuje chování jednotlivých hráčů a výhodnost jejich strategií (především s ohledem na stabilitu, maximalizaci společného zisku, atp.) ve více hráčovém prostředí.

Stavový prostor hry (herní strom) je dán opět počátečním stavem, stavovým operátorem, testem na ukončení hry a užitkovou funkcí.

Cílem adversariálního prohledávání je nalézt nikoliv stav prostoru, nýbrž herní strategii. Strategie vybere nejvhodnější tah s ohledem na racionalitu protihráče. Optimální herní strategie je taková strategie (nepřesně), pro kterou neexistuje žádná lepší strategie, která by vedla k lepšímu výsledku při hře s bezchybným oponentem.



Obr. 1 – Metoda minimaxu u piškvorek

MINIMAX algoritmus dělí stavový prostor do MAX a MIN úrovní. Na každé MAX úrovni hráč A vybere tah s maximálním užitkem a na každé MIN úrovni vybere protihráč tah naopak minimalizující užitek hráče A.

Problém minimaxu je,že počet stavů hry roste exponenciálně s počtem tahů. V reálných hrách je prostor hry ohromný a nelze ho celý prohledat v rozumném čase. Tento problém lze řešit pomocí omezení hloubky nebo odhadem místa přesné hodnoty.

Uvádím zde jednoduchý zdrojový kód použití minimaxu pro hru piškvorky. Program počítá s odpověďmi soupeře a to do hloubky kterou si nastavíme.

Po každém zahraném půltahu zavoláme kód, který vyzkouší všechny odpovědi soupeře na náš tah. Jednu po druhé zahraje, ohodnotí je statickou ohodnocovací funkcí a zahraje zpět a zapamatuje si hodnotu nejlepšího tahu, tentokrát z hlediska soupeře. Za cenu konkrétního našeho tahu z úvodní pozice je tak prohlášena cena nejlepší soupeřovy odpovědi.

```
function minimax(p: Pozice,hloubka:integer) {
var  t:Tahy;
     i, indexNejlepsiho:integer;
     cena:integer;
begin
  { V případě koncové pozice nebo nulové hloubky propočet ukončíme. }
  if (jsemVKonci(p)) then result -KONEC;
  if (hloubka <= 0) then  result hodnotaPozice(p);

  t := generujTahy(p);
  nejlepsi := -NEKONECNO;

  for i := 0 to  i = tahy.pocet do
  begin
  zahrajTah(p, t[i]);
    cena := -minimax(p, hloubka - 1);
    zahrajTahZpet(p, t[i]);
{ Nejlepší tah nás nezajímá, důležitá je jen jeho hodnota. }
    if (cena > nejlepsi) then nejlepsi := cena;
  end;
  result := nejlepsi;
end;
```

V kořeni stromu propočtu je postup nepatrně jiný, především nás zajímá nejlepší tah samotný a nikoliv pouze jeho cena.

```
Function Tah nejlepsiTah(p: Pozice, hloubka:integer) {
 var t:Tahy;
    i, indexNejlepsiho:integer;
    cena, nejlepsi:integer;
begin
  t := generujTahy(p);
  nejlepsi := -NEKONECNO;

  for i := 0 to i = tahy.pocet do
  begin
    zahrajTah(p, t[i]);
    cena := -minimax(p, hloubka);
    zahrajTahZpet(p, t[i]);
    if (cena > nejlepsi) then
      begin
        nejlepsi := cena;
        indexNejlepsiho := i;
      end;

  end;
  result := t[indexNejlepsiho];
end;
```

4. Vývoj programu

4.1 Psaní programu

Program je složen ze tří formulářů. V prvním se hraje dáma. V druhém vidíme obraz snímaný kamerou, můžeme zde nastavovat rozlišení kamery a číslo sériového portu pro komunikaci s robotem. Třetí ukazuje všechny možné tahy hráčů a můžeme zde načítat a ukládat pozice z a do textového souboru. Hlavní kód programu se nachází v jednotce `dama.pas`, zde je algoritmus hry, ovládání form1 a příkazy pro robota. V unit_kam.pas ovládáme druhý `form` a `unit3.pas` obsluhuje `form_tahy`. V jednotce typy.pas jsou předefinované typy proměnných. Nutno připomenout, že pro chod programu využíváme jednotek robot.dcu (ta využívá komunikace.dcu), kamera.dcu a knihovny DirectX.

