# Bakalářská práce

Simulace kulečníku

**Tomáš Lembacher**

2006

# Obsah

# Úvod

Tématem mojí bakalářské práce je analýza pružného odrazu dvou koulí a dále vytvoření demonstračního programu v systému MatLab, simulujícího chování kulečníku. Práce je rozdělena do tří částí. V první části se budu věnovat matematické analýze chování kulečníku, chování koulí při odrazu od hran kulečníkového stolu a analýze pružného odrazu dvou koulí. Druhá část bude věnována demonstračnímu programu, popisu jednotlivých jeho částí a předvedení jeho výsledků. Poslední část obsahuje shrnutí všech výsledků.

# Analýza

## Kulečník

Jestliže se řekne kulečník, každému se zpravidla vybaví hra, při které se hráči snaží dostat koule do děr, které jsou rozloženy po stranách stolu a v jeho rozích. Této hře se správně říká billiard nebo pool. Já se ale budu této vžité chyby pro přehlednost držet a budu o billiardu mluvit jako o kulečníku.

Kulečník se hraje na stole o maximální úhlopříčce 9 stop (asi 2.75 metru, tj. o rozměrech 356 x 187 cm; budu ho v simulaci používat), který má v rozích a uprostřed delších stran díry. Do těchto děr se hráči snaží umístit obvykle 15 koulí. Z těchto patnácti koulí je jedna černá (hraje se sní až naposledy), 7 koulí celobarevných a 7 koulí polobarevných. Hráči se snaží „šťouchnout“ do šestnácté bílé koule tak, aby pomocí ní zasáhli koule své barvy (tedy buď celo nebo polobarevné) tak, že je dopraví do některé z děr. Na závěr se hráči snaží umístit do některé z děr poslední, černou, kouli. Vyhrává hráč, kterému se to podaří nejdříve.

Toto jsou velice zjednodušená pravidla. Kompletní pravidla je možné najít například na http://www.ok.cz/ludmila/aron/pool.htm

Obrázek č. 1 znázorňuje výchozí postavení při kulečníku.

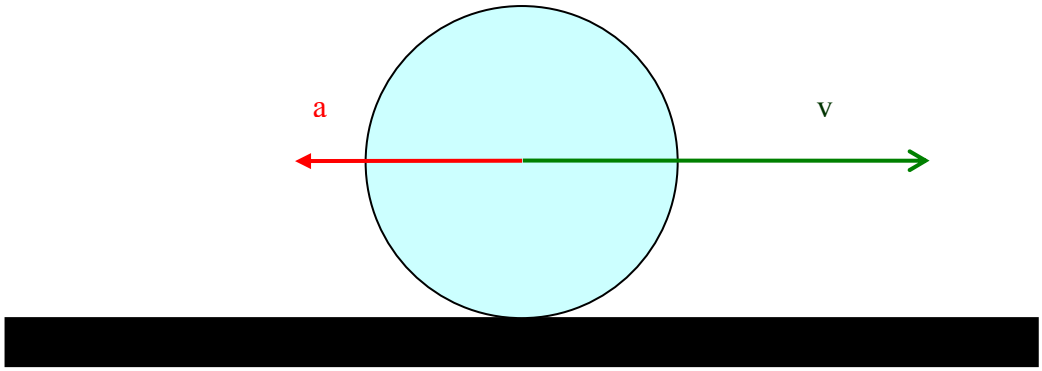
*Obrázek č. 1: Výchozí pozice při kulečníku*

## Pohyb kulečníkové koule

Kulečníkových koulí je několik různých typů lišících se zejména velikostí. Já jsem si vybral koule o poloměru 60 mm a hmotnosti 210 g. Mechanika pohybu koule po kulečníkovém stole je velice složitá a komplexní úloha. Pro zjednodušeni jsem zanedbal jiné rotace než valivý pohyb koule a předpokládám, že se koule hned zpočátku pohybu valí. Dále považuji veškeré srážky (v souladu se zadáním práce) za dokonale pružné a jediný odpor uvažuji valivý. Odpor vzduchu zanedbávám.

Problém se tímto zúžil na kutálení koule za účasti valivého tření, jehož síla působí proti pohybu koule. Způsobuje tak zrychlení s opačným směrem k rychlosti, a tím kouli zpomaluje.

Lépe celou situaci popisuje obrázek č. 2.




*Obrázek č. 2: Bilance rychlosti a zrychlení působící na kouli (řez kolmo na stůl)*

Velikost zrychlení *a* je přímo úměrné rychlosti *v* a nepřímo úměrné hmotnosti koule. Platí:

$$m \cdot a = -\xi \cdot v$$

$$a = -\frac{\xi}{m} v$$

Kde $\xi$ je součinitel tření a *m* je hmotnost koule. Zde je však nutná ještě malá odbočka. Rychlost *v* je translační rychlost koule způsobená valivým pohybem. Koule však vykonává rotační pohyb okolo své osy a proto část její kinetické energie přísluší právě této rotaci. Tato „ztráta“ energie by se měla projevit menší translační rychlostí. V těchto rovnicích by se to mohlo projevit větší hmotností koule. Proto je třeba brát zde uvedenou hmotnost *m* jako již přepočtenou zvýšenou hmotnost. Pro zjednodušení celé situace jsem však tento přepočet zanedbal.

Vezmu-li v úvahu, že $\frac{dv}{dt} = a$ a $\frac{dx}{dt} = v$, čili $\frac{d^2x}{dt^2} = a$, kde *x* je poloha středu koule, tak dostávám diferenciální rovnici pro pohyb koule (*u* je počáteční rychlost).

$$k = \frac{\xi}{m}$$

$$\frac{d^2x}{dt^2} = -kv + u$$

Z čehož po úpravě dostanu: $\frac{d^2x}{dt^2} + k\frac{dx}{dt} = u$

Po provedení Laplaceovy transformace a úpravě dostanu přenos pro polohu, který lze již bez dalších problémů použít v programu MatLab.

$$G_x(s) = \frac{1}{s(s+k)}$$

Obdobně pro rychlost koule platí rovnice:

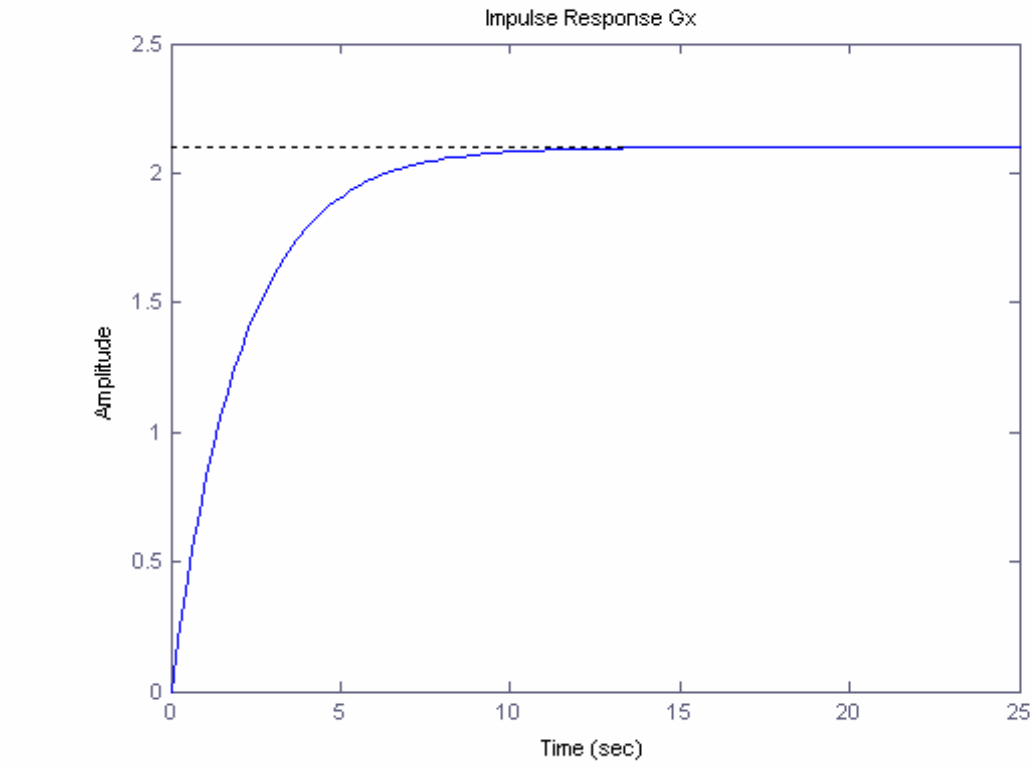
$$\frac{dv}{dt} + kv = u$$

Po provedení transformace a úpravě dostanu přenos pro rychlost:

