

**Střední průmyslová škola strojní a elektrotechnická  
a Vyšší odborná škola Liberec**

**vyšší odborné studium**

**studijní obor**

**26-31-N/015 Počítačové systémy**

**ABSOLVENTSKÁ PRÁCE**

Jindřich Vrba

Školní rok 2005/2006

**Střední průmyslová škola strojní a elektrotechnická  
a Vyšší odborná škola Liberec**

**Vyšší odborné studium**

Jindřich Vrba

**Hra Formulky**

absolventská práce

Rychnov u Jablonce nad Nisou, květen 2006

# Absolventská práce

Autor: Jindřich Vrba, rodné číslo 821125/2522  
SPŠSE a VOŠ Liberec  
Studijní obor 26-31-N/015 Počítačové systémy  
Školní rok 2005 / 2006, 3. ročník

Téma práce: Hra Formulky  
schváleno ředitelem školy dne \_\_.\_\_. 2005

Vedoucí práce: Ing. Marek Pospíchal, SPŠSE a VOŠ Liberec

Prohlášení: Absolventskou práci jsem vypracoval samostatně  
jen s použitím citované literatury.

Květen 2006

.....

Jindřich Vrba

Anotace:

# Obsah

<b>1 Úvod</b> .....	<b>6</b>
<b>2 O hře</b> .....	<b>7</b>
<b>3 Ještě než začneme</b> .....	<b>9</b>
3.1 Vývojové prostředí.....	9
3.2 Na co se připravit.....	9
<b>4 Přehled</b> .....	<b>11</b>
4.1 Editor.....	11
4.2 Hra.....	11
<b>5 Struktura</b> .....	<b>13</b>
5.1 Konfigurace.....	13
5.2 Mapa.....	13
<b>6 Taktika počítače</b> .....	<b>14</b>
<b>7 Techniky</b> .....	<b>18</b>
7.1 Editor.....	18
7.1.1 Hlavní formulář.....	18
7.1.1.1 Aktualizuj okno.....	19
7.1.1.2 Otevřít obrázek.....	19
7.1.1.3 Vytvoření nové mapy.....	19
7.1.1.4 Stisknutí myši.....	20
7.1.1.5 Pohyb myši.....	20
7.1.1.6 Uvolnění myši.....	20
7.1.1.7 Označ pole.....	21
7.1.2 Formulář pro nastavení mřížky.....	21
7.1.2.1 Nastavení mřížky.....	21
7.1.2.2 Zavření formuláře.....	22
7.1.3 Formulář pro ukládání.....	22
7.1.3.1 Stisknutí tlačítka OK.....	23
7.1.3.2 Ulož mapu.....	23
7.1.4 Formulář pro výběr mapy.....	24
7.1.4.1 Zobrazení formuláře.....	24
7.1.4.2 Stisknutí tlačítka OK.....	24
7.2 Hra.....	25
7.2.1 Hlavní formulář.....	25
7.2.1.1 Vytvoření formuláře.....	26
7.2.1.2 Tah počítače.....	26

7.2.1.3	Řešení jízdy přes hranu.....	29
7.2.1.4	Pohyb.....	29
7.2.1.5	Šikmý dojezd.....	30
7.2.1.6	Trojité možný dojezd.....	31
7.2.1.7	Možný dojezd.....	33
7.2.1.8	Hledej cíl.....	33
7.2.1.9	O kolik popojet.....	34
7.2.1.10	Průchozí.....	34
7.2.1.11	Můžu k šípce.....	36
7.2.1.12	Lze jet do cíle.....	36
7.2.2	Formulář nastavení.....	37
7.2.2.1	Zobrazení formuláře.....	37
7.2.2.2	Stisknutí OK.....	37
<b>8</b>	<b>Závěr.....</b>	<b>38</b>

# 1 Úvod

Cílem této práce bylo naprogramovat hru, známou jako autíčka. Samozřejmě včetně editoru pro vytváření vlastních map. Hlavním námětem a také nejtěžším úkolem se stalo programování taktiky počítačového hráče.

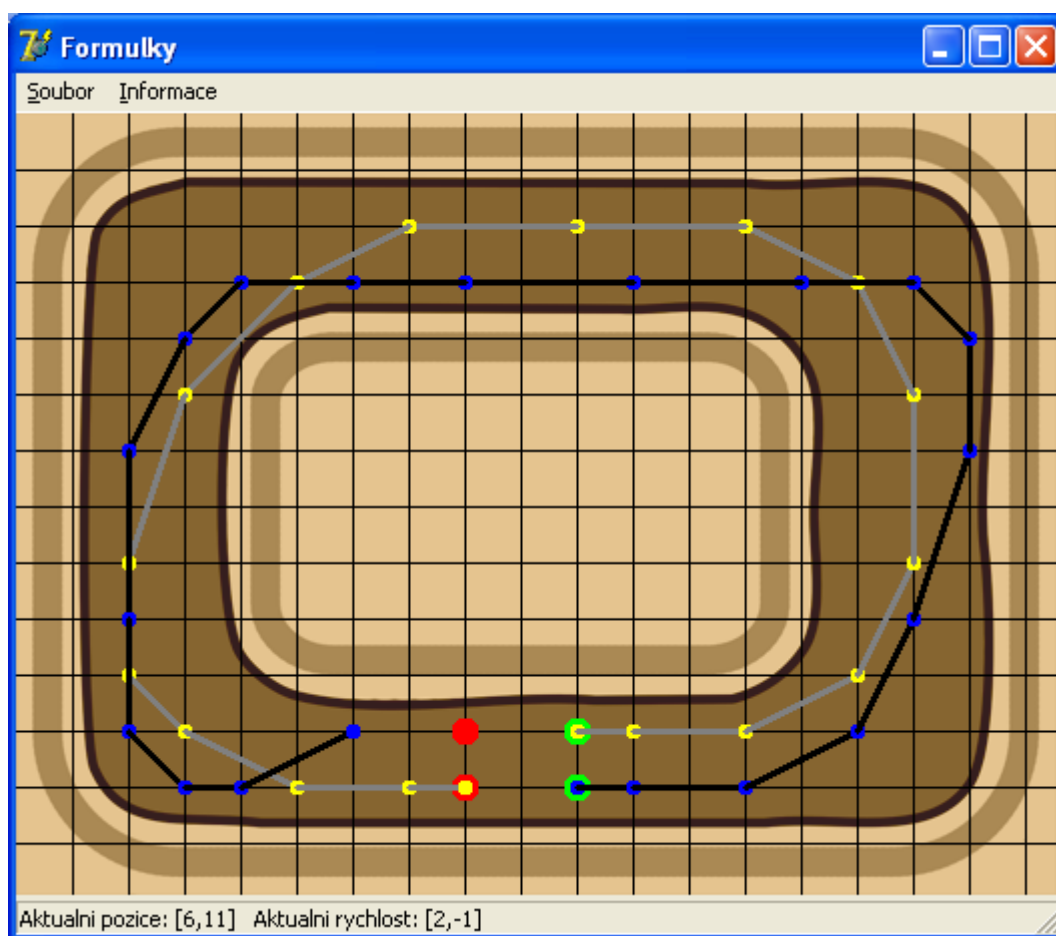
Tuto hru hodlám zveřejnit pod svobodnou licencí GNU/GPL, proto jsem se snažil především o čisté a průhledné řešení problémů. Aplikace je napsána v Delphi 7 a téměř celá je založena na matematických úvahách. Použité techniky budu doplňovat obrázky pro snazší pochopení.

Na následujících stránkách nejprve vysvětlím pravidla hry, dále jak editovat mapu a hrát. Následovat bude trochu informací o použitém programovacím prostředí a úvahy pro přípravu k samotnému naprogramování. Dále vás seznámím s konfiguračními soubory hry, strukturou map a jejich konfiguračními soubory. Vysvětlím také základní principy, podle kterých se počítač orientuje v mapě. Na závěr popíši zdrojový kód obtížnějších funkcí a procedur. Na přiloženém CD najdete sestavenou hru a kompletní zdrojové kódy.

## 2 O hře

Mnozí jste ji asi už někdy hráli na čtverečkovaném papíře proti lidským protihráčům. Asi nejjednodušší by bylo udělat přesně to samé, čili hru více lidí po síti. Realizace mi ale nepřišla dostatečně zajímavá, a tak jsem hru pojal především jako souboje člověka vůči počítači.

Někteří tuto hru ještě neznají, a tak by bylo vhodné popsat její základní principy. Jelikož existuje více způsobů hry, doporučuji nahlédnout i zkušeným. Pro popis budu používat následující obrázek:



Začíná se na zelených bodech, cílem je dorazit na červené pole na co nejméně tahů a nevybourat se. Vybouráním se totiž hra končí.

Základem je postupně snižovat, či zvyšovat rychlosti podle potřeby. Počítá se zvlášť rychlost v ose X (vodorovný směr) a Y (vertikální směr). Nejlépe vysvětlím pomocí obrázku:



Začínáme na startovním zeleném poli a chceme se pohybovat vpravo. Čili jen ve směru X. Startovní rychlost, jak už to bývá, je nulová, nejprve se tedy rozjedeme na rychlost 1, v dalším tahu 2, v následujícím už můžeme docílit rychlosti 3. Tak bychom mohli pokračovat dále, vždy o 1 více, ale dále nás čeká překážka a tak potřebujeme zpomalit. Stejně jako v reálném životě to nejde naráz a tak zpomalíme nejprve na rychlost 2, poté 1 a až potom už jsme opravdu zastavili.