Při psaní bylo důležité si nejprve uvědomit, jaké typy proměnných budeme používat. Všechny níže uvedené typy jsou zapsány v jednotce typy.pas. V celém programu se pracuje s polem čísel, představujícím hrací plochu, proto je vytvořen type `Tpole = array[0..7,0..7] of integer`. Pro ukládání jednotlivých pozic je vytvořeno pole pozic type `Tpole_pozic = array[0..30] of Point`. Type `minitah` obsahuje informaci, odkud kam se pohnu, kolik kamenů seberu, a jejich souřadnice. Minitahy jsou dílčí kroky při každém tahu. Jsou použity v poli cesta v `Ttahu`. `Ttah` je opravdový tah kamene. Uvádí, z jakých souřadnic se kam pohnu, cestu kterou se pohnu (to je pole minitahů a jejich počet), souřadnice figurek které seberu, jejich počet, jestli se jedná o dámu a cenu tohoto tahu. Dále je využito `Tpole` tahů, které obsahuje pole Ttahů a jeho délku. Všechny proměnné jsou nastaveny na výchozí hodnoty v proceduře `zacatek`.

Nyní se pokusím přiblížit samotný kód hlavního programu (Unit `dama.pas`). Pro lepší pochopení slouží vývojový digram, viz. následující podkapitola. Nejprve bych rád vysvětlil, jak pracují nejdůležitější funkce a procedury. Následovat bude jejich využití v celém programu a jejich propojenost s ostatními.

Nejdůležitější funkce je `function TForm1.generuj_tahy`. Tato funkce vrací pole všech možných tahů dané barvy. Nejprve si zjistí, se kterými kameny může

táhnout a uloží si jejich pozice do pole pozic. Následně se u každé pozice rozliší, zda jde o dámu či nikoliv. Zjistí, zda se kámen může pohnout, jestli že ano, zavolá funkci posun, která vytvoří tah s příslušnými souřadnicemi a uloží jej do výsledného pole tahů. Obyčejný kámen se může pohybovat pouze jedním směrem do obou stran. Dáma se pak může pohybovat diagonálně všemi směry. U skoku je situace trochu složitější. Musíme zjistit, jaké kameny se nacházejí okolo a kolik jich mohu skočit. Volají se další funkce `skok`, `zkus_skok`, `skokmini`, `skokmini_dama`, `zkus_skok_dama`. Tyto funkce vytvářejí tahy pro skok, souřadnice skočených kamenů, cestu kterou se pohybuji apod.

`Proceduru TForm1.hlavni` voláme pro vypočítání všech možných tahů. Tahy i s příslušnou cenou se zobrazí v `listboxech` ve formuláři `form_tahy`. Pod nimi se také ukáže nejlepší tah, vybraný na základě ceny. V hlavní proceduře si vytvoříme tah (Ttah) a zavoláme funkci `nejlepší_tah`. První parametr je hloubka prohledávání a druhý je barva. Funkci zavoláme pro obě barvy, vypíšeme nejlepší tahy a uvolníme paměť.

Funkce nejlepší_tah je mozkem celého programu. Vrací nám nejlepší možný tah v dané situaci. Nejprve zavolá funkci generuj tahy s parametry hlavního pole a barvou. Výsledek uložíme do pole tahů. Celé toto pole prohledáme. Na začátek zahrajeme tah (`zahraj_tah`). Ohodnotíme dosaženou pozici (minimax), uložíme cenu tahu a vrátíme tah zpět (`vrat_tah`). Ten to postup provedeme pro všechny kameny, které se mohou hnout. Zde je vidět metoda minimaxu. Zahrajeme náš tah figurkou. Poté zahrajeme všechny možné odpovědi soupeře. Soupeřovy tahy vrátíme zpět a hrajeme jinou naší figurkou. Hloubka prohledávání určuje, kolikrát necháme soupeře odpovědět. Po prohledání všech pozic zjistíme, při kterém tahu jsme schopni vzít nejvíce figurek. To je důležité proto, abychom věděli, jestli hráč hrál správně, anebo mu musíme figurku, se kterou měl skákat, vzít. Přednost má pochopitelně dáma. Cenu všech ostatních tahů snížíme o jednu, protože při zahrání jiného tahu přijdeme o kámen, který měl skákat. Zároveň vybereme tah s nejlepší cenou. Pokud je více nejlepších tahů, vybereme jeden náhodně. To je důležité při hře proti počítači, aby se nedaly předvídat jeho tahy. Nakonec funkce vrací nejlepší tah a uvolní paměť.