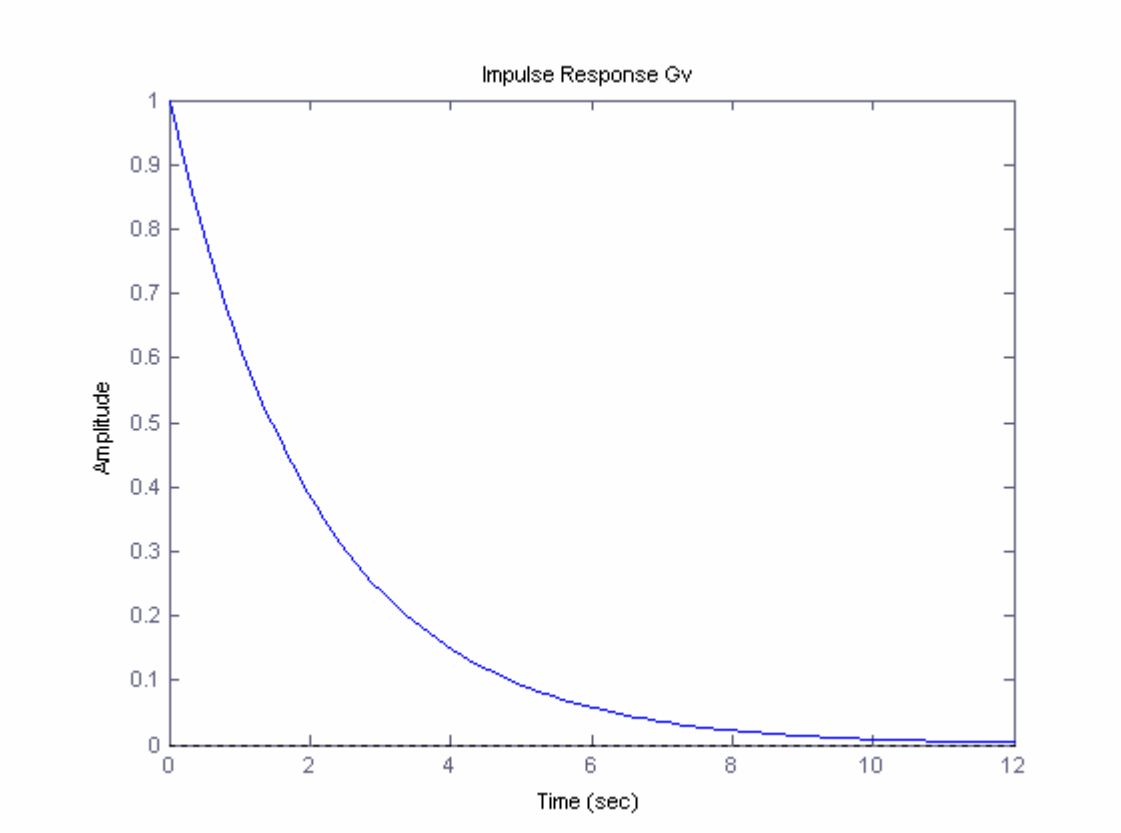
$$G_v(s) = \frac{1}{(s+k)}$$

K ověření výpočtů jsem použil program MatLab. Abych co nejlépe napodobil kulečník, použil jsem k ověření impulsní charakteristiku, která se podobá „šťouchnutí“ do koule. Cílem bylo ověřit, zda rychlost bude klesat k nule (koule vlivem valivého odporu zpomaluje, až se zastaví), zatímco poloha se bude měnit, až se ustálí na nějaké hodnotě (koule se kutálí na nějaké místo, kde pak zůstane). Naměřené charakteristiky jsou níže.

```
Command Window

>> xi=0.1;
>> m=0.21;
>> k=xi/m;
>> Gx=tf(1,[1 k 0]);
>> impulse(Gx)
>> Gv=tf(1,[1 k]);
>> impulse(Gv)
```




*Obrázek č. 4: Impulsová charakteristika přenosu polohy $G_x$*




*Obrázek č .5: Impulsová charakteristika přenosu rychlosti $G_v$*

Z obrázků je jasně patrné, že koule se začne pohybovat, její rychlost exponenciálně klesá a poloha nabíhá na konečnou hodnotu. Přenosy se tedy chovaly podle předpokladů a jsou matematickým modelem valení koule.

## ***Odraz koule na hranách (stěnách) stolu***

Při odrazu koule využiji několika obecných principů. Je to jednak princip nezávislosti pohybů, a jednak zákony zachování hybnosti a kinetické energie. Princip nezávislosti pohybů mi dovoluje rozložit rychlost na její složky $v_x$ a $v_y$ a dále s nimi zacházet nezávisle na sobě. Představme si dvě tělesa (koule), která leží na jedné přímce rovnoběžně s osou x. Jejich počáteční rychlosti jsou ve směru přímky. Označím je $v_{01}$ a $v_{02}$. Nyní hledám vztah mezi těmito rychlostmi a rychlostmi obou koulí po srážce $v_{11}$ a $v_{12}$ Ten se dá odvodit z obou zmíněných zákonů zachování. V následujícím zanedbávám třecí síly a tělesa považuji za dokonale hladká.

V mém případě bude mít zákon zachování hybnosti tvar:

(1) $$m_1 v_{01} + m_2 v_{02} = m_1 v_{11} + m_2 v_{12}$$

Zákon zachování kinetické energie má tvar:

(2) $$\frac{1}{2} m_1 {v_{01}}^2 + \frac{1}{2} m_2 {v_{02}}^2 = \frac{1}{2} m_1 {v_{11}}^2 + \frac{1}{2} m_2 {v_{12}}^2$$

Rychlosti $v_{01}$ a $v_{02}$ jsou počáteční a $v_{11}$ a $v_{12}$ jsou konečné rychlosti obou koulí. Úpravami dostávám:

(3) $$m_1\left(v_{01}-v_{11}\right)=m_2\left(v_{12}-v_{02}\right)$$

(4) $$m_1\left({v_{01}}^2-{v_{11}}^2\right)=m_2\left({v_{12}}^2-{v_{02}}^2\right)$$

Tímto mám již vše připraveno, abych rovnice vyřešil. Nejdříve obě rovnice vydělím a vyjádřím si postupně $v_{11}$ a $v_{12}$

(5) $$v_{01}+v_{11}=v_{12}+v_{02}$$

(6) $$v_{11}=v_{12}+v_{02}-v_{01}$$

(7) $$v_{12}=v_{11}+v_{01}+v_{02}$$

Rovnice (6) a (7) můžu teď dosadit do (1) a dostávám řešení ve tvaru:

$$v_{11}=\frac{v_{01}\left(m_1-m_2\right)+2m_2v_{02}}{m_1+m_2}$$

$$v_{12}=\frac{2m_1v_{01}+v_{02}\left(m_2-m_1\right)}{m_1+m_2}$$

Uvažujme teď odraz koule o hranu stolu, která je ve směru osy $x$. Složka rychlosti $v_x$ je na plochu odrazu kolmá, a tak se odrazu neúčastní. Bude se ho tedy účastnit jen složka $v_y$. Pro tu, dosadíme-li ji za rychlost $v_{01}$, budou platit rovnice uvedené výše. Složka $v_x$ se odrazem nijak nezmění.

Další důležitou úvahou je, že koule má zanedbatelnou hmotnost oproti stolu, tedy že platí $m_2 >> m_1$ (kde $m_1$ je hmotnost koule a $m_2$ je hmotnost stolu). Rovnice se tím zjednodušují na tvar

$$v_{11}=-v_{01}+2v_{02}$$

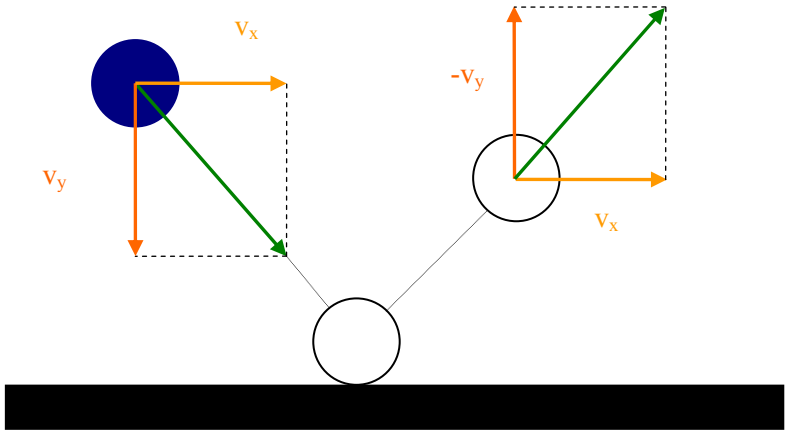
$$v_{12}=v_{02}$$

Stůl je ale na počátku v klidu (tedy $v_{02}=0$), a tak dostáváme konečné řešení ve tvaru

$$v_{11}=-v_{01}$$

$$v_{12}=0$$

Je vidět, že když do rovnic dosadíme za $v_{01}$ rychlost koule $v_y$, tak po odrazu vychází tato rychlost stejné velikosti, ale opačného směru. Názorněji je situace zobrazena na obrázku č. 6.