To byla jednoduchá situace, kde se pohybujeme jen v jednom směru. To se při hře povede málokdy, naopak nejzajímavější část je, jak co nejlépe projet zatáčkou.

Začátek je stejný, jako v předešlém případě. Nejprve jedeme ve směru X rychlostí 1, dále zvyšujeme na 2. Blíží se zatáčka, ve směru X nám rychlost 2 vyhovuje, takže ji ponecháme, ale ve směru Y zvýšíme rychlost z 0 na 1. V následujícím tahu zpomalíme ve směru X na 1 a zrychlíme ve směru Y na 2. V dalším tahu ve směru x opět přibrzdíme a to na rychlost 0 a ve směru Y zrychlíme na 3. Blíží se další zatáčka, ve směru X měníme rychlost z 0 na 1 doleva (vlastně rychlost -1), rychlost ve směru Y snižujeme na 2. V posledním tahu této ukázky zrychlujeme vlevo na 2 a zpomalujeme nahoru na 1.

## **3 Ještě než začneme**

### **3.1 Vývojové prostředí**

Při programování této hry jsem použil Delphi 7 s personální licenci, kterou lze použít, pokud nevyvíjím komerční software komerčně. Popisovat jak instalovat Delphi je asi zbytečné. Po instalaci je výchozí nastavení víceméně v pořádku. Doporučuji jen zapnout si další kontroly při kompilaci. Toto nastavení najdete v menu pod volbou Project → Options → Compiler. Zde v rámečku Runtime Errors zapněte vše. Při závěrečné kompilaci před zveřejněním je naopak vhodné kontroly vypnout, ve Windows prostředí je zvykem chyby skrývat. Běh programu se také vypnutím kontrol zrychlí.

### **3.2 Na co se připravit**

Před samotným programováním čehokoliv je vhodné vymezit si cíle. A to nejenom, co všechno naprogramovat, ale také pro jaký typ lidí, pro jaké operační systémy, pro jaký hardware apod.

Je tak možné, že ihned odhalíte nedostatky a můžete se podle toho zařídit. V mém původním plánu například bylo, aby tato hra byla dostupná nejen pro operační systém Windows, ale také pro všechny systémy dle normy POSIX. Z tohoto plánu jsem nakonec slevil, díky rozporuplné podpoře těchto systémů ze strany společnosti Borland. Jejich Delphi Kylix se už delší dobu nevyvíjí a přestože existují další open-source překladače, jako Lazarus, podpora všech možností Delphi není úplně 100%.

Abych věděl jaké číselné typy použít, počítal jsem s největším možným rozlišením obrazovky 3200\*2400 pixelů a minimální velikostí mřížky 6 pixelů. Vydělením těchto hodnot získáváme maximální rozsah pole:

$$\text{V ose X: } 3200/6 = 533,3$$

$$\text{V ose Y: } 2400/6 = 400$$

Na pozicování v mřížce bude tedy vyhovovat word (0..65535). V určitých případech však budou potřeba i záporná čísla, na to bude vhodné použít smallint (-32768..32767).

Šířka mřížky bude mezi 6 a 100, na to bude vhodný byte (0..255). Jaká bude maximální dosažitelná rychlost není úplně zřejmé, proto je lepší si to spočítat. Jelikož zvyšovat nebo snižovat rychlost lze vždy jen o jednu, bude vzorec následující:

$$0+1+2+\dots+(v_{max}-2)+(v_{max}-1)+(v_{max})=s_{pole}$$

$v_{max}$  ... maximální dosažitelná rychlost

$s_{pole}$  ... maximální velikost pole

Za podmínky, že  $v_{max} \geq 0$  lze vyjádřit předešlý vzorec jako:

$$(v_{max}+1)*\left(\frac{v_{max}}{2}\right)=s_{pole}$$

Po úpravách získáváme kvadratickou rovnici:

$$v_{max}^2 + v_{max} - 2*s_{pole} = 0$$

Podle vzorce  $x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$  :

$$v_{1,2} = \frac{-1 \pm \sqrt{1^2 - 4*1*(-2)*S_{pole}}}{2*1}$$

Zajímá nás jen kladná hodnota, po vykrácení a dosazení:

$$v = \frac{-1 + \sqrt{1 + 8*534}}{2} = \frac{-1 + \sqrt{4273}}{2} \doteq 32,18$$

Vychází maximální rychlost 33, potřebujeme jak kladné tak záporné hodnoty, proto zvolíme shortint (-128..127).

Pro proměnnou tg\_alfa stačí desetinné -33..33, jelikož rychlost může být maximálně 33, zvolíme single ( $1,5*10^{-45}$  ..  $3,4*10^{38}$ ).

Aby to bylo úplně, stanovím si maximální počet startovních bodů 50.

## **4 Použití**

Pravidla už znáte, nyní by bylo vhodné ukázat, jak program používat. V hlavním adresáři hry najdete soubor editor.exe a formulky.exe. Jeden pro spuštění editoru, druhý pak samotné hry.

### **4.1 Editor**

Používání editoru je z uživatelského hlediska opravdu jednoduché. Vyberte si nějaký obrázek formátu bmp a zvolte si rozměr mřížky. Všechna pole se nastaví do výchozí podoby jako nevhodná pro jízdu. Tomu normálně odpovídá červené pole, ale pro lepší přehlednost se zde barva nevykresluje.

Dále už pomocí myši nastavujete trať. Pravým tlačítkem se vyznačují zářivě zelená startovní pole, prostředním tlačítkem zářivě červená cílová pole, nakonec levým tlačítkem se přepíná mezi zeleným hracím polem a červeným polem pro označení míst, kde nelze jet.

Až budete s mapou spokojeni, případně jí budete chtít dokončit později, nezbyvá, než si ji uložit a editor ukončit. Pokud se později budete chtít vrátit k rozdělané mapě, případně vytvořit mapu z nějaké již funkční, stačí zvolit „Upravit mapu“ a některou si vybrat. Trať se vám barevně vykreslí a můžete pokračovat jako při vytváření nové. Pro lepší přehlednost se červené body, označující oblasti mimo trať, nevykreslují.

### **4.2 Hra**

Pokud hru spouštíte poprvé, ukáže se vám výchozí mapa, tu si můžete změnit v nastavení. Na mapě po startu uvidíte svítivě zelená a červená pole. Jak asi tušíte zelená pole jsou startovní. Jakékoliv z nich si vyberte a klikněte na něj, začínáte právě na tomto místě. Váš počítačový protivráč, si také nějaké vybere. Je možné, že si vyberete oba stejné, to je v pořádku.

Poté pokračujte pokud možno nejrychlejší cestou do červeně vyznačeného cíle. Po každém tahu se střídáte s protivráčem a snažíte se dojet jako první. Na konci hry vám bude oznámeno, na kolik tahů se vám trať podařilo projet. Můžete to zkusit znovu, na ještě menší počet tahů.



## 6 Taktika počítače

Jelikož programováním samotné taktiky počítače jsem strávil více než 95% celkového času věnovaného této hře, zaslouží si vlastní kapitolu. Opravdu velmi zjednodušeně zde shrnu základní pravidla podle kterých se počítač rozhoduje. Tím získáte nadhled na hru a kapitola Techniky, věnovaná samotným procedurám a funkcím pro vás bude mnohem pochopitelnější.

V prvním tahu se počítač rozhodne na jakém poli začne. To není nic těžkého. Přečte si celou strukturu tratě, a pak si jedno startovní pole čistě náhodně vybere.

V dalších tazích je nutné řešit spoustu aspektů a podle nich se zachovat. Nejprve se tedy rozhlédne jestli ve směru přímo nahoru, dolů, vpravo nebo vlevo není cíl. Pokud ano, poznamená si v jakém směru jej našel a jak daleko je.

At' už jej našel nebo ne, zjišťuje, jak daleko lze na kterou stranu dojet. To je obtížnější úkol, zatím vám jen prozradím, že pro každou stranu je nutné řešit to třikrát pro různé odchylky jízdy. Z těchto hodnot vybírá tu odchylku, kde je největší dojezd. Pokud je více stejných, vybírá náhodně některou z nich. Pro úplnost ještě prozradím, že zkoumá dojezd jen v těch směrech, kde má rychlost alespoň nulovou a také, že zkoumá dojezd v šikmém směru, ve kterém se zrovna jede, to je vhodné pro zatáčky a hlavně šikmé tratě.

„Alespoň nulová rychlost“ je trochu nestandardní výraz, proto jej uvedu na pravou míru:

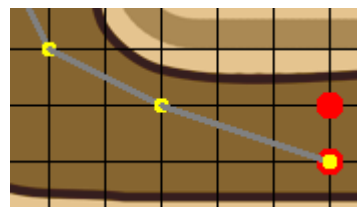


Na tomto obrázku je vidět, že počítač je rozjetý vpravo o 3 body. Ve svislém směru osy Y se nepohybuje. To znamená, že rychlost vpravo je 3, dolů 0, vlevo -3 a nahoru 0. Počítač tedy bude zkoumat jen směry vpravo, nahoru, dolů a samozřejmě také šikmo (díky odchylkám). Směr vlevo není třeba zkoumat.