`Funkce minimax` je velmi podobná předešlé. Její parametry jsou tah, hloubka a barva. Vrací cenu tahu. Na začátku zjistíme, jestli již hloubka není rovna nule. Pokud ano, ohodnotíme současné rozestavení kamenů (`hodnotaPozice`) ze

strany příslušné barvy a vyskočím z minimaxu. Pokud ne, postupujeme podobně jako u funkce nejlepší tah. Zjistíme všechny možné tahy, zahrajeme je, určíme cenu a vrátíme tahy zpět.

K ohodnocení pozice kamenů slouží funkce `hodnotaPozice`, která prohledá celé hrací pole a sčítá figurky hráčů. Vrací hodnotu pozice kamenů na scéně. Výsledek je určen součtem figurek jedné barvy, od kterého je odečten součet figurek barvy druhé. Obyčejný kámen má váhu jedna a dáma má váhu 5. Tento vzorec je velmi jednoduchý, ale vystihuje to nejdůležitější, a to je počet kamenů na scéně. Podle hodnoty pozice jsou vybírány nejlepší tahy počítače.

Procedury `udelej_tah_nej` a `udelej_tah` mají podobnou funkci. První zahraje nejlepší tah a druhá námi vybraný tah. Parametry jsou barva a popřípadě i index tahu. Procedura `udelej_tah_nej` se využívá v `Timeru1`, pokud mají hrát počítače proti sobě nebo když proti nám hraje počítač. Nejprve tah zahraje robot (`robot_udelej_tah`). Poté zjistí, zda byl zahrán správný tah nebo jestli mohl hráč sebrat víc kamenů, a tudíž o příslušný kámen přijde (`robot_seber`). Následně vykreslíme (`vykresli`) novou hrací plochu, zjistíme počet kamenů (`score`) a uvolníme po sobě paměť.

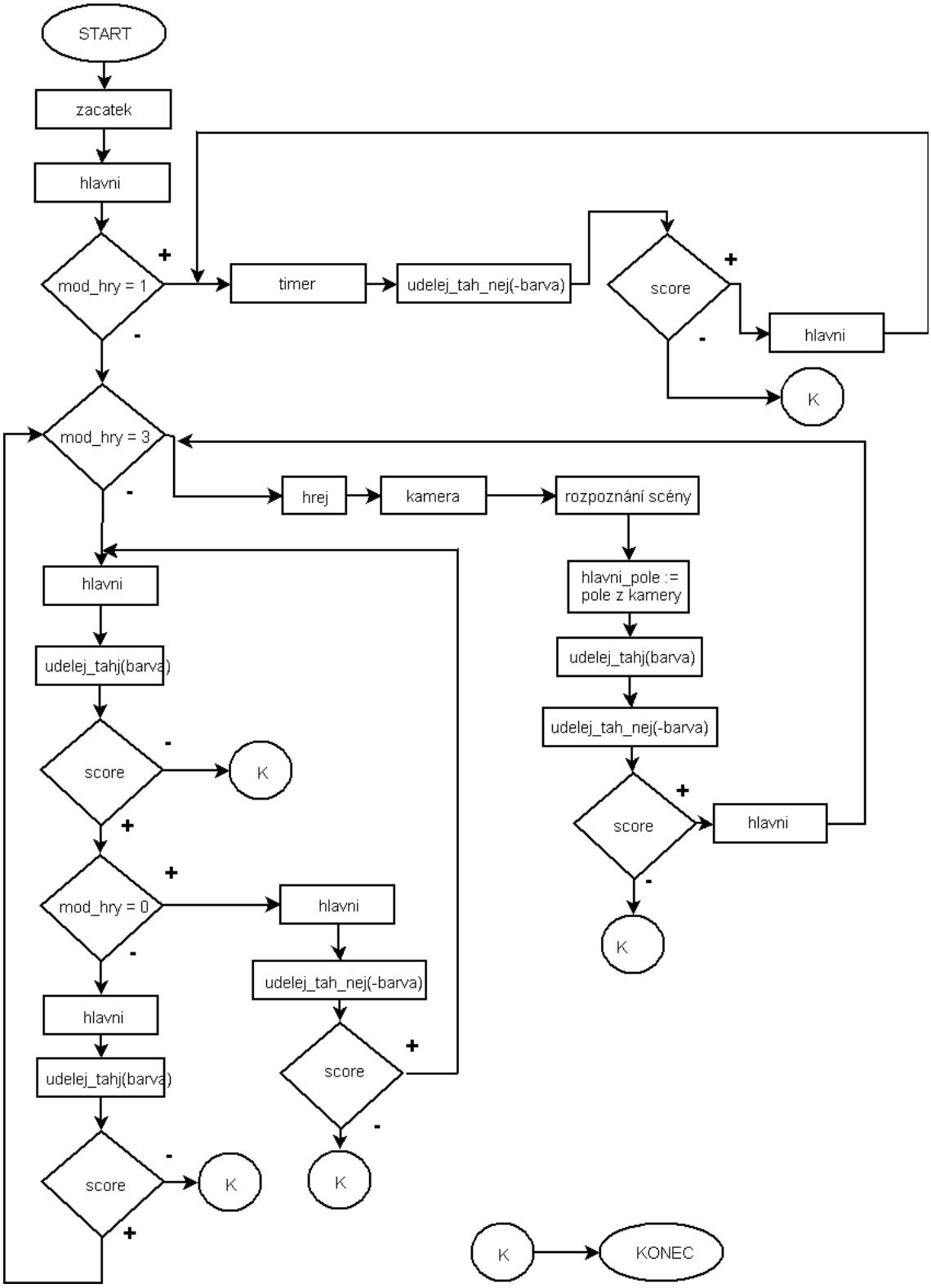
Procedura vykresli namaluje aktuální rozestavení kamenů. Vstupní parametr je `hraci_pole`. Vše se vykresluje na `Image1`. Nejprve se vykreslí mřížky hrací plochy. Podle čísla v poli vykreslíme na dané pozice shapy. Například je-li to číslo rovno 1, vykreslíme bílý `shape`. U každého `shape` nastavíme velikost, barvu, tvar a události `on mouse down`, `on mouse move` a `on mouse up`. Těmito událostmi ovládáme kameny a hrajeme hru.