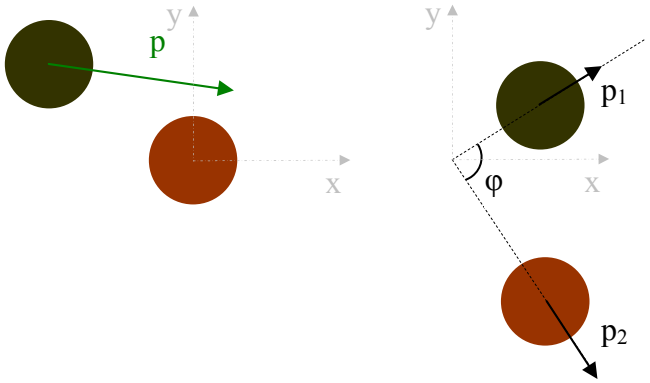



*Obrázek č. 6: Odraz koule na hraně stolu (pohled shora).*

Při odrazu na hraně rovnoběžné s osou y je situace obdobná, jenom se srážky neúčastní rychlost $v_y$, ale rychlost $v_x$.

## ***Odraz dvou koulí***

Vzájemný odraz dvou koulí je složitějším problémem. Opět se zde ovšem využívá stejných principů zákonů zachování hybnosti a kinetické energie a principu nezávislosti pohybů. Na obrázku č. 7 je zobrazen odraz dvou kouli, z nichž jedna je v klidu.




*Obrázek č. 7: Odraz dvou koulí. Situace před a po odrazu.*

Koule s hybností $p$ se střetne s koulí která je v klidu. Výsledkem jsou koule s hybnostmi $p_1$ a $p_2$, které se pohybují od sebe pod úhlem $\varphi$. Na začátek nás bude zajímat právě úhel $\varphi$. Jeho velikost se dá odvodit z obou zákonů zachování. Protože zákon zachování hybnosti platí pro jednotlivé složky hybnosti (tedy $p_x$ a $p_y$), tak také platí i pro celkovou velikost hybnosti. Předpokládejme stejné hmotnosti obou koulí.

$$p_1 = mv_1$$

$$p_2 = mv_2$$
$$p = mv$$

Po využití Kosinovy věty dostanu

$$p^2 = {p_1}^2 + {p_2}^2 + 2p_1p_2\cos(\varphi)$$

Dosadím předchozí vztahy, vykrátím *m* a mám

$$v^2 = {v_1}^2 + {v_2}^2 + 2mv_1v_2\cos(\varphi)$$

Ze vztahu pro kinetickou energii mi po dosazení a vykrácení vyjde

$$v^2 = {v_1}^2 + {v_2}^2$$

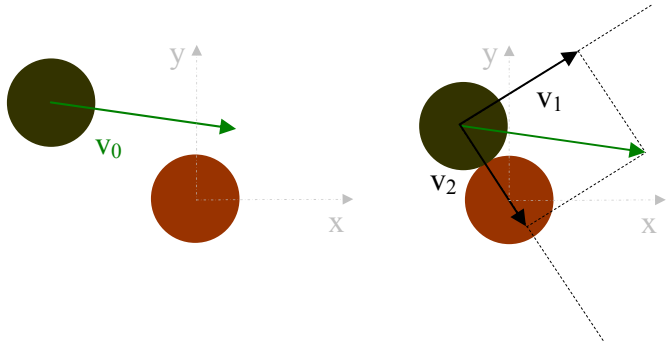
Je zřejmé, že pro úhel $\varphi$ musí platit

$$\cos(\varphi) = 0$$

Takže

$$\varphi = \frac{\pi}{2};0$$

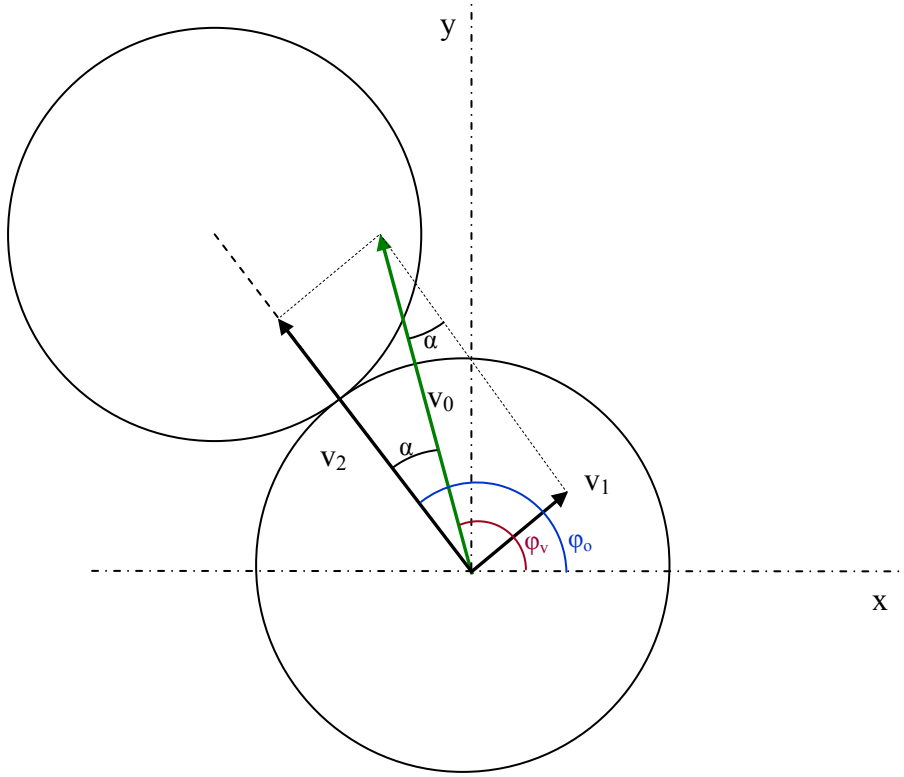
Úhel mezi koulemi je tedy buď 90° a nebo je 0° (to dopovídá případu, kdy jsou koule na jedné přímce a rychlost jedné z nich je s touto přímkou rovnoběžná). S tímto poznatkem můžu udělat rozklad rychlostí při střetu.




*Obrázek č. 8: Rozklad rychlostí při srážce*

Po srážce se první koule bude pohybovat rychlostí $v_1$ a koule, která byla v klidu, se bude pohybovat rychlostí $v_2$. Rychlosti budou svírat pravý úhel. Problém se tímto zúžil na výpočet rozkladu rychlosti $v_0$.

Při tomto rozkladu se využije právě pravého úhlu a goniometrických funkcí. Důležitým úkolem je zjistit úhel, který svírá rychlost $v_0$ s osou nárazu, úhel α. Názorněji to ukáže obrázek č. 9.




*Obrázek č. 9: Zjištění úhlu α*

Pro zjištění úhlu α mi nejlépe poslouží polární souřadnice. Úhel $\varphi_v$ zjistím transformací vektoru $v_0$ do polárních souřadnic. Úhel $\varphi_o$ zjistím vytvořením vektoru odečtením souřadnic středů koulí a převedením tohoto vektoru do polárních souřadnic. Úhel α pak zjistím z rozdílu těchto vektorů.

$$\alpha = sign(\varphi_o - \varphi_v)(\varphi_o - \varphi_v)$$

Funkce signum je tu proto, aby se ošetřil případ, kdy bude úhel $\varphi_o$ menší než $\varphi_v$. Velikosti obou vektorů rychlosti potom budou

$$v_1 = v_0 \cos\alpha$$

$$v_2 = v_0 \sin\alpha$$

Toto celé platí v případě, že se pohybuje jen jedna koule a druhá je v klidu. Pakliže se pohybují obě koule, využije se principu nezávislosti pohybů. Nejdříve se uvažuje, že jedna z koulí je v klidu a spočítají se pro obě koule příslušné rychlosti. Pak se uvažuje druhá koule v klidu, a opět se spočítají rychlosti pro obě koule. Následně se všechny rychlosti příslušné té které kouli sečtou a vznikne z nich výsledná rychlost pro danou kouli. Nutno ještě podotknout, že toto platí, srazí-li se pouze dvě koule. Pakliže by jedna koule narazila naráz do dalších dvou, pak by se její rychlost rozložila do směrů spojnic středů koulí a koule by se zastavila. Při srážce čtyř koulí najednou, již není možné početně odrazy určit. Já zde ale předpokládám jen srážku dvou koulí najednou.