Dále se počítač rozhoduje podle toho, jestli byl nalezen cíl. Pokud ano, je

směr pohybu jasný, v jaké odchylce se pohybovat už je také spočítáno. Pokud cíl nalezen nebyl, vybere směr s největší hodnotou možného dojezdu. Pokud je více možností se stejnou hodnotou, vybere náhodně jednu z nich. Jelikož to se stává relativně často, máme zaručeno, že počítač nepojede stále stejné tahy.

Teď už počítač ví, jakým směrem se pohybovat, ale neví o kolik. To závisí opět na tom, jestli už našel cíl nebo ne. V cíli není potřeba zastavit na nulovou rychlost, a tak pokud cíl našel, vybírá si takovou rychlost, aby konec tahu byl přesně v cíli. Pokud cíl nenašel, je nutné spočítat rychlost tak, aby se hlavně nevyboural.

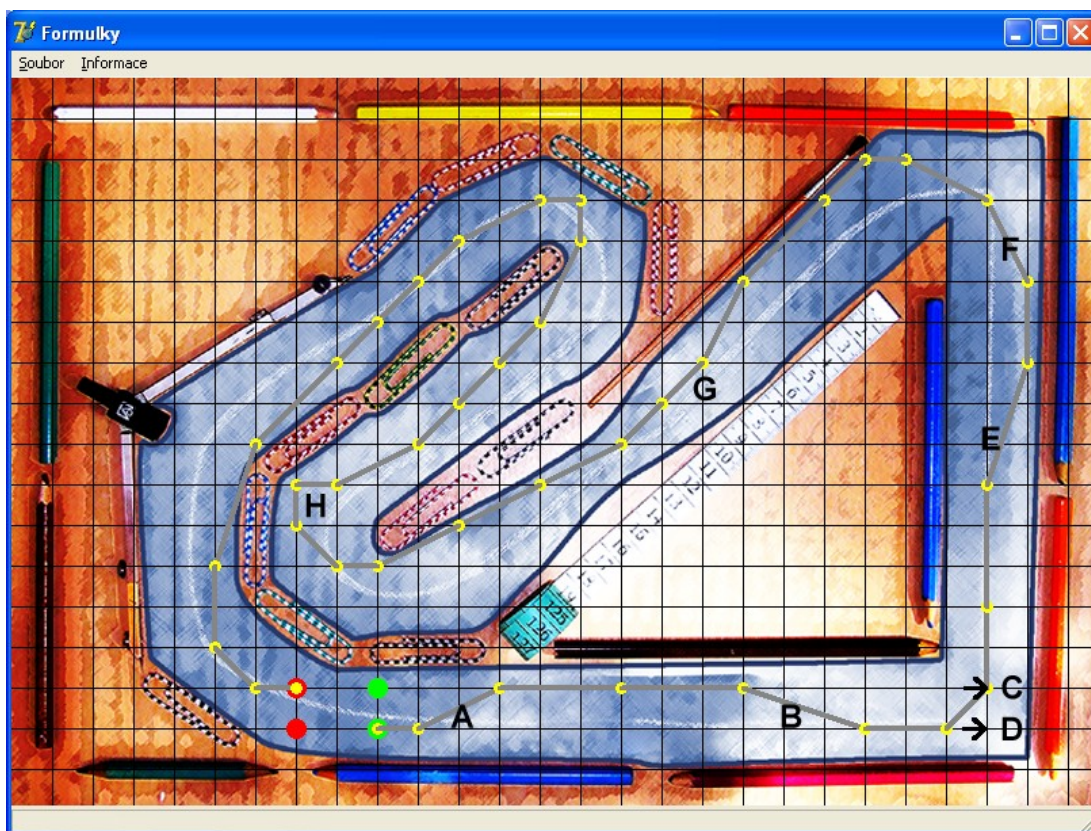


Jelikož pro popojíždění nefigurují jen směry, ale také odchylky, je možné, že pohybem, který si právě spočítal by přejížděl hranu. To by vedlo k vybourání, proto se tato možnost prozkoumá a dojezd se případně poupraví.

Nyní už počítač přesně ví, kam se posunout. Při samotném vykreslení pohybu je překontrolováno, zda se nevyboural. Pokud ano, místo vybourání je vyznačeno a jeho hra končí. To se ale stává velmi zřídka, především při doplňování nových funkcí hry, nebo testování extrémních tratí.

Pro lepší pochopení, názorně popíšu jednu hru počítače. Mapa, kterou jste doposud viděli, je opravdu jednoduchá, na ní bychom si toho moc nepopsali, proto vybírám obtížnější, kde už se počítač opravdu musí snažit:

Startovní bod je vybrán náhodně, proč se počítač dále vydá doprava, postupně zrychluje a před zatáčkou zpomalí, je asi také jasné. Možná trochu nelogické je, proč v bodě A vyjíždí o 1 nahoru a v bodě B opět sjíždí dolů. To je dáno tím, že zde ještě „nevidí“ za zatáčku a rozhoduje se podle toho, kde lze dojet dále. Jelikož k bodu C i D je stejný dojezd, rozhoduje se náhodně. To samé platí i v bodě E. Zajímavý je bod F, zde najíždění do zatáčky není způsobeno náhodou, ale tím, že zde už sleduje trať za zatáčkou a proto si chytře najíždí.



Asi vás udiví, proč v bodě G zpomaluje na rychlost  $[-1,1]$   $([X,Y])$ . Všimněte si, že trať je dále zúžena na minimální hodnotu. Zde při počítání nejdelších dojezdů zcela s přehledem vyhrává dojezd šikmo. Bohužel při menších rychlostech a úhlech jízdy  $45^\circ$  zde dochází k překryvům úhlů a díky tomu často k chybnému výběru rychlosti 1 jako nejvhodnější. Není zdaleka triviální tento problém vyřešit a jelikož nemá podstatnější vliv na funkčnost hry, ponechal jsem jej zatím neopraven.

Také by vás mohlo udivit zatočení do pravého úhlu v bodě H. To skutečně není chyba, počítač zde zvyšuje rychlost vpravo z 0 na 1 a snižuje rychlost nahoru z 1 na 0. To je podle pravidel zcela správně.

## 7 Techniky

Nyní se dostáváme do části, kde popíši, jak fungují některé funkce či procedury. Budu popisovat jen úryvky kódu, především těch delších a obtížnějších. Kvůli přehlednosti a omezenému místu také vynechám podobně formulované části daných funkcí nebo procedur.

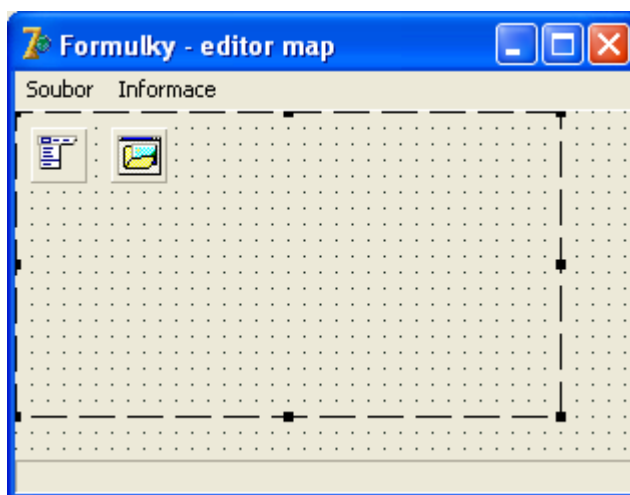
Nebudu popisovat ani ladící a různé záchytné části pro odhalování chyb. Pokud by vás přece jen zajímaly, najdete je ve zdrojovém kódu na přiloženém CD disku.

První kratší část bude o editoru, nejdelší část bude zabírat samotná hra.

### 7.1 Editor

#### 7.1.1 Hlavní formulář

Na hlavním formuláři najdeme Delphi prvky Image, Open Picture Dialog, Main Menu a Status Bar. Image má nastaveno Auto Size na True, naproti tomu velikost formuláře si kvůli chybě v této verzi Delphi řeším sám. Delphi



totiž nepočítají s velikostí Status Baru a tak se obrázek do okna nevejde a musí být zobrazen posuvník. S tím Delphi také nepočítají a tak se velikost použitelné části okna dále zmenšuje.

Status Bar používám pro různé informace uživateli. Také se v něm díky Auto Hint = True zobrazují informace k položkám menu apod., zapsané v proměnné Hint daného prvku.