Celou hru spouští tlačítko start. Při stisku zkontroluje, zda byl vybrán typ hry a případně zamezí další změně. Podle módu hry spustí časovač nebo odblokuje kameny na scéně. Nakonec se přemění v tlačítko konec, které ukončí rozehranou hru a připraví ji pro novou hru.

Dále jednotka obsahu řadu procedur a funkcí obsluhující prvky na scéně, procedury k pomocným výpočtům. Důležité jsou i procedury obsluhující události při kliknutí, tahu a puštění tlačítka myši. Ty nám zajišťují pohyb kamenů na ploše.

4.2 Vývojový diagram

mód 0: PC x člověk, 1: PC x PC, 2: člověk x člověk, 3: PC x člověk (bez použití robota). Blok score testuje konec programu (počítá kameny na scéně).



Obr. 2 – vývojový diagram

4.3 Problémy při tvorbě programu

V první verzi programu jsem nepoužil žádnou metodu minimaxu, a tudíž program nevypočítával tahy do určité hloubky. Vybíral jsem pouze kameny, které mohou skočit a skočil jsem s nimi. Program nebyl příliš složitý, avšak nepřehledný. Při hře proti počítači se dalo dosáhnout lehce vítězství. V druhé verzi je již použita metoda minimaxu, možnost nastavení hloubky prohledávání a také rekurze, díky které se program více zpřehlednil.

Při tvorbě algoritmu hry dáma v Delphi byla použita rekurze (viz. výše). Tento postup je sice přehledný, ale nemusí být vždy nejefektivnější. Často byla použita pole možných tahů vytvořených voláním konstruktoru. Zde je nutno podotknout, že každé volání konstruktoru nám zabere část paměti, a proto nesmíme zapomenout po sobě tuto paměť uvolnit. V případě tohoto typu programu by totiž po několika minutách hraní mohla zaplněná paměť dosahovat i řádu gigabytů!