# Software

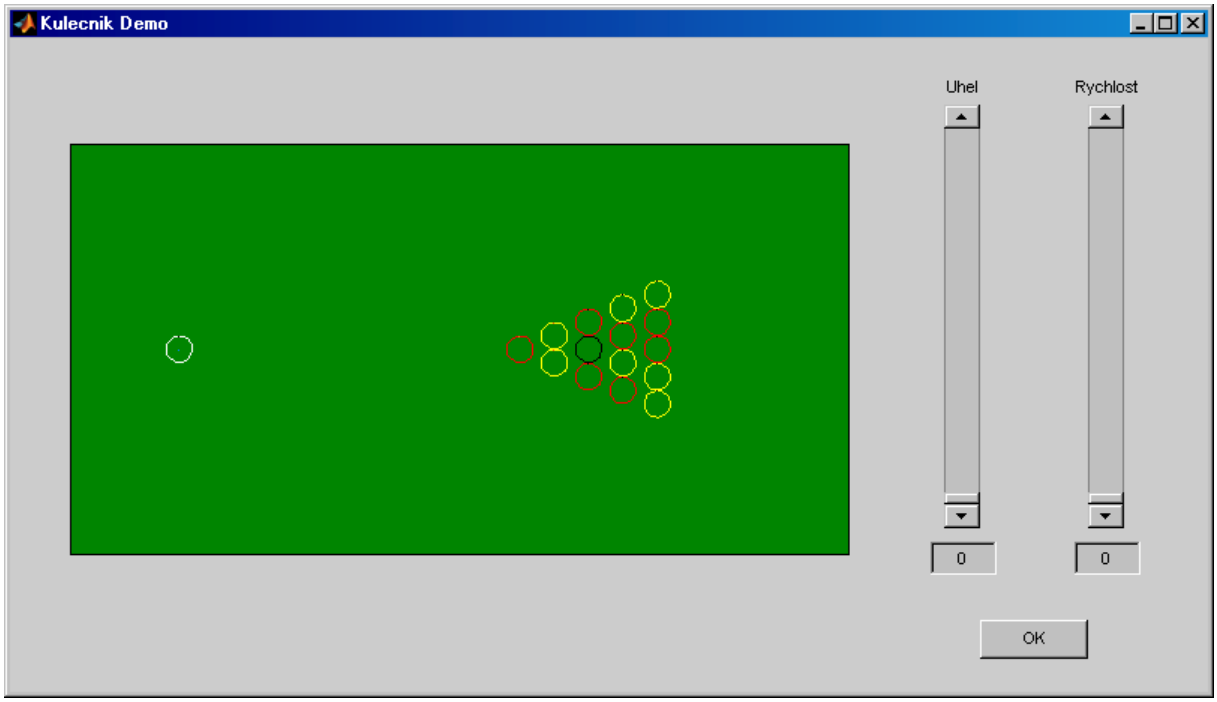
Pro účely demonstrace odvozených vztahů jsem naprogramoval v prostředí MatLabu demo program, simulující chování kulečníku. V této části popíší jeho jednotlivé části a jejich funkci. Následovat pak budou obrázky z běhu programu.

## MatLab

MatLab je velice komplexní prostředí pro matematické výpočty všeho druhu. Je však dobře optimalizované pro veškeré simulace. Pro většinu problémů jsou připravené i jednotlivé dílčí součásti. Pakliže uživateli cokoliv začne chybět, může si danou chybějící komponentu doprogramovat například v jazyce C. Prostředí MatLab je dokonale optimalizováno pro práci s maticemi a pro optimální práci je vhodné matice hojně používat. Většina operací se tím velice urychlí. Proto jsem se i já snažil (pakliže to bylo možné) většinu „hromadných“ operací převést na matice (ne vždy to ale bylo možné). Pro spuštění mého programu je třeba jen soubor `kulecnik.m`, který musí být v některém adresáři, který MatLab prohledává při spouštění funkcí. Demo se pak spustí příkazem `kulecnik`.

## Program `kulecnik`

Po spuštění programu `kulecnik` se spustí grafické uživatelské rozhraní vyobrazené na obrázku č. 10.




*Obrázek č. 10: Grafické ovládací prostředí programu.*

Hlavní část plochy okna zabírá zobrazení situace na kulečníku (jde o pohled shora). V pravé části jsou pak ovládací prvky. Těmi se nastavuje úhel a rychlost pohybu bílé koule.

Parametry lze nastavit jednak zapsáním do textového pole a potvrzením klávesou Enter, a nebo nastavením posuvníků. Úhel pohybu koule se nastavuje ve stupních vzhledem k ose *x*. Rozsah úhlu je 0 až 360 stupňů. Rychlost se nastavuje v metrech za sekundu. Její rozsah je 0 až 8 metrů za sekundu. V průběhu zadávání se zobrazuje pomocná úsečka, která vychází ze středu bílé koule, zobrazuje směr pohybu a její délka se mění s nastavenou rychlostí (čím větší, tím delší). Po kliknutí na tlačítko OK se spustí samotná simulace. Během ní jsou ovládací prvky nepřístupné. Po skončení simulace se ovládací prvky opět zpřístupní, je možné zadat nové hodnoty a pokračovat v simulaci opětovným kliknutím na OK.

# ***Základní principy aplikace***

## Reprezentace koulí

Každá koule je reprezentována strukturou, ve které jsou uloženy všechny potřebné parametry. Předpokládám ale stejný poloměr všech koulí a stejnou hmotnost. Tyto položky proto ve struktuře nejsou. Protože je koulí dohromady 16, tak jsou struktury uloženy v poli. Následující ukázka kódu ukazuje inicializaci celého pole `koule`.

```
%inicializace pole kouli
        koule(1).color='white'; %barva
        koule(1).x=0; %souradnice x
        koule(1).y=0; %souradnice y
        koule(1).v0=[0 0]; %pocatecni rychlost
        koule(1).x0=0; %pocatecni poloha
        koule(1).y0=0; %pocatecni poloha
        koule(1).i=1; %pocet kroku od posledního odrazu
        koule(1).handle=0; %handle na 'graf' dané koule

        for i=2:15
            if(mod(i,2)==0)
                koule(i).color='red';
            else
                koule(i).color='yellow';
            end;
            koule(i).x=0;
            koule(i).y=0;
            koule(i).v0=[0 0];
            koule(i).x0=0;
            koule(i).y0=0;
            koule(i).i=1;
            koule(i).handle=0;
        end;

        koule(16).color='black';
        koule(16).x=0;
        koule(16).y=0;
        koule(16).v0=[0 0];
        koule(16).x0=0;
        koule(16).y0=0;
        koule(16).i=1;
        koule(16).handle=0;
```

Je vidět, že první koule bude mít bílou barvu, poslední černou barvu a mezi nimi budou sudé koule červené a liché žluté. Jednotlivé složky struktury jsou popsány přímo ve zdrojovém kódu.

## Pohyb koulí

Při pohybu koule využívám toho, že výstup systému je impulzní charakteristika vynásobená velikostí vstupu. Impulsní charakteristika je nejlepším napodobením úderu tágem do koule. Výstupem je diference výstupní veličiny od počátečního stavu. Proto je možné vytvořit si na začátku programu dostatečně dlouhou impulsovou charakteristiku pro každý použitý přenos a pak už jen používat tuto „tabulku“ a násobit jí příslušnými vstupy. Abychom pak dostali správné výstupy, je třeba ještě přičítat počáteční hodnoty výstupních veličin.

```
konstanty.t=[0:0.01:100];%cas
%...
konstanty.v=tf(1,[1 konstanty.k]);%prenos rychlosti
konstanty.p=tf(1,[1 konstanty.k 0]);%prenos polohy
%vychozi impulzni charakteristiky
konstanty.xref=impulse(konstanty.p,konstanty.t);
konstanty.vref=impulse(konstanty.v,konstanty.t);
```

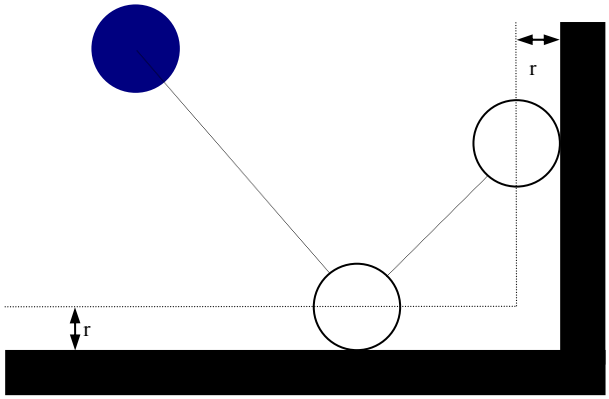
V proměnných `konstanty.xref` a `konstanty.vref` jsou referenční impulsové charakteristiky dostatečné dlouhé (100s) na to, aby jejich konce nebylo v programu dosaženo. V simulační smyčce se pak poloha koulí vypočte snadno.

```
%Vypocte se nova poloha...
koule(k).x=koule(k).x0+koule(k).v0(1)*konstanty.xref(koule(k).i);
koule(k).y=koule(k).y0+koule(k).v0(2)*konstanty.xref(koule(k).i);
%...
%Vypočte se nová výchozí poloha koule (naprikald pri odrazu)
koule(k).v0(1)=koule(k).v0(1)*konstanty.vref(koule(k).i);
koule(k).v0(2)=koule(k).v0(2)*konstanty.vref(koule(k).i);
```

Každá koule má svoje počítadlo, které určuje jak je koule „daleko“ v impulsové charakteristice, respektive před jakou dobou se naposledy od něčeho odrazila.