##### 7.1.1.1 Aktualizuj okno

Změna velikost okna podle obrázku:

```
form1.Height:=image1.Height+74;
```

```
form1.Width:=image1.Width+10;
```

Dále je kontrolováno, zda okno programu nevybíhá mimo obrazovku. Pokud ano, je vycentrováno. Pokud je ale větší než obrazovka, je posunuto vlevo nahoru tak, aby byla vidět jeho co největší část. Ukázka pro horizontální kontrolu:

```
if form1.Left+form1.Width > screen.Width then
begin
rozdil:=screen.Width-form1.Width;
if rozdil>0 then form1.left:=rozdil div 2
else form1.Left:=0;
end;
```

### **7.1.1.2 Otevřít obrázek**

Otevře dialog pro výběr obrázku, zobrazí jej v okně programu a informuje aplikaci o úspěšném nahrání obrázku.

```
obrazek_nastaven:=false;
if Openpicturedialog1.execute then
begin
Obrazek:=TPicture.Create;
Obrazek.LoadFromFile(OpenPictureDialog1.filename);
Image1.Picture.Bitmap.Assign(Obrazek.graphic);
Obrazek.Free;
obrazek_nastaven:=true;
end;
```

### **7.1.1.3 Vytvoření nové mapy**

Tato procedura se spustí při kliknutí v menu Soubor → Vytvořit mapu. Zavolá procedury Otevřít obrázek a Aktualizuj okno. Do proměnných zapisuje, co už je nastaveno, maže přednastavený název pro uložení a dále volá Nastavení mřížky.

```
pole_pripraveno:=false;
OtevirObrazek;
if obrazek_nastaven then
begin
AktualizujOkno;
FormSaveAs.Edit1.Text:="";
Nastavitmrizku1.Enabled:=true;
form1.StatusBar1.Panels[0].Text:='Nastavte mrizku';
Nastavitmrizku1Click(self);
end;
```

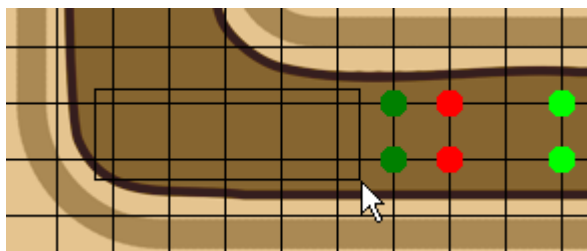
#### 7.1.1.4 Stisknutí myši

Po kliknutí myši si uložíme polohu kurzoru v okně. To potřebujeme pro orámování více polí.

```
if pole_pripraveno=false then exit;  
start_x:=x;  
start_y:=y;
```

#### 7.1.1.5 Pohyb myši

Zajišťuje právě vykreslení orámování vybraného myši. Orámování funguje při stisknutí jakéhokoliv tlačítka myši.



```
if pole_pripraveno=false then exit;  
if (ssleft in Shift) or (ssRight in Shift) or (ssMiddle in Shift) then  
begin  
image1.Repaint;  
canvas.Brush.Style:=bsClear;  
Canvas.Rectangle(start_x,start_y,x,y);  
end;
```

#### 7.1.1.6 Uvolnění myši

Podle toho, jestli se orámoval větší úsek, nebo se jen kliklo na jedno místo, posílá informace proceduře Označ pole. Při orámování posílá umístění všech polí uvnitř orámování. Pokud se jen klikne, posílá umístění nejbližšího bodu. Díky tomu, při definování mapy, nemusíme dávat pozor na přesnost, to zařídí počítač za nás.

Pokud se jen kliklo:

```
if (x=start_x) and (y=start_y) then  
begin  
druhe_x:=(x + mrizka div 2) div mrizka;  
druhe_y:=(y + mrizka div 2) div mrizka;  
OznacPole(druhe_x,druhe_y,Button);  
end
```

Pokud byla orámována větší oblast, zjistíme nejdříve pozici levého horního bodu orámování. Od bodu nejvíce nahoře vlevo najdeme nejbližší mřížkový průsečík směrem doprava dolů,

```
for z:=x1 to x1+mrizka do if z mod mrizka = 0 then prvni_x:=z div mrizka;  
for z:=y1 to y1+mrizka do if z mod mrizka = 0 then prvni_y:=z div mrizka;
```

dále posíláme pozice orámovaných bodů proceduře Označ pole.

```
druhe_x := x2 div mrizka;  
druhe_y := y2 div mrizka;  
  
for i:=prvni_x to druhe_x do  
  for j:=prvni_y to druhe_y do  
    OznacPole(i,j,Button);
```

### 7.1.1.7 Označ pole

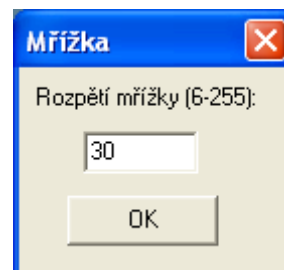
Zapíše do dvourozměrného pole změnu daného bodu, zároveň daný bod vykreslí. Co zapíše a vykreslí záleží na stisknutém tlačítku myši. Velikost vykreslených bodů se počítá automaticky, podle velikosti mřížky.

Na ukázkou kód pro označení startovního pole pravým tlačítkem myši:

```
if Button=mbRight then  
  begin  
    pole[x,y]:='s';  
    Image1.Canvas.pen.Width:=mrizka div 2;  
    Image1.Canvas.pen.Color:=clLime; {svitiva zelena}  
    Image1.Canvas.MoveTo(x*mrizka,y*mrizka);  
    Image1.Canvas.LineTo(x*mrizka,y*mrizka);  
  end;
```

## 7.1.2 Formulář pro nastavení mřížky

Na tomto formuláři jsou jen Delphi prvky Label, Edit a Button. V prvku Label je dostatečně jasně napsáno, jaké hodnoty lze zadávat. Číselný rozsah však není omezen, a tak je možné zvolit si i menší či větší hodnoty dle potřeby. Pokud se uživatel rozhodne psát jiné znaky než čísla, bude mu odměnou anglická hláška Delphi.



### 7.1.2.1 Nastavení mřížky

Tato procedura se spustí po stisknutí tlačítka OK. Vykreslí přes obrázek mřížku (ukázka jen vertikálně),

```
form1.mrizka:=strtoint(FormMrizka.edit1.text);  
  
for i:=1 to form1.Image1.Width div form1.mrizka do  
  begin  
    form1.Image1.Canvas.MoveTo(i*form1.mrizka,0);  
    form1.Image1.Canvas.LineTo(i*form1.mrizka,form1.Image1.Height);  
  end;
```

dynamicky přiřadí rozsah poli pro mapu,

```
i:=form1.image1.Width div form1.mrizka + 1;  
j:=form1.image1.Height div form1.mrizka + 1;  
SetLength(form1.pole,i,j);
```

naplní ji nulami,

```
for j:=0 to form1.Image1.Height div form1.mrizka do  
for i:=0 to form1.Image1.Width div form1.mrizka do  
form1.pole[i,j]:='0';
```

zařídí nutnou signalizaci a ve stavovém panelu zobrazí informace uživateli.

```
form1.pole_pripraveno:=true;  
form1.Ulozit.Enabled:=true;  
form1.UlozitJako.Enabled:=true;  
form1.StatusBar1.Panels[0].Text:='Pomocí myši nastavte kde je pole. Použijte všechna tlačítka myši.  
Trať - zeleně, start - sv. zeleně, cíl - sv. červeně.';
```

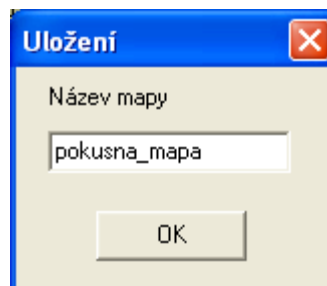
### 7.1.2.2 Zavření formuláře

Pokud byl formulář zavřen bez nastavení mřížky, bude uživatel informován, co je potřeba udělat.

```
if form1.pole_pripraveno=false then  
begin  
MessageDlg('Mřížka nebyla nastavena. Než budete vytvářet mapu, musíte ji nastavit.',  
mtWarning, mbOkCancel, 0);  
end;
```

### 7.1.3 Formulář pro ukládání

Stejně jako při nastavování mřížky, zde najdete Delphi prvky Label, Edit a Button.



#### 7.1.3.1 Stisknutí tlačítka OK

Je prozkoumáno, jestli už taková mapa existuje. Pokud ano, uživatel je dotázán, zda původní mapu přepsat.

```
nazev:=Edit1.Text;  
  
if FileExists('\mapy\' + nazev + '.bmp') then  
begin  
if MessageDlg('Taková mapa již existuje, chcete ji přepsat?', mtInformation, mbOkCancel, 0) = 2  
then exit;  
end
```

Pokud mapa s daným názvem ještě není, je připsána do konfigurace.

```
AssignFile(prg_mapy, '\data\mapy.cfg');  
Append(prg_mapy);  
writeln(prg_mapy, nazev);  
CloseFile(prg_mapy);
```

Na závěr zavoláme proceduru Ulož mapu.

```
UlozMapu(nazev);
```

### 7.1.3.2 Ulož mapu

V této proceduře už není potřeba řešit jestli dané soubory existují, vše už je vyřešeno předchozí procedurou. Proč je toto samostatná procedura a ne součást předchozí, je velmi prosté. Pokud už máme nastaveno jméno mapy a postupně si ji ukládáme, abychom nepřišli o data, není potřeba neustále vypisovat jméno mapy a být dotazováni, zda chceme původní přepsat. Je tudíž už přímo volána tato procedura.

V menu hlavního programu najdete položky „Uložit“ a „Uložit jako“. Při kliknutí na „Uložit jako“ vždy zadáváte jméno mapy, při kliknutí na „Uložit“ jste na jméno dotázáni, jen pokud jste jej dosud nezadali.