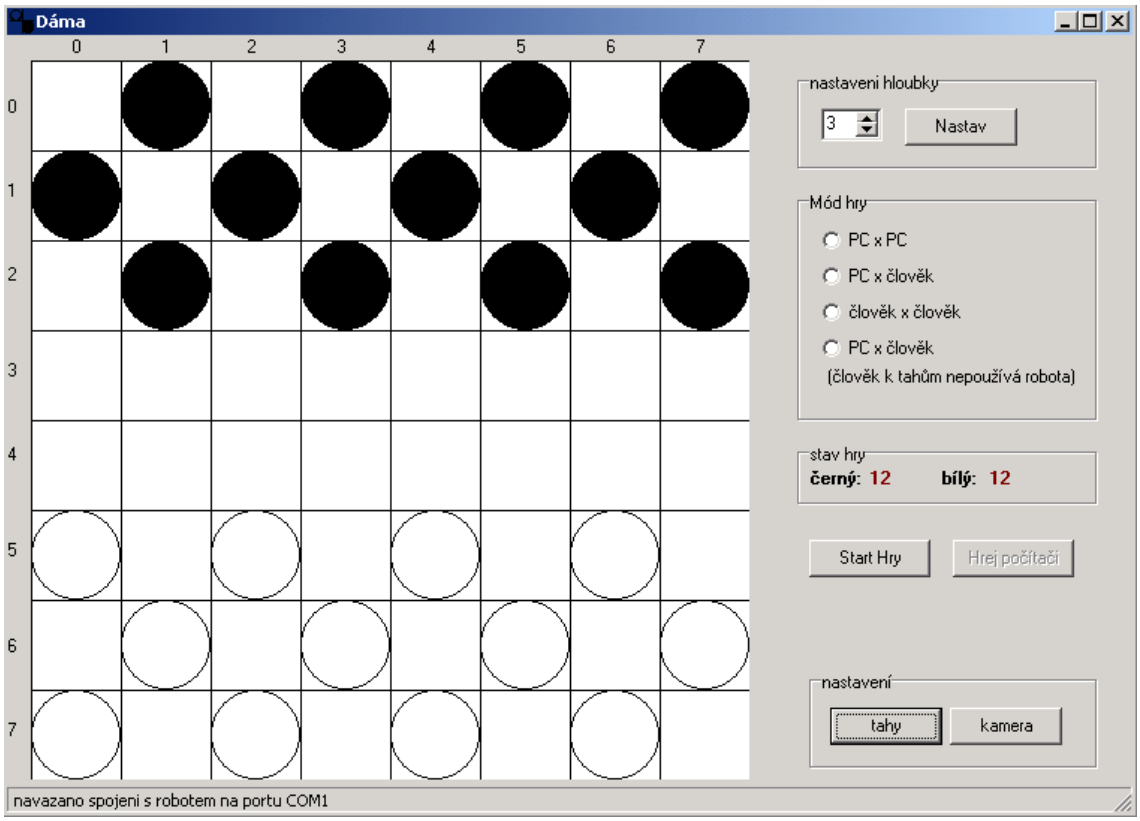
Převedení samotného algoritmu do programu není až tak těžké, ale je nutné ošetřit podmínky, jako je například sebrání figurky, která má skákat a neskočí. Nejtěžší bylo napsat hlavní část programu, která se skládá z funkce generuj tahy. Tato funkce vygeneruje všechny možné tahy. Je zde spousta podmínek pro pohyb kamene a jeho skok.

Pro vykreslování pole kamenů bylo použito pole naplnění shapy. Jednodušší vykreslování by se dalo realizovat jednoduchým kreslením na canvas. Na začátku psaní se mi zdál způsob se shapy přehlednější. Musí se však při každém překreslování mazat.

5. Výsledný program

5.1 Obsluha programu

Program není nikterak graficky ohromující. Grafická nadstavba by mohla být součástí jiné bakalářské práce. Prostředí je jednoduché a lehce potopitelné na ovládání. Pro jistotu zde uvádím “návod k použití”.



Obr. 3 – Hlavní okno programu

Při spuštění programu se zobrazí základní rozmístění kamenů pro hru dáma. V levé části vidíme pole 8x8 a aktuální rozmístění kamenů. Na pravé straně jsou ovládací prvky. Spodní lišta informuje o stavu komunikace s robotem na daném portu.

Mód hry:

Nejprve vybereme v jakém módu chceme hrát a poté stiskem tlačítka START spustíme hru.