## Odrazy od stěn

Pro programování odrazů koulí od rovných pevných překážek (kterou stěna je) se používá technika stínových zdí (shadow wall).




*Obrázek č. 11: Shadow wall*

Technika stínových zdí spočívá v tom, že se ve vzdálenosti poloměru od zdi vytvoří jistá pomyslná hranice, za kterou se nesmí dostat střed koule. Ve chvíli, kdy na této hranici střed koule je (nebo je za ní), nastala určitě kolize se zdí a musí se spočítat příslušný odraz.
Na kulečníkovém stole je situace o to jednodušší, že stěny jsou na sebe kolmo. Jestliže se počátek souřadnic posune do levého dolního rohu (což jsem udělal), tak se situace opět zjednodušuje. Střed koule nesmí opustit pomyslný čtyřúhelník vytvořený stínovými zdmi okolo hraničních oblastí (tedy opravdových zdí, resp. stěn stolu).
Rozměry stolu jsou 356 x 187 cm.

```
%leva a prava stena
if((koule(k).x>(356-konstanty.r))||(koule(k).x<konstanty.r))
   koule(k).v0(1)=koule(k).v0(1)*konstanty.vref(koule(k).i);
   koule(k).v0(2)=koule(k).v0(2)*konstanty.vref(koule(k).i);
   koule(k).x0=koule(k).x;
   koule(k).y0=koule(k).y;
   koule(k).v0(1)=(-1)*koule(k).v0(1); % vypocet odrazu
   koule(k).i=1;
end

%horni a dolni stena
if(((koule(k).y>=(187-konstanty.r))||(koule(k).y<=konstanty.r))
   koule(k).v0(1)=koule(k).v0(1)*konstanty.vref(koule(k).i);
   koule(k).v0(2)=koule(k).v0(2)*konstanty.vref(koule(k).i);
   koule(k).x0=koule(k).x;
   koule(k).y0=koule(k).y;
   koule(k).v0(2)=(-1)*koule(k).v0(2); %vypocet odrazu
   koule(k).i=1;
end
```

Pro určení všech odrazů postačují 4 podmínky. Protože se ale koule na dvou protilehlých stěnách chová velice podobně, tak jsem mohl vždy dvě a dvě podmínky sloučit do jednoho bloku příkazů. Po detekci odrazu je třeba provést ještě několik dalších kroků na zajištění správného chodu simulace. Ze zdrojového kódu je vidět, že nejprve se spočítá aktuální rychlost koule a přiřadí se do výchozí rychlosti. Dále se aktuální poloha koule přiřadí do počáteční polohy. V následujícím kroku se vypočte nová rychlost koule po odrazu (viz. kapitola „Odraz koule na stěnách stolu“). Posledním krokem, který je třeba udělat, je nastavit počítadlo koule na výchozí hodnotu po odrazu, aby koule „věděla“, že se odrazila.

Na závěr nutno ještě poznamenat, že při odrazu jsem zanedbal několik skutečností. Jednak jsem zanedbal to, že se koule v době odrazu může nacházet nepatrně za okrajem stolu. Do

simulace to však nevnáší příliš velkou chybu. Dále jsem ještě zanedbal pružnost koule a považuji jí za ideálně tuhé těleso.

## Vzájemná kolize dvou koulí

Úvaha, kdy dochází ke kolizi dvou koulí není příliš obtížná. Tak jako v předchozím případě jsem zanedbal drobné přesahy koulí při vzájemné kolizi. Na rozdíl od odrazu na stěně stolu to již na tomto místě dělalo problémy a tak jsem musel vytvořit algoritmus, který pozná, jestli se koule od sebe vzdalují, nebo se k sobě přibližují. Jinak bych musel velice složitě dopočítávat přesný čas kolize a to nebylo moji hlavní náplní práce.

Kolize dvou koulí nastane tehdy, když se vzdálenost středů rovná nebo menší, než dvojnásobek poloměru. Při výpočtu vzájemných vzdáleností všech koulí se dají využít schopnosti MatLabu pracovat s maticemi. Jinak bych musel počítat v cyklu, vždy pro každou kouli, šestnáct vzdáleností v dalším cyklu. Při použití matic se výpočet vzájemných vzdáleností „smrskne“ na 3 řádky zdrojového kódu. Jak jsem postupoval?

Mějme souřadnice středů dvou koulí $S_1[x_1, y_1]$ a $S_2[x_2, y_2]$. Jejich vzdálenost je

$$|S_1S_2| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Toto je třeba udělat vzájemně pro každé dvě koule. Pakliže máme k dispozici matice, je celý problém poměrně jednoduchý. Předvedu to na čtyřech koulích.

$$\begin{pmatrix} x_1 & x_1 & x_1 & x_1 \\ x_2 & x_2 & x_2 & x_2 \\ x_3 & x_3 & x_3 & x_3 \\ x_4 & x_4 & x_4 & x_4 \end{pmatrix} - \begin{pmatrix} x_1 & x_2 & x_3 & x_4 \\ x_1 & x_2 & x_3 & x_4 \\ x_1 & x_2 & x_3 & x_4 \\ x_1 & x_2 & x_3 & x_4 \end{pmatrix} = \begin{pmatrix} 0 & x_1 - x_2 & x_1 - x_3 & x_1 - x_4 \\ x_2 - x_1 & 0 & x_2 - x_3 & x_1 - x_4 \\ x_3 - x_2 & x_3 - x_2 & 0 & x_3 - x_4 \\ x_4 - x_1 & x_4 - x_2 & x_4 - x_3 & 0 \end{pmatrix}$$

Pod diagonálou výsledné matice máme všechny potřebné rozdíly. Nyní můžeme udělat totéž s ypsilonovými souřadnicemi a pak již stačí obě matice umocnit, sečíst a součet odmocnit. Získáme tím výsledek. Za povšimnutí stojí, že matice v rozdílu jsou vůči sobě transponované. Získáme-li si tedy jednu, jednoduše z ní získáme i druhou.

```
[x_stejne y_stejne]=meshgrid(cat(2,koule(1,:).x),cat(2,koule(1,:).y));
x_stejne=x_stejne';
distance=sqrt(((x_stejne'-x_stejne).^2)+((y_stejne'-y_stejne).^2));
distance=tril(distance)+konstanty.upper_diag;
```

První dva řády slouží na vytvoření výchozích matic. Funkce `meshgrid` se používá při tvorbě 3D grafů. Zde mi ale pěkně posloužila. Na třetím řádku probíhá samotný výpočet vzájemných vzdáleností. Protože je výsledná matice symetrická podle hlavní diagonály a v prohledávání této matice by se vyskytovali duplicitní nálezy, tak jsem na posledním řádku provedl operaci, která všechny hodnoty nad hlavní diagonálou nastaví na přijatelné (tedy větší než dvojnásobek poloměru) a tím jsem se vyhnul duplicitním nálezům kolizí. Matice `konstanty.upper_diag` je mnou předpřipravená matice, která má na a pod hlavní diagonálou samé nuly a zbytek prvků je nastaven na hodnotu 20.

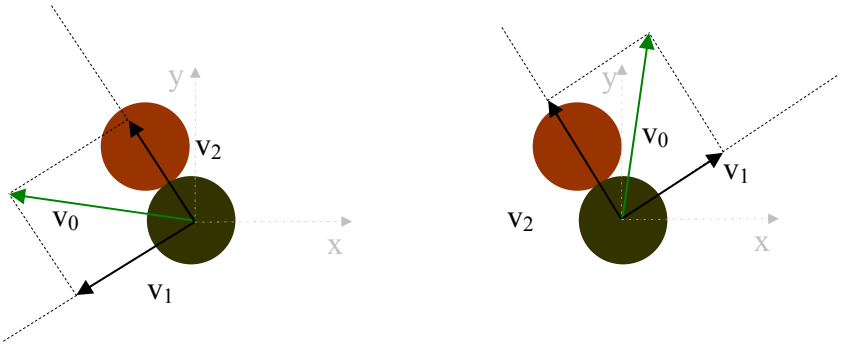
Dále následuje vyhledání hodnot, které odpovídají kolizi

```
%nalezeni všech prvku, které jsou mensi nez dvojnasobek polomeru
[x_found y_found]=find(distance<(2*konstanty.r*100));
%...vypocte se diference poloh ve dvou po sobe nasledujicich
%krocich simulace...
d_dist=distance-distance2;

%...pro kazdou kolizi se spocita odraz
for k=1:size(x_found)
    if(d_dist(x_found(k),y_found(k))>=0)
    %...jestlize je diference predchazejiciho a tohoto kroku
    %vetsi, nebo rovna nule, tak koule vuci sobe bud stoji,
    %a nebo se vzdaluji. V tom pripade odraz nenastava...
       continue;
    end
%pokracuje dalsi zpracovani odrazu...
```

Nalezené prvky se procházejí v cyklu a pokud je rozdíl jejich předchozí a aktuální vzdálenosti záporný, znamená to, že se k sobě přibližují a tak opravdu nastal odraz. Pakliže by rozdíl byl kladný, tak se koule od sebe vzdalují a jedná se o jejich dočasný přesah z minulého odrazu.