Na ukázkou uložení obrázku

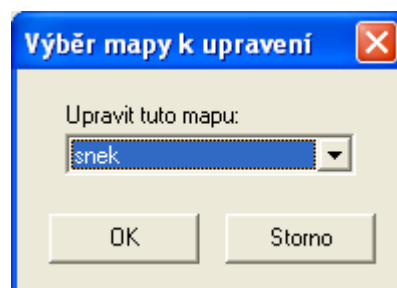
```
Obrazek:=TPicture.Create;  
Obrazek.LoadFromFile(form1.OpenPictureDialog1.filename);  
Obrazek.SaveToFile('.\mapy\' + nazev + '.bmp');
```

a pole.

```
AssignFile(mapa_pole, '.\mapy\' + nazev + '.pole');  
ReWrite(mapa_pole);  
for j:=0 to form1.Image1.Height div form1.mrizka do  
begin  
for i:=0 to form1.Image1.Width div form1.mrizka do  
write(mapa_pole, form1.pole[i,j]);  
writeln(mapa_pole);  
end;  
CloseFile(mapa_pole);
```

### 7.1.4 Formulář pro výběr mapy

Pokud budete chtít upravit již existující mapu, bude vám nápomocen právě tento formulář. Použity jsou zde prvky Label, ComboBox a Button. ComboBox má styl nastaven na csDropDownList, což oproti výchozímu csDropDown nedovolí uživateli zapisovat jiné než dané hodnoty.



#### 7.1.4.1 Zobrazení formuláře

Ihned při zobrazení formuláře jsou do výběru ComboBox načteny ze souboru ./data/mapy.cfg všechny mapy, které jsou k dispozici.

```
ComboBox1.Items.Clear;
AssignFile(file_mapy, '\data\mapy.cfg');
Reset(file_mapy);
while not Eof(file_mapy) do
  begin
    readln(file_mapy, text);
    ComboBox1.Items.Add(text);
  end;
CloseFile(file_mapy);
```

#### 7.1.4.2 Stisknutí tlačítka OK

Po kliknutí na tlačítko OK je zkontrolováno, zda byla nějaká mapa vybrána.

```
mapa:=ComboBox1.Text;
if mapa="" then
  begin
    MessageDlg('Musíte si některou mapu vybrat.', mtInformation, [mbOk], 0);
    exit;
  end;
```

Načítají se data ze souborů dané mapy a podle nich je přiřazena paměť, naplněno pole,

```
AssignFile(mapa_pole, '\mapy\' + mapa + '.pole');
Reset(mapa_pole);
for j:=0 to form1.Image1.Height div form1.mrizka do
  begin
    for i:=0 to form1.Image1.Width div form1.mrizka do
      read(mapa_pole, form1.pole[i, j]);
    readln(mapa_pole);
  end;
CloseFile(mapa_pole);
```

vykreslena mřížka a nastavení bodů.

```
form1.Image1.Canvas.pen.Width:=form1.mrizka div 2;
for j:=0 to form1.Image1.Height div form1.mrizka do
  begin
    for i:=0 to form1.Image1.Width div form1.mrizka do
      if form1.pole[i, j]='1' then {klasicke pole}
        begin
          form1.Image1.Canvas.pen.Color:=clGreen; {svitiva cervena}
          form1.Image1.Canvas.MoveTo(i*form1.mrizka, j*form1.mrizka);
          form1.Image1.Canvas.LineTo(i*form1.mrizka, j*form1.mrizka);
        end
      end;
```

Dále se načítá obrázek a je volána procedura Aktualizuj Okno.

```
form1.Image1.Picture.LoadFromFile('\mapy\' + mapa + '.bmp');
form1.AktualizujOkno;
```

Aby bylo možné ukládat přímo, nastavuje se jméno mapy.

```
FormSaveAs.edit1.Text:=mapa;  
form1.OpenPictureDialog1.FileName:='.\mapy\'+'mapa+'.bmp';
```

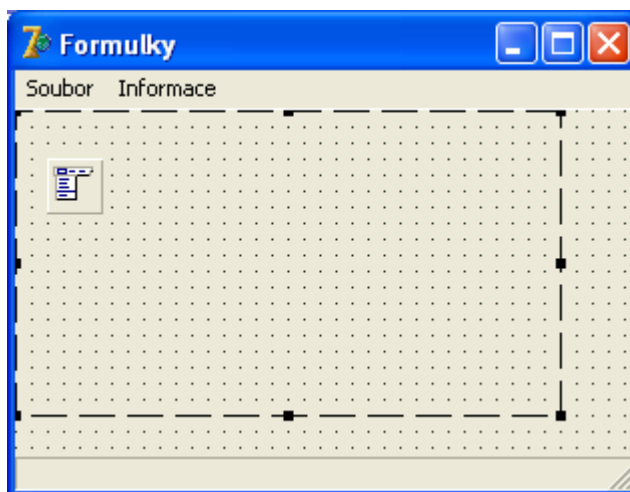
Nakonec jsou uloženy nové hodnoty proměnných nutných pro signalizaci a zviditelnění položek v menu. Do panelu jsou vypsány nejdůležitější informace pro uživatele.

```
form1.obrazek_nastaven:=true;  
form1.Nastavitmrizku1.Enabled:=false;  
form1.pole_pripraveno:=true;  
form1.Ulozit.Enabled:=true;  
form1.UlozitJako.Enabled:=true;  
form1.StatusBar1.Panels[0].Text:='Pomocí myši upravte pole. Použijte všechna tlačítka myši.  
Trať - zeleně, start - sv. zeleně, cíl - sv. červeně.';
```

## 7.2 Hra

### 7.2.1 Hlavní formulář

Stejně jako v editoru zde najdeme prvky Image, MainMenu a StatusBar. Chybí jen OpenPictureDialog, který zde není potřeba, jelikož načítání obrázku řeším sám podle vybrané mapy. Nastavení prvků je velmi podobné jako v případě editoru.



#### 7.2.1.1 Vytvoření formuláře

Spustí se při zapnutí hry, z konfigurace zjistí, jaká mapa je výchozí. Načte příslušný obrázek, nastavení mřížky a pole. Přiřadí paměť pro pole, načte je a vykreslí mřížku. Dále vykreslí startovní a cílové pole a inicializuje proměnné do výchozích hodnot:

```
clovek_bodu:=0;  
pocitac_bodu:=0;  
clovek_dojeto:=false;  
pocitac_dojeto:=false;  
randomize;
```

Nakonec náhodně určí, kdo bude začínat.

### 7.2.1.2 *Tah počítače*

Tato procedura rozhoduje, jaký další tah bude pro počítač ten nejlepší. Jelikož to není jednoduché rozhodování, většinu problémů rozděljuje na podproblémy a ty deleguje na další funkce a procedury. Ty jej přímo řeší, nebo často rozdělují do dalších menších částí a delegují je dále.

Při prvním tahu se zjišťuje, kde jsou umístěna startovní pole a jedno z nich se náhodně vybírá.

```
for j:=1 to form1.Image1.Height div form1.mrizka do
  for i:=1 to form1.Image1.Width div form1.mrizka do
    if pole[i,j]='s' then {nasel startove pole}
      begin
        inc(cislo);
        starty[cislo].x:=i;
        starty[cislo].y:=j;
      end;
  cislo:=random(cislo)+1;
  pozice_pocitace_x:=starty[cislo].x;
  pozice_pocitace_y:=starty[cislo].y;
  rychlost_pocitace_x:=0;
  rychlost_pocitace_y:=0;
  pohyb('pocitac', pozice_pocitace_x, pozice_pocitace_y, pozice_pocitace_x, pozice_pocitace_y,
  rychlost_pocitace_x, rychlost_pocitace_y);
```

Při jakémkoliv dalším tahu počítač hledá cíl ve všech směrech kde má rychlost alespoň nulovou. Pokud cíl nalezne, poznačí si jakým směrem a jak daleko cíl je. V šikmém směru se cíl prozatím nehledá. Ukázka hledání cíle směrem nahoru:

```
if (rychlost_pocitace_y<=0) and (HledejCil('nahoru', pozice_pocitace_x, pozice_pocitace_y,
rychlost_pocitace_x, rychlost_pocitace_y)>0) then
  begin
    cil_nalezen:=true;
    s_do_cile:=HledejCil('nahoru', pozice_pocitace_x, pozice_pocitace_y, rychlost_pocitace_x,
    rychlost_pocitace_y);
    smer_cile:='nahoru';
  end
```

Dále hledá nejdelší možný dojezd šikmo a trojcestně na všechny strany. Procedura Šikmý dojezd přímo odpovídá, na jaké pole dále jet a jaký dojezd je v takovém směru možný. Trojitý možný dojezd odpovídá s jakou odchylkou je nejlepší v daném směru jet a jaký je dojezd je možný zde. Zkoumá se vždy šikmý dojezd a strany, kde je rychlost alespoň nulová. Ukázka pro šikmý pohyb a pohyb nahoru:

```
SikmyDojezd(pozice_pocitace_x, pozice_pocitace_y, rychlost_pocitace_x, rychlost_pocitace_y,  
s_sikmo, sikmo_x, sikmo_y);
```

```
if rychlost_pocitace_y<=0 then TrojityMoznyDojezd('nahoru', pozice_pocitace_x, pozice_pocitace_y,  
rychlost_pocitace_x, rychlost_pocitace_y, s_nahoru, odchylka_nahoru);
```