PC x PC – při přepnutí do toho módu proti sobě hraje počítač s počítačem, tlačítkem konec hry se hra zastaví

PC x člověk – počítač hraje s černými figurkami, člověk ovládá bílé a tedy začíná

Člověk x člověk – v tomto módu mohou hrát dva lidé proti sobě

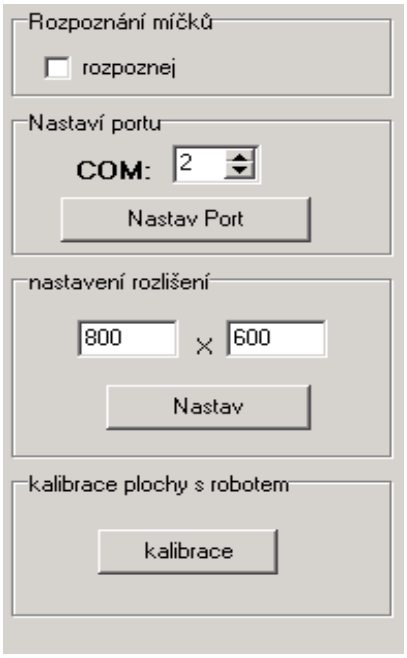
PC x člověk (člověk k tahům nepoužívá robota) – počítač hraje s černými figurkami, člověk ovládá bílé a tedy začíná. V tom to módu člověk ručně pohybuje s míčky na hrací ploše.

Provedení tahu

Po spuštění hry klikneme na kámen se kterým chceme hrát a tahem ho přeneseme na konečné místo našeho tahu. Pokud se budeme snažit kámen umístit mimo možné tahy, vrátí se zpět do počáteční pozice. Při skákání umístíme kámen do konečné pozice skoku.

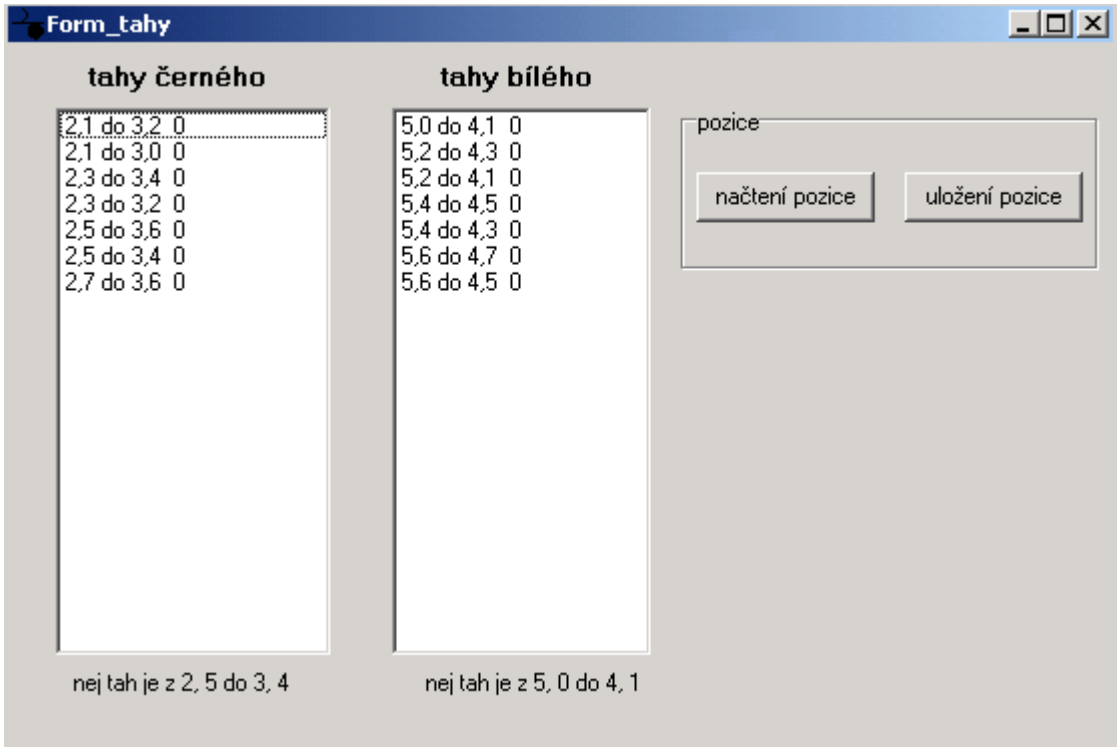
Hloubka – v pravém horním rohu je natavení hloubky prohledávání. Při změně hloubky je nutno tuto změnu potvrdit tlačítkem *najdi tahy,* aby se všechny tahy přepočítaly do nastavené hloubky.