## Výpočet odrazu dvou koulí

Výpočet toho co nastane po kolizi dvou koulí je asi nejkomplikovanější. U každé výsledné rychlosti je třeba určit její velikost a nejlépe úhel, aby se daly spočítat její složky. V praxi funguje výpočet odrazu tak, že se postupně jedna i druhá koule uvažují v klidu. Následně se výsledné rychlosti z obou srážek sečtou a vzniknou výsledné rychlosti obou koulí. Z obrázku č. 9 je zřejmé, že jediné co nám zbývá zjistit je úhel rychlosti $v_1$ vzhledem k ose x (aby se dala rychlost rozložit do složek).Úhel rychlosti $v_1$ se ale mění v závislosti na poloze koulí. Obrázek č. 12 ukazuje dvě možné polohy.




*Obrázek č. 12.: Různé polohy rychlosti $v_1$ v závislosti na poloze $v_0$ a spojnice středů*

Je vidět, že jestliže argument rychlosti $v_0$ je větší než argument spojnice středů, tak je úhel $v_1$ roven argumentu spojnice s 90ti stupni navíc. Je-li tomu opačně, tak je úhel roven argumentu spojnice o 90 stupňů zmenšený. Může nastat ještě třetí možnost. A to ta, že rozdíl argumentu rychlosti $v_0$ a spojnice středů koulí je 0°. Pak je rychlost $v_1$ nulová a její argument mě nezajímá (tedy může být libovolný). Tím mám spolu s obrázkem č. 9 již všechny potřebné informace, abych mohl zahájit výpočet odrazu.

V programu se nejdříve vypočtou jednotlivé složky původních rychlostí koulí a jejich velikosti

```
v01x=koule(x_found(k)).v0(1)*konstanty.vref(koule(x_found(k)).i);
v01y=koule(x_found(k)).v0(2)*konstanty.vref(koule(x_found(k)).i);

v02x=koule(y_found(k)).v0(1)*konstanty.vref(koule(y_found(k)).i);
v02y=koule(y_found(k)).v0(2)*konstanty.vref(koule(y_found(k)).i);

v01=sqrt(v01x^2+v01y^2);
v02=sqrt(v02x^2+v02y^2);
```

Následuje výpočet všech potřebných úhlů pro jednu i druhou kouli

```
%uhly pro kouli 1
deltax1=koule(y_found(k)).x-koule(x_found(k)).x;
deltay1=koule(y_found(k)).y-koule(x_found(k)).y;

alfa1=cart2pol(deltax1,deltay1);%argument osy stretu
alfa1=mod(alfa1,2*pi);

alfa2=cart2pol(v01x,v01y);%argument rychlosti koule 1
alfa2=mod(alfa2,2*pi);

alfa=alfa1-alfa2;
alfa=sign(alfa)*alfa;

switch(sign(alfa2-alfa1))
     case 1
          alfa3=alfa1+pi/2; % uhel rychlosti v1

     case 0
          alfa3=0;

     case -1
          alfa3=2*pi-(pi/2-alfa1);
end
%uhly pro kouli 1
deltax2=koule(x_found(k)).x-koule(y_found(k)).x;
eltay2=koule(x_found(k)).y-koule(y_found(k)).y;

beta1=cart2pol(deltax2,deltay2);%argument osy stretu
beta1=mod(beta1,2*pi);

beta2=cart2pol(v02x,v02y);%argument rychlosti koule2
beta2=mod(beta2,2*pi);

beta=beta1-beta2;
beta=sign(beta)*beta;

switch(sign(beta2-beta1))
      case 1
           beta3=beta1+pi/2;

      case 0
           beta3=0;

       case -1
           beta3=2*pi-(pi/2-beta1);
end
```

A nyní následují konkrétní výpočty jednotlivých rychlostí a jejich následné součty

```
%rychlost v1 pro kouli 1
v11x=v01*sin(alfa)*cos(alfa3);
v11y=v01*sin(alfa)*sin(alfa3);

%rychlost v2 pro kouli 1
v21x=v01*cos(alfa)*cos(alfa1);
v21y=v01*cos(alfa)*sin(alfa1);

%rychlost v1 pro kouli 2
v12x=v02*sin(beta)*cos(beta3);
v12y=v02*sin(beta)*sin(beta3);

%rychlost v2 pro kouli 2
v22x=v02*cos(beta)*cos(beta1);
v22y=v02*cos(beta)*sin(beta1);

koule(y_found(k)).v0(1)=v12x+v21x;
koule(y_found(k)).v0(2)=v12y+v21y;

koule(x_found(k)).v0(1)=v22x+v11x;
koule(x_found(k)).v0(2)=v22y+v11y;
```

Celý výpočet probíhá přesně podle odvozených vztahů. Nepočítám však nejprve velikosti rychlostí, ale hned také jejich složky.

Po výpočtu rychlostí již stačí jen nastavit výchozí parametry podobně, jako u odrazů od stěn stolu.

```
%nastavení novych vychozich hodnot pro obe koule
koule(x_found(k)).i=1;
koule(y_found(k)).i=1;

koule(x_found(k)).x0=koule(x_found(k)).x;
koule(x_found(k)).y0=koule(x_found(k)).y;

koule(y_found(k)).x0=koule(y_found(k)).x;
koule(y_found(k)).y0=koule(y_found(k)).y;
```

## Ukončení simulace

Simulace skončí tehdy, jestliže součet velikostí rychlostí klesne pod stanovenou hranici. Tuto hranici jsem stanovil experimentálně.

```
v_celkova=0;
for k=1:size(koule,2)
    vx=koule(k).v0(1)*konstanty.vref(koule(k).i);
    vy=koule(k).v0(2)*konstanty.vref(koule(k).i);
    koule(k).i=koule(k).i+1;
    v_celkova=v_celkova+sqrt(vx^2+vy^2);
end
%...jestlize klesne celkova rychlost pod danou mez, tak je
%simulace zastavena...
if(v_celkova<5)
  disp('Konec simulace');
%...
```

## Způsob zobrazování dat

Každé kouli je přiřazen soubor dat, který představuje kružnici na ploše grafu. V cyklu se pak vykreslují jednotlivé koule. Pro správné a rychlé zobrazování je u grafů použita vlastnost `EraseMode` nastavená na `xor`. Nepřekresluje se tak celá plocha grafu, ale jen změněné části. Při inicializaci se jednotlivé koule vykreslují příkazem `plot`

```
hold on;
for k=1:size(koule,2)
    koule(k).x=koule(k).x0;
    koule(k).y=koule(k).y0;

    koule(k).handle=plot((6*konstanty.xx+koule(k).x),...
                (6*konstanty.yy+koule(k).y),koule(k).color,...
                'LineWidth',1,...
                'EraseMode', 'xor',...
                'Tag', num2str(k));
end
```

V průběhu simulace jsou už ale měněná jen data těchto grafů.