Pokud cíl nalezen nebyl, vybírá místo kam jet podle největšího možného dojezdu. Nejprve zjistí nejvyšší hodnotu dojezdu,

```
s_nejvetsi:=s_nahoru;  
if s_dolu>s_nejvetsi then s_nejvetsi:=s_dolu;  
if s_doleva>s_nejvetsi then s_nejvetsi:=s_doleva;  
if s_doprava>s_nejvetsi then s_nejvetsi:=s_doprava;  
if s_sikmo>s_nejvetsi then s_nejvetsi:=s_sikmo;
```

poté zjišťuje, kolikrát se tato hodnota vyskytuje a označí si u jakých směrů. Jak můžete vidět, je to řešeno opravdu nestandardním způsobem, ale myslím si, že takto je to rozumně přehledné. Ukázka pro směr nahoru a dolů:

```
maximalnich:=0;  
if s_nahoru=s_nejvetsi then  
begin  
inc(maximalnich);  
max[maximalnich]:='nahoru';  
end;  
if s_dolu=s_nejvetsi then  
begin  
inc(maximalnich);  
max[maximalnich]:='dolu';  
end;
```

K předchozímu patří ještě porovnání s šikmým dojezdem, v něm je ale ještě pár nepřesností způsobených překryvem úhlů, a proto se šikmý pohyb používá, jen když je u něj dojezd vyšší než u všech ostatních. Až bude šikmý pohyb dokonale vyladěný, je možné, že mu přiřadím úplně nejvyšší prioritu.

```
if s_sikmo=s_nejvetsi then  
begin  
if maximalnich=0 then  
begin  
inc(maximalnich);  
max[maximalnich]:='sikmo';  
end;  
end;
```

Následně je náhodně vybrán jeden ze směrů, kde je nejvyšší dojezd.

Pokud cíl nalezen byl, vybírá se právě jeho směr.

Dále se provádí tah, podle toho, který směr byl vybrán. Směr šikmo má jinou strukturu, než ostatní. Díky tomu je zde jeho zápis jednodušší.

```
if max[nahoda]='sikmo' then  
Pohyb('pocitac', pozice_pocitace_x, pozice_pocitace_y, pozice_pocitace_x + sikmo_x,  
pozice_pocitace_y + sikmo_y, rychlost_pocitace_x, rychlost_pocitace_y)
```

U ostatních směrů je navíc potřeba také řešit, jestli byl nalezen cíl. Pokud ano, vybírá se nejvyšší rychlost, jakou by mohl počítač pokračovat, aby se trefil na bod cíle. Pokud cíl není v dosahu, vybírá se rychlost, tak aby byl schopen dobrzdit v zatáčkách. Řešení jízdy přes hranu upravuje cílové pole, pokud by se aktuálně vybranou rychlostí přejížděl nějaký roh. Po těchto náročných úkonech už se konečně vybraný tah pošle proceduře Pohyb, která zajistí vše potřebné. Na ukázkou jen směr nahoru:

```

if max[nahoda]='nahoru' then
  begin
    if cil_nalezen=true then
      begin
        if LzeJetDoCile(s_do_cile, abs(rychlost_pocitace_y-1)) then
          popojet:=abs(rychlost_pocitace_y-1)
        else if LzeJetDoCile(s_do_cile,abs(rychlost_pocitace_y)) then
          popojet:=abs(rychlost_pocitace_y)
        else popojet:=abs(rychlost_pocitace_y+1);
        end
      else popojet:=OKolikPopojet(s_nahoru,-rychlost_pocitace_y);
      ReseniJizdyPresHranu(pozice_pocitace_x, pozice_pocitace_y, pozice_pocitace_x +
        rychlost_pocitace_x + odchylka_nahoru, pozice_pocitace_y - popojet, 'nahoru',
        -rychlost_pocitace_y, popojet);
      Pohyb('pocitac', pozice_pocitace_x, pozice_pocitace_y, pozice_pocitace_x + rychlost_pocitace_x
        + odchylka_nahoru, pozice_pocitace_y - popojet, rychlost_pocitace_x, rychlost_pocitace_y);
      end
    end
  end

```

Nesmíme zapomenou zvýšit počet tahů. Dále je řada na tahu člověka. Případně další tah počítače, pokud už člověk dohrál a nechává počítač dokončit hru také.

```

inc(pocitac_bodu);
if (clovek_dojeto) and (pocitac_dojeto=false) then
  begin
    Sleep(10);
    image1.Repaint;
    TahPocitace;
  end;

```

### 7.2.1.3 Řešení jízdy přes hranu

Tato procedura upravuje dojezd, pokud by se přejížděl roh, nebo úzká hrana. Pomocí proměnné popojet, vrací přímo opravené popojetí.

```

if (Pruchozi(zacatek_x, zacatek_y, konec_x, konec_y)=false)
and (popojet>=rychlost_timto_smerem) then
  begin
    popojet:=popojet-1;
    if smer='nahoru' then konec_y:=konec_y+1
    else if smer='dolu' then konec_y:=konec_y-1
    else if smer='doleva' then konec_x:=konec_x+1
    else if smer='doprava' then konec_x:=konec_x-1;
  end;

```

#### 7.2.1.4 Pohyb

Tato procedura především vykresluje samotný pohyb,

```
Image1.Canvas.pen.Width:=(mrizka div 10)+1;  
if kdo_jede='clovek' then Image1.Canvas.pen.Color:=clBlack {cerna}  
else Image1.Canvas.pen.Color:=clDkGray; {tmava seda}  
Image1.Canvas.MoveTo(odkud_x*mrizka,odkud_y*mrizka);  
Image1.Canvas.LineTo(kam_x*mrizka,kam_y*mrizka);
```

nastavuje novou rychlost a pozici hráče.

```
rychlost_x:=kam_x-odkud_x;  
rychlost_y:=kam_y-odkud_y;
```

```
odkud_x:=kam_x;  
odkud_y:=kam_y;
```

Samozřejmě také vykreslí „autíčko“

```
Image1.Canvas.pen.Width:=(mrizka div 5)+3;  
if kdo_jede='clovek' then Image1.Canvas.pen.Color:=clBlue {modra}  
else Image1.Canvas.pen.Color:=clYellow {do hneda};  
Image1.Canvas.MoveTo(kam_x*mrizka,kam_y*mrizka);  
Image1.Canvas.LineTo(kam_x*mrizka,kam_y*mrizka);
```

a informuje o dojetí do cíle.

```
if pole[kam_x,kam_y]='c' then  
if kdo_jede='clovek' then  
begin  
MessageDlg('Cíl! Zvládl jste to na'+inttostr(clovek_bodu)+' tahů.',mtInformation,[mbOk],0);  
clovek_dojeto:=true;  
end
```

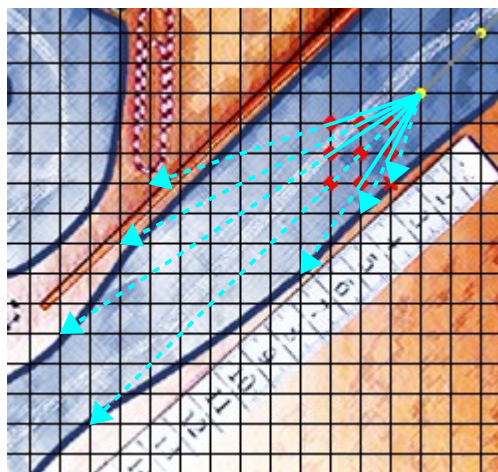
#### 7.2.1.5 Šikmý dojezd

Nyní se dostáváme k problematické proceduře pro šikmý dojezd. Zastupuje procedury Dojezd a Trojitý možný dojezd pro šikmé úseky. Jejím cílem je především najít nejlepší směr a spočítat rychlost, kterou bude vhodné v tomto směru jet.

Na obrázku je znázorněno, jak je sledováno, kam by se dále mohlo jet.

Šipky jsou vykresleny až do prvního bodu nárazu. Všimněte si, že směrem nejdelší šipky jsou v zákrytu 3 body. To rozhodování počítače dále ztěžuje.

Nejprve se projdou všechna pole, kam lze jet a zjistí se, jaké jsou dojezdy:



```

for j:=-1 to 1 do
for i:=-1 to 1 do
  begin
  a:=0; kam_x:=0; kam_y:=0;
  if (v_x+i=0) and (v_y+j=0) then dojezd[i,j]:=0
  else repeat
    if poz_x+(a+1)*v_x+(a+1)*i>=0 then kam_x:=poz_x+(a+1)*v_x+(a+1)*i
    else break;
    if poz_y+(a+1)*v_y+(a+1)*j>=0 then kam_y:=poz_y+(a+1)*v_y+(a+1)*j
    else break;
    if Pruchozi(poz_x,poz_y,kam_x,kam_y) then a:=a+1
    else break;
  until a>534;
  if a=0 then velikost_delsi_strany:=0 else velikost_delsi_strany:=max(abs(v_x+i),abs(v_y+j));
  dojezd[i,j]:=a*velikost_delsi_strany;
  if LzeJetToutoRychlosti(a*velikost_delsi_strany,velikost_delsi_strany)=false then dojezd[i,j]:=0;
  end;

```

Najdeme nejvyšší hodnotu:

```

for j:=-1 to 1 do
  for i:=-1 to 1 do
    if dojezd[i,j]>max_dojezd then max_dojezd:=dojezd[i,j];

```

Nyní je potřeba vybrat tu s největším dojezdem. Na začátku jsem sliboval, že ukážu stejné věci různě naprogramované. Vybírání z nejvyšších a náhodný výběr už jsme prováděli v Tahu počítače. Tam to byl výběr z 5ti hodnot a zabralo nám to 31 řádků kódu. Zde vybíráme z devíti hodnot a tudíž by to odpovídalo zhruba 56 řádkům kódu. Podařilo se mi to ale napsat na řádků 10 a dokonce je to přehlednější než předchozí.