Tlačítko *kamera* otevře formulář, který slouží k nastavení hrací plochy do předem daných souřadnic robota, k výběru rozpoznávané plochy snímaného obrazu kamerou, k rozlišení kamery a k nastavení com portu pro komunikaci s robotem. Ukazuje nám obraz snímaný kamerou. Po zaškrtnutí políčka rozpoznej, vybereme část obrazu který bude rozpoznávat barvu míčků. Pokud se pozice scény ve vztahu ke kameře během hry změní, musíme opět označit hrací pole. Stiskem tlačítka kalibrace se robot začne postupně přemisťovat do rohů hracího pole.



Obr. 6 – panel z formuláře kamera

Tlačítka *tahy* otevře formulář tahy, ve kterém vidíme všechny vypočítané tahy. Při označení tahu se vykreslí na hlavním formuláři zvýrazněná počáteční a koncová pozice kamenu. V pravé části jsou tlačítka *načti ze souboru* a *ulož,* které načítají a ukládají pozice do textových souborů.



Obr. 5 – Formulář form_tahy

5.2 Možná vylepšení

Program by se rozhodně dal vylepšit po grafické stránce. Prostředí je velmi jednoduché a prosté. Například animace pohybu kamenů by byla jistě velmi zajímavá.

Hloubka prohledávání tahů se dá nastavit maximálně na hodnotu 6. To je dáno obrovským množstvím prohledávaných tahů. Počítač větší množství informací nezpracuje. Větší hloubky prohledávání by se dalo dosáhnout zefektivněním procedur prohledávající pole tahů. Dále pak vyřazením tahů které nemá smysl prohledávat.

Tahy počítače jsou vybírány na základě nejlepší ceny tahu. Tato cena se určuje pouhým sečtením kamenů na scéně s tím, že obyčejný kámen má váhu jedna a dáma váhu 5. V ohodnocování tahů by se mohla vzít v úvahu vzdálenost kamene od konce hrací plochy – jak daleko má k tomu stát se dámou.

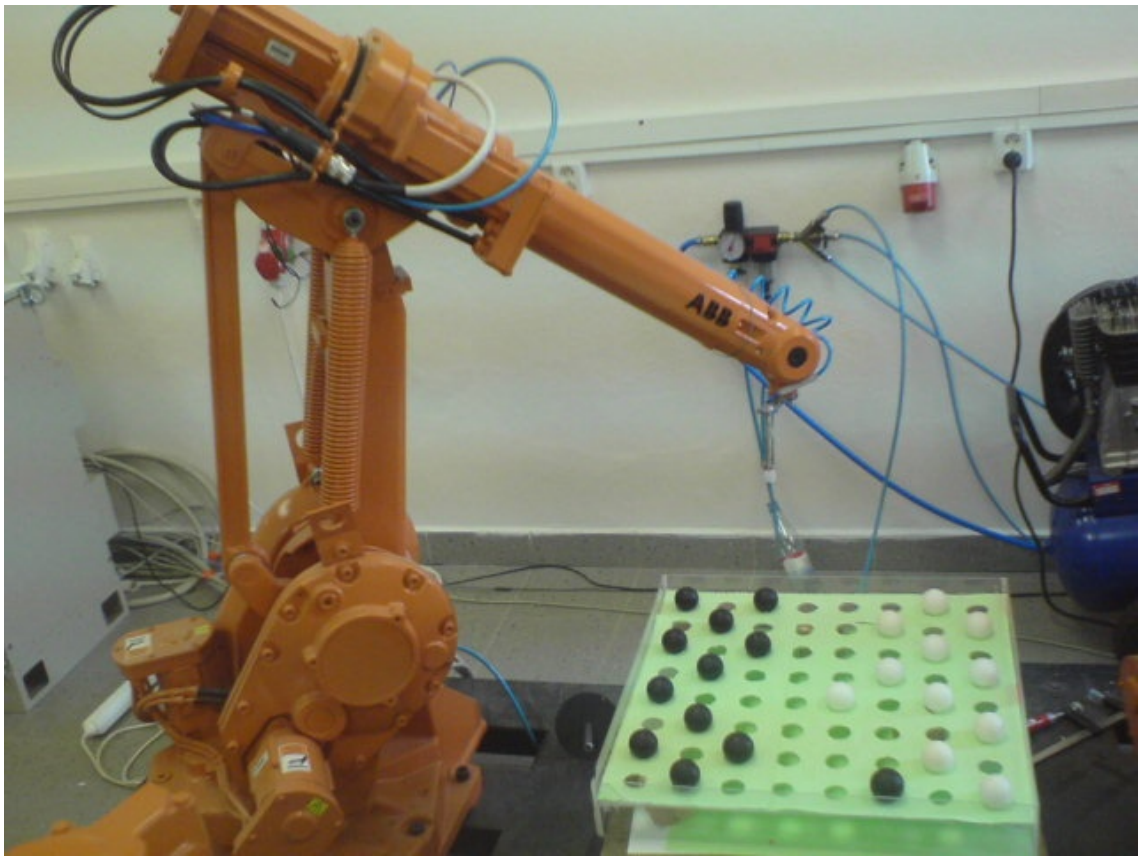
V použité jednotce robot.dcu není ošetřený problém zpětné vazby. Tudíž robotu posílám data, ale nevím, jestli už je vykonal nebo stále pracuje. Problém nastává v případě módu hraní počítač s počítačem. Program posílá robotovi příkazy, jak má hrát, ale robot nestačí plnit dané úkoly a rozestavní kamenů na obrazovce neodpovídá rozestavení míčků na hracím poli. Přidal jsem proto ke každému povelu k pohnutí robota pauzu v programu. Tyto pauzy však problém neřeší. Daný tah může být libovolně dlouhý. Takže když například táhnu pouze o jedno políčko, robot potom zbytečně dlouho čeká na další povel. Naopak je-li tah složitější, např. skok několika kamenů, program robota předběhne. Celou situaci by vyřešila zpětná vazba od robota.