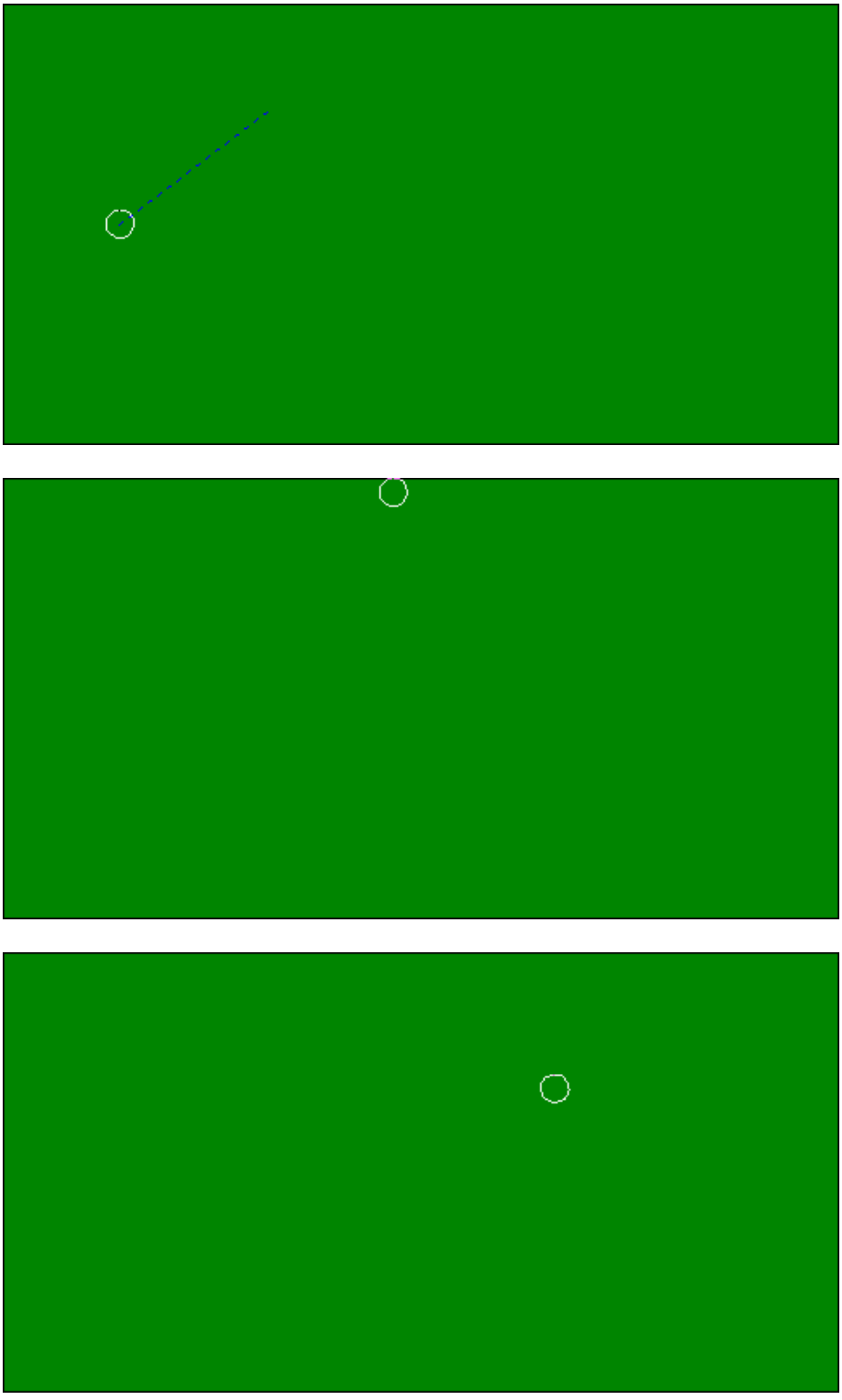
```
set(koule(k).handle,'XData',(6*konstanty.xx+koule(k).x),...
                    'YData',(6*konstanty.yy+koule(k).y));
```

Simulační smyčka běží s periodou odpovídající času potřebnému k vykonání jedné smyčky plus 0.01 sekundy. Na současných průměrných počítačích je doba výpočtu zanedbatelná a perioda je přibližně oněch 0.01 sekundy. Touto frekvencí jsou překreslovány i koule. Je tak zajištěna plynulá animace.

## ***Demonstrace aplikace***

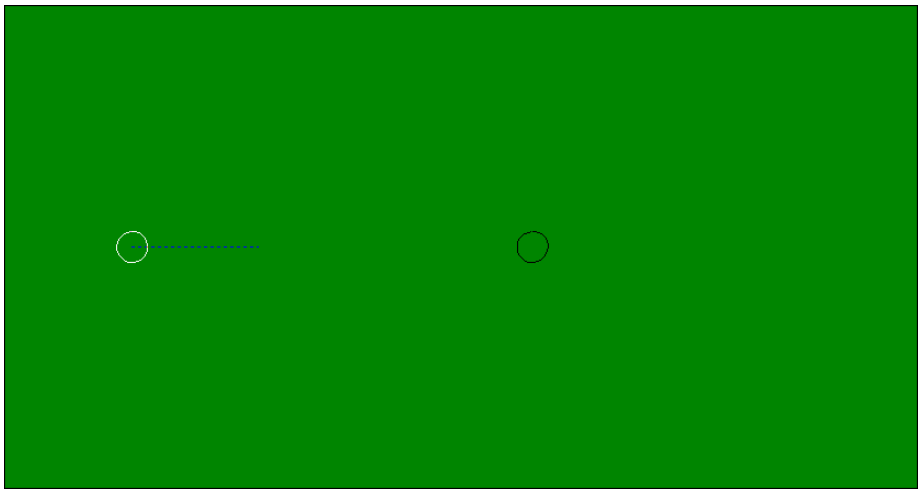
Na tomto místě bych rád ukázal chování programu při některých základních jednoduchých situacích. V každé ukázce budou 3 obrázky. Jeden před, druhý při a třetí po ukazovaném jevu.

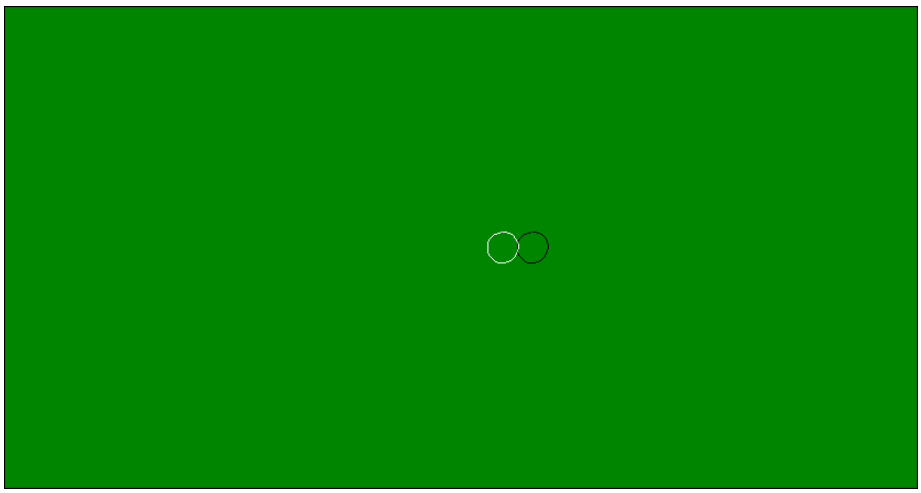
## Odraz od stěny stolu

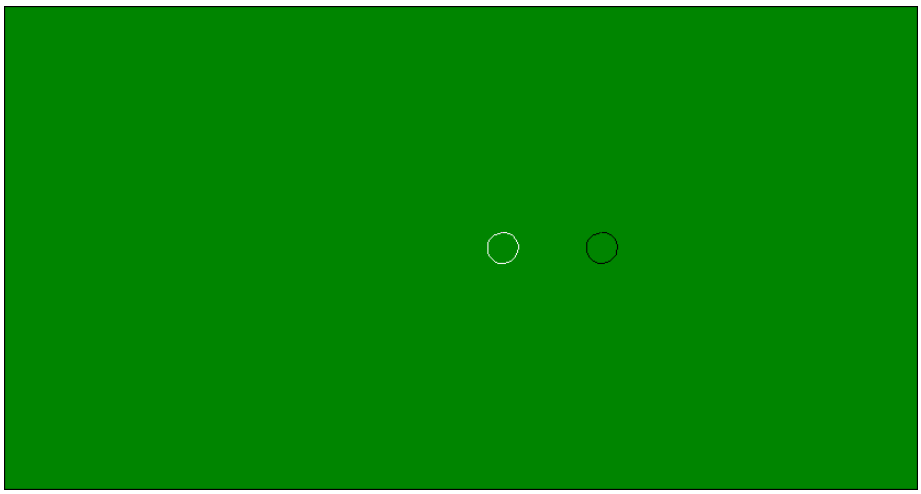


## Srážka dvou koulí 1

Srážka dvou koulí. Jedna (černá) je v klidu a druhá se k ní pohybuje po stejné přímce, na které leží spojnice středů.