Vyřešení předchozí verze trvá počítači vždy stejnou dobu, naproti tomu zde je to dáno náhodou a při větším počtu nejvyšších hodnot výkonnost stoupá. Řešení to ale rozhodně není příliš čisté. Nechal jsem je zde jen kvůli ukázkovým důvodům a určitě by se vyplatilo napsat pro tento problém samostatnou funkci, která by dokázala řešit všechny takové případy.

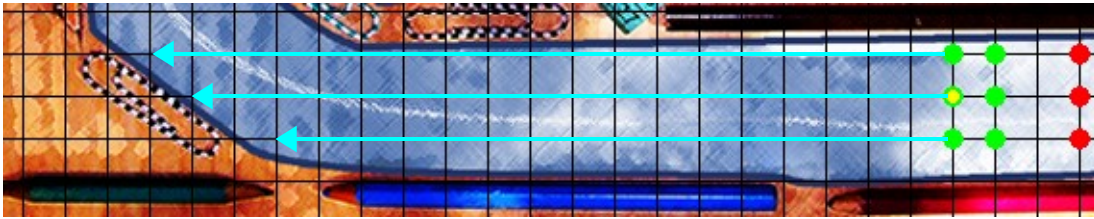
```

repeat
  i:=random(3)-1;
  j:=random(3)-1;
  if dojezd[i,j]=max_dojezd then
    begin
      jed_x:=v_x+i;
      jed_y:=v_y+j;
      break;
    end;
until false;

```

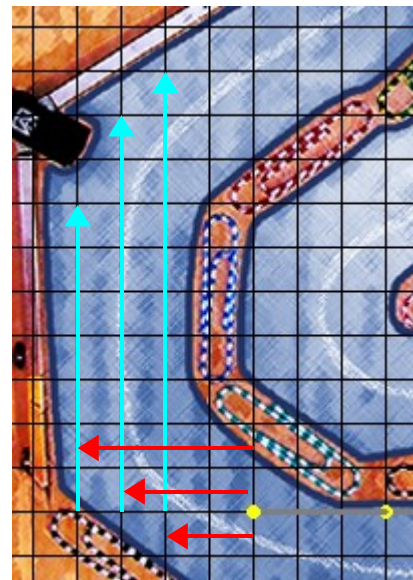
### 7.2.1.6 *Trojité možný dojezd*

Jak název napovídá, bude se hledat dojezd v 3 různých posunech. Abyste si to dokázali představit, použijte obrázek:



Počítač je na startu a rozhoduje se ve všech směrech, kam pojede. To se řeší v proceduře Tah počítače, této proceduře už přichází dotazy na konkrétní směr. Na tomto obrázku vidíte grafické znázornění dotazu na směr vlevo. Teď už je snad zřejmé, proč je to řešeno právě ve 3 různých posunech. Přestože jedeme vlevo, je totiž možné přidat nebo ubrat na rychlosti ve směru Y. Asi nikoho nepřekvapí, že počítač dále pojede na šipku nejvíce nahoře.

Ale je tu ještě jeden problém, je potřeba započítávat aktuální rychlost v opačné ose. Světle modré šipky na tomto obrázku znázorňují zjištění pohybu nahoru, když rychlost vlevo je 3. Prostřední ze šipek se právě proto posunuje o 3 body mřížky vlevo, zbylé šipky o 1 respektive -1 vůči prostřední šipce.



Červené šipky pro úplnost ukazují, že opravdu závisí jen na rychlosti v opačné ose. Jelikož rychlost v ose Y je nulová, žádný posun u nich nenastane. Situace se samozřejmě dále ztěžuje, pokud je rychlost v obou osách nenulová.

Nyní už samotná procedura. Nejprve se projdou všechny 3 šipky, pokud výsledek funkce Můžu k šipce pozitivní, jsou započítány:

```

if (smer='nahoru') or (smer='dolu') then
  begin
    for i:=-1 to 1 do
      begin
        if pozice_x+rychlost_x+i<0 then dojezd[i]:=-1
        else begin
          x_sipky:=pozice_x+rychlost_x+i;
          if MuzuKSipce(pozice_x,pozice_y,x_sipky,pozice_y)
            then dojezd[i]:=MoznyDojezd(smer,x_sipky,pozice_y) else dojezd[i]:=-1;
          end;
        end;
      end;
    end
  end

```

Najde se nejvyšší hodnota a je uložena, případně vybrána jedna náhodně

z více stejných. Toto je třetí způsob na ukázkou, jak je možné jej naprogramovat.

```
if (dojezd[-1]=max_dojezd) and (dojezd[0]=max_dojezd) and (dojezd[1]=max_dojezd) then
  begin
    x:=random(3);
    case x of
      0:odchylka:=-1;
      1:odchylka:=0;
      2:odchylka:=1;
    end;
  end

else if (dojezd[-1]=max_dojezd) and (dojezd[0]=max_dojezd) then
  begin
    x:=random(2);
    case x of
      0:odchylka:=-1;
      1:odchylka:=0;
    end;
  end

else if dojezd[-1]=max_dojezd then odchylka:=-1
else if dojezd[0]=max_dojezd then odchylka:=0
else if dojezd[1]=max_dojezd then odchylka:=1
```

### 7.2.1.7 Možný dojezd

Pro zjednodušení Trojitého možného dojezdu, zjišťuje jak daleko lze dojet daným směrem. Na ukázkou jen směr nahoru.

```
if smer='nahoru' then
  begin
    while (pole[pozice_x,pozice_y-dojezd]='1') or (pole[pozice_x,pozice_y-dojezd]='s')
    or (pole[pozice_x, pozice_y-dojezd]='c') do
      begin
        inc(dojezd);
      end;
    if dojezd<>0 then Dec(dojezd);
  end
```

### 7.2.1.8 Hledej cíl

Tato funkce hledá cíl daným směrem. Zpátky vrátí, kolik míst je vzdálen. V následující části kódu je vidět, jak řeší posuny známé z Trojitého možného pohybu

```
if (smer='doleva') or (smer='doprava') then
  begin
    x:=poz_x;
    y:=poz_y+v_y;
  end
```

a následně samotné hledání cíle.

```

while (pole[x,y]<>'0') and (pole[x,y]<>'c') do
  begin
    if smer='doleva' then Dec(x)
    else if smer='doprava' then Inc(x)
    else if smer='nahoru' then Dec(y)
    else if smer='dolu' then Inc(y);
    inc(dojezd);
  end;
if pole[x,y]='c' then
  begin
    HledejCil:=abs(dojezd);
  end;

```

### 7.2.1.9 O kolik popojet

Tato funkce spočítá z maximálního dojezdu a aktuální rychlosti, jaká rychlost bude dále vhodná.

Nejprve zjistí jaká je maximální dosažitelná rychlost, tak aby se ještě dalo dobrzdit

```

dojezd_kde_ubrzdim:=0;
max_mozne_popojeti:=0;
while dojezd_kde_ubrzdim + max_mozne_popojeti + 1<=max_dojezd do
  begin
    inc(max_mozne_popojeti);
    dojezd_kde_ubrzdim:=dojezd_kde_ubrzdim + max_mozne_popojeti;
  end;

```

a podle výsledků spočítá rychlost odpovídající možnostem zrychlení nebo zpomalení.

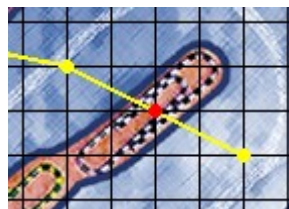
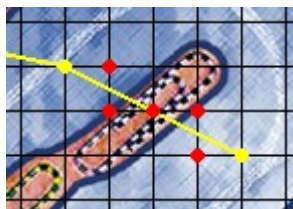
```

if max_mozne_popojeti>rychlost_v_tomto_smeru then OKolikPopojet:=rychlost_v_tomto_smeru+1
else if max_mozne_popojeti=rychlost_v_tomto_smeru then OKolikPopojet:=rychlost_v_tomto_smeru
else if max_mozne_popojeti<rychlost_v_tomto_smeru then OKolikPopojet :=
rychlost_v_tomto_smeru-1

```

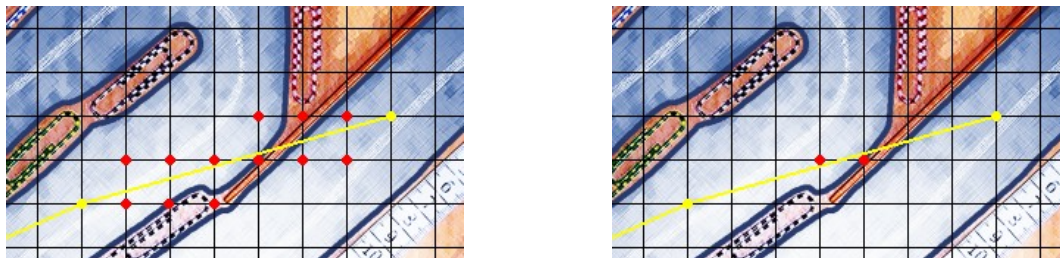
### 7.2.1.10 Průchozí

Pro zjištění správnosti tahu nestačí zjišťovat, jestli počáteční a koncové pole není mimo trať, je nutné také zjistit jestli se neprojíždí nějaký blok nebo hrana mimo oblast hry. Pro vysvětlení použijí následující obrázky:



První obrázek představuje kontrolu ve vertikálním směru osy Y, druhý potom v horizontálním směru osy X. Je vždy potřeba zkontrolovat jestli

alespoň jedno z dvojice políček je umístěno na trati. Kde průjezd zrovna protíná souřadné pole, vzniká samozřejmě právě jedno políčko, které musí ležet na trati. V druhém obrázku je kontrola jen jedna, neboť začáteční a koncový bod mají mezi sebou od jen jeden.