5.3 Použitý Hardware a software

K tvorbě programu byl použit stolní počítač s procesorem Intel Pentium 3, 600MHz, 128MB RAM s operačním systémem Microsoft Windows XP Profesional Service pack 2. Rozhraní Microsoft DirectX(r) 9.0c. Program byl napsán v prostředí Delphi 7. Robot byl použit ABB. Jako nástroj měl přísavku, která byla připojena ke kompresoru.

Hrací plochu tvoří přípravek vyrobený z plexiskla o rozměrech 50x50 cm. Je v něm 64 (8 x 8) děr pro pingpongové míčky reprezentující kameny.

Typ kamery AWT MERLIN. Použita může být jakákoliv kamera podporující rozhraní DirectX.



Obr. 6 – Robot ABB při hraní dámy

Závěr

Zadání vytvořit algoritmus pro hraní dámy, se podařilo splnit. Program bez problémů propočítává tahy do hloubky 4. Prohledávání do větší hloubky je sice možné, ale je třeba vzít v úvahu, že počet možných tahů narůstá exponenciálně, proto jsou tyto výpočty závislé na výkonu PC. Vyšší hloubky prohledávání by se dalo dosáhnout zefektivněním algoritmů. Bohužel dámu proti sobě nemohou (alespoň ne ve školní laboratoři robotiky) hrát dva roboti zároveň. Problém je v tom, že druhý robot je staršího data výroby a nedisponuje sériovým portem, tudíž s ním nemůžeme komunikovat po sériové lince. Program tedy ovládá pouze jednoho robota který hraje za oba hráče. Na první pohled robot po každém tahu nesmyslně odjíždí od hracího pole. Tento manévr je ale nezbytný pro snímání hracího pole kamerou umístěnou přímo nad scénou. Robot při hraní jednoduše brání rozpoznání míčků v poli. Program by se mohl vylepšit po grafické stránce. Vhodné by bylo ošetření odezvy robota, aby program věděl, zda již byl příkaz vykonán a mohl poslat další.

Literatura

[1] ing. Tomáš Pluhař Dokumentace ke knihovnám robot.dcu a kamera.dcu

[2] Michal Pěchouček Hraní dvouhráčových her – Adversariální prohledávání stavového prostoru

[3] Svoboda L, Voleš P., Konšal T., Mareš M. 1001 tipů a triků pro Delphi. Computer press Brno 2002.

[4] Holan T. Delphi v příkladech, BEN – technická literatura 2001