Na obrázcích výše, můžete vidět, jak vypadá špatně udělaná mapa. Ohraničení je příliš úzké a tak je vyhodnocen tento pohyb jako správný. V takových případech je nutné mapu předělat, nebo zvolit jinou mřížku.

Nejprve se zjišťuje jestli je konečné pole na trase:

```
if (kam_y>high(pole[0])) or (kam_x>high(pole)) or (pole[kam_x,kam_y]='0') then
  begin
    pruchozi:=False;
    exit;
  end;
```

Dále se počítá správnost pohybu při úhlu  $0^\circ$  respektive  $90^\circ$  pro daný kvadrant a poté pro úhel  $45^\circ$ . Pro jakékoliv jiné úhly se počítá průchodnost následujícím obecným postupem.

Zjištění pozice bodu více vlevo.

```
if odkud_x<=kam_x then
  begin
    nizsi_x:=odkud_x;
    nizsi_y:=odkud_y;
  end
else begin
  nizsi_x:=kam_x;
  nizsi_y:=kam_y;
end;
```

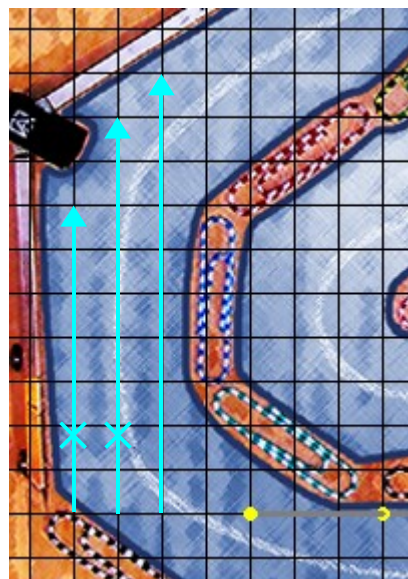
Postupné procházení bodů:

```
tg_alfa:=posun_y/posun_x;
for i:=0 to abs(posun_x) do
  begin
    posun1:=floor(i*tg_alfa);
    posun2:=ceil(i*tg_alfa);
    if (pole[nizsi_x+i,nizsi_y+posun1]='0') and (pole[nizsi_x+i,nizsi_y+posun2]='0') then
      begin
        pruchozi:=false;
        exit;
      end;
  end;
```

### 7.2.1.11 *Můžu k šipce*

Tato funkce zjišťuje průjezdnost k začátku šipky. Dále také, jestli by při popojetí na dané místo, bylo možné během následujících tahů dobrzdit.

V tomto obrázku vidíte oba případy. K levé šipce nelze vůbec dojet, neboť začíná mimo trať. K prostřední šipce dojet lze, ale dále by nebylo možné jízdu ubrzdit a nutně by se narazilo.



Pro ukázkou jen kód pro směr osy Y.

Nejprve zjišťuje vhodnost rychlosti, kdyby se popojelo na začátek šipky.

```
if (odkud_x-sipka_x=0) and (odkud_y-sipka_y=0) then
  begin
    MuzuKSipce:=True;
  end
else if odkud_x-sipka_x=0 then
  begin
    v:=sipka_y-odkud_y;
    if v>0 then smer:='dolu' else smer:='nahoru';
    dojezd:=MoznyDojezd(smer,odkud_x,odkud_y);
    if LzeJetToutoRychlosti(dojezd,abs(v)) then MuzuKSipce:=True;
  end;
```

Na závěr ještě kontroluje, jestli je trať k začátku šipky průjezdná.

```
if Pruchozi(odkud_x,odkud_y,sipka_x,sipka_y)=false then MuzuKSipce:=False;
```

### 7.2.1.12 *Lze jet do cíle*

Cílem této funkce je zjistit maximální možné popojetí, tak aby se počítač zvládl některým dalším tahem trefit přesně do cíle. Díky tomu, že je tato funkce řešena rekurzivně, je její kód opravdu krátký.

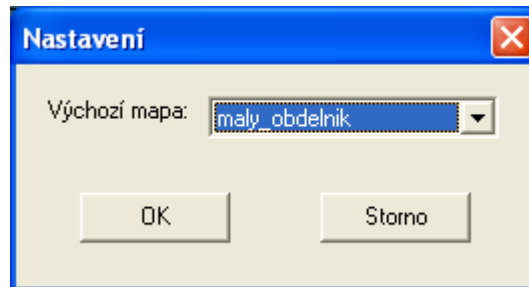
```
if zbyla_draha=0 then test:=true
else if zbyla_draha>0 then
  begin
    if LzeJetDoCile(zbyla_draha-pouzita_rychlost,pouzita_rychlost+1)=true then test:=true
    else if LzeJetDoCile(zbyla_draha-pouzita_rychlost,pouzita_rychlost)=true then test:=true
    else if LzeJetDoCile(zbyla_draha-pouzita_rychlost,pouzita_rychlost-1)=true then test:=true
    else test:=false;
  end
else test:=false;
```

## 7.2.2 Formulář nastavení

### 7.2.2.1 Zobrazení formuláře

Při vstupu do nastavení jsou do výběru vypsány všechny dostupné mapy.

```
ComboBox1.Items.Clear;  
AssignFile(file_mapy, '\\data\mapy.cfg');  
Reset(file_mapy);  
while not Eof(file_mapy) do  
  begin  
    readln(file_mapy, text);  
    ComboBox1.Items.Add(text);  
  end;  
CloseFile(file_mapy);
```



### 7.2.2.2 Stisknutí tlačítka OK

Po potvrzení výběru je nové nastavení uloženo a načtena nová hra s vybranou mapou.

```
AssignFile(file_nastaveni, '\\data\nastaveni.cfg');  
Rewrite(file_nastaveni);  
writeln(file_nastaveni, ComboBox1.Text);  
CloseFile(file_nastaveni);
```

```
form1.Novahra1Click(ButtonOK);
```

## 8 Závěr

Jelikož programování je časově náročná, při obtížnějších projektech nikdy nekončící práce, ani v tomto případě tomu nebude jinak. Když jsem se rozhodl, že naučím počítač taktiku, netušil jsem jak náročný úkol to bude. A tak jsem neustále ladil, opravoval chyby a vylepšoval. Myslím si, že se mi to nakonec podařilo obstojně, což ovšem neznamená, že není dále co vylepšovat.

Zjistil jsem, že práce programátora opravdu není procházkou rájem a že se neobejde bez dlouhého promýšlení co nejlepší taktiky a hodinám strávených nad manuálem hledáním a testováním co nejvhodnějších funkcí. Abych stejně během samotného psaní vymyšleného kódu zjistil, že by se to a to mohlo napsat lépe a pak to často bývá vnitřní boj mezi čistým, krátkým, výkonově optimalizovaným nebo co nejlépe rozšiřitelným kódem. Co vyhrálo záleželo hlavně na situaci a představě do budoucna. Některé kódy jsem čistě z ukázkových důvodů nechal dvakrát různě naprogramované, a tak lze využít jednou to a příště ono, záleží na potřebě a dalším rozšiřování programu.

V dalších fázích vývoje mám v plánu především vylepšit šikmý pohyb a komunikaci se začínajícím uživatelem. Připravené a promyšlené už mám vykreslování mřížky ve hře, tak aby bylo zcela jasné, na která pole lze jet.

Testováním jsem zjistil, že ne vždy je vhodné rozjíždět se na co nejvyšší rychlost, ale bylo by lepší nechávat si nějakou rezervu. Tato funkce už je také připravena k testování. Dále mám připraveno učení se počítače od tahů člověka. Zde jsem se však ještě nerozhodl, jestli jej do hry opravdu zařadím.

## ***Použitá literatura***

1. *Teixeira, S., Pacheco, X.:* Borland Delphi průvodce vývojáře kniha III, Brno, UNIS Publishing, 1999.
2. *Swan, T.:* Mistrovství v Delphi 4, Brno, Computer Press, 1999.
3. *Kučera, L.:* Programování v Borland Delphi, <http://kukensius.webz.cz/delphi.htm>, 2005.