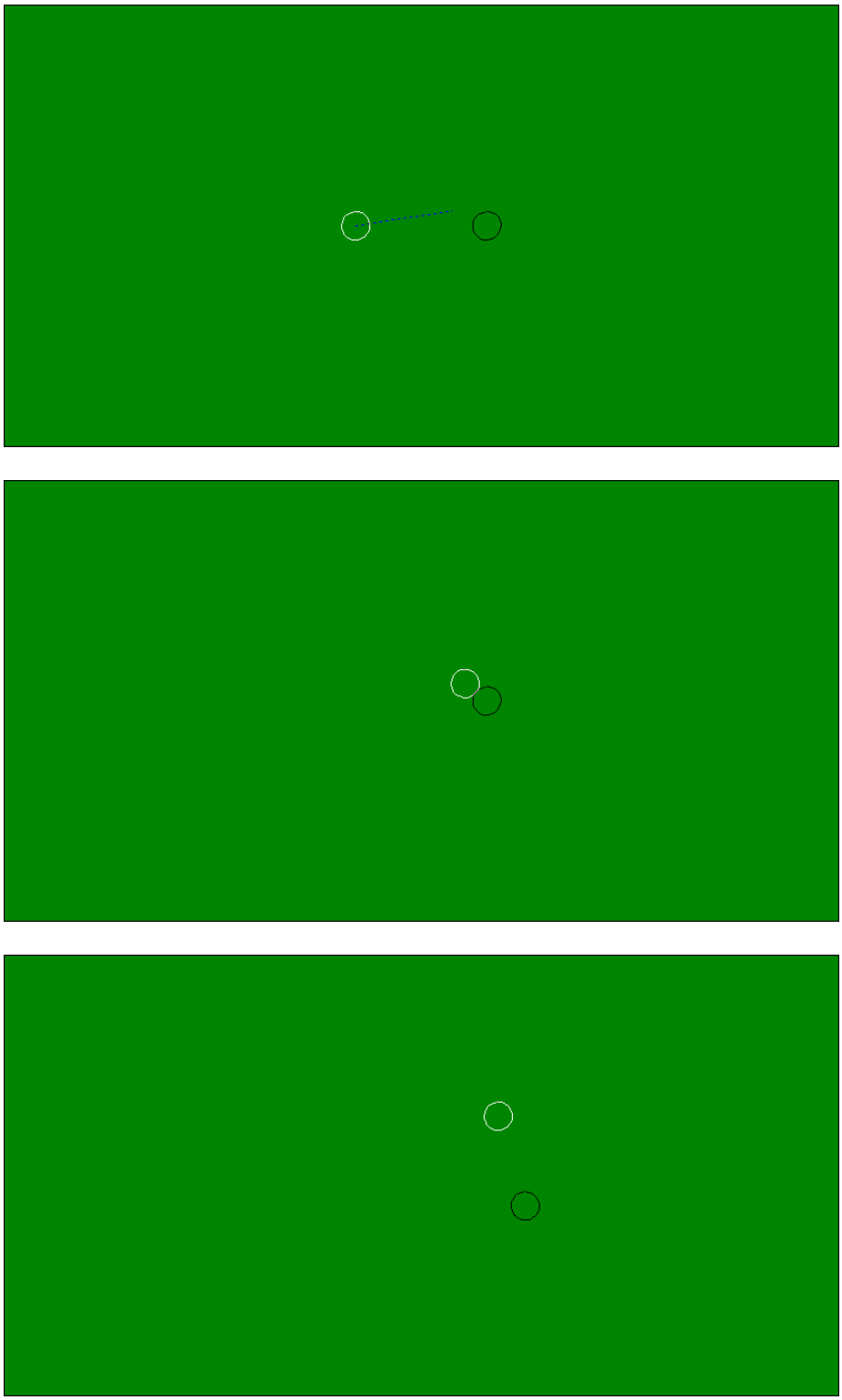






## Srážka dvou koulí 2

Opět srážka dvou koulí, z nichž jedna je v klidu (opět černá). Tentokrát ale bílá neletí přímo na černou, ale mírně šikmo.



# Závěr

## Co jsem zanedbal

V průběhu práce jsem musel v rámci zachování určité jednoduchosti zanedbat některé skutečnosti. Co se analýzy týče, tak jsem zanedbal veškeré rotace koulí (krom jejich valivého pohybu), a to, jak tyto rotace ovlivňují pohyb koule. I při jakýchkoliv srážkách jsem rotace zanedbal.

Kouli uvažuji ideální, to znamená dokonale hladkou a kulatou, stejně tak i ideálně tuhou. To znamená, že při srážkách koule nevykazuje pružnost.

V souladu se zadáním jsem veškeré srážky považoval za dokonale pružné, to znamená bez ztráty energie. A to jak srážky s okraji kulečníkového stolu, tak vzájemné srážky koulí. Zároveň jsem zanedbal případné nerovnosti na kulečníkovém stole a i ten považuji za dokonale tuhý a koule proti němu zanedbatelně lehké.

V demo programu `kulecnik` jsem též zanedbal několik věcí. Předně to jsou možné přesahy při odrazech koulí (koule se na chviličku překryjí). Dále jsem ještě zanedbal případný úder tága do koule, při kterém dochází k rozpohybování koule. Předpokládám tedy, že kouli je na začátku simulace udělena rychlost v době těsně před začátkem.

## Možná vylepšení

V reálném kulečníku se koule nechová přesně podle mnou uvedených rovnic. Koule na kulečníku se totiž nezačne ihned po úderu valit, ale dochází k prokluzu a koule nějakou dobu klouže po stole před tím, než přejde do valivého pohybu. Pro více realistickou simulaci by bylo vhodné tento pohyb popsat. Stejně tak si reálný kulečník nelze představit bez rotací koulí. Ovšem tyto rotace vážně komplikují veškeré odrazy.

V programu kulecnik je také několik problémů. Algoritmus na detekci kolizí by měl eliminovat případné přeryvy objektů při odrazech. Při odrazu na hraně stolu například tak, že pakliže by se koule ocitla už za hranou, tak by se zrcadlově přenesla zpět do vyhrazeného prostoru. Program dále nepočítá s dírami na stole, do kterých koule zapadají. Na to by algoritmus detekce kolizí mohl myslet a testovat, zda se koule již nenachází v prostoru díry a nespadne do ní. Zároveň by neměl být velký problém převést v MatLabu zobrazování scény do 3D.

Dalším možným rozšířením by byla analýza současné srážky tří koulí a následné implementování do demo programu.

## Dosažené výsledky

S přihlédnutím k zanedbaným věcem se mi podařilo popsat pohyb kulečníkové koule po stole. Zároveň jsem popsal odraz koule od hrany stolu a srážku dvou koulí. Dosažené výsledky odpovídají mým zkušenostem z kulečníkem i obecně zažitým předpokladům.
V prostředí MatLab jsem naprogramoval demo program `kulecnik`, který simuluje výsledky mé analýzy. Do tohoto programu jsem vyvinul algoritmus na pohyb koulí a na jejich vykreslování. Dále jsem vymyslel poměrně optimalizovaný (do prostředí MatLab) algoritmus na detekci kolizí koulí a následný výpočet parametrů této srážky. Výsledky tohoto programu

jsou předvedené v části *Demonstrace aplikace*. V příloze je pak několik obrázků z chování programu při náročnějším výpočtu (větší počet koulí).

# Seznam použitých zdrojů

## ***Papírové zdroje***

Prof. RNDr. Pavel, KUBEŠ, CSc.; Doc RNDr. Zdeněk, KYNCL, DrSc. **Fyzika I**, Vydavatelství ČVUT v Praze, 2003, ISBN 80-01-02671-X

Doc. Ing. Petr, HORÁČEK, CSc. **Systémy a Modely**, Vydavatelství ČVUT v Praze 2001

Karel, ZAPLATÍLEK; Bohuslav, DOŇAR. **MatLab pro začátečníky**, 2. vydání, Praha: BEN 2005, ISBN 80-7300-175-6

Karel, ZAPLATÍLEK; Bohuslav, DOŇAR. **MatLab tvorba uživatelských aplikací**, Praha BEN 2004, ISBN 80-7300-133-0

## ***Elektronické zdroje***

MatLab **Help**

MFF UK, FyzWeb: **Srážky a rotace**: http://fyzweb.mff.cuni.cz/dilna/krouzek/index.htm

Použit screenshot ze hry Billiard Simulator. http://www.darxidegames.com/

# Příloha A

Kompletní zdrojový kód vytvořeného programu

```
function kulecnik(param)

%jestlize neni zadny parametr, tak probehne inicializace
if nargin<1,
    param='inicializace';
end

switch param
    %inicializace GUI
    case 'inicializace'
        fig=figure('Name','Kulecnik Demo',...
            'MenuBar','none',...
            'Tag','fig',...
            'Position',[189 297 799 435],...
            'NumberTitle','off');

        stul=axes('Units','normalized', ...
            'Position',[0.05 0.1 0.65 0.85], ...
            'Tag','stul',...
            'XLim',[0 356],...
            'YLim',[0 187]);

        buttonOK=uicontrol('Units','normalized', ...
            'Style','pushbutton',...
            'Position',[0.81 0.05 0.09 0.06],...
            'String','OK',...
            'Tag','buttonOK',...
            'Callback','kulecnik(''buttonOK'')');

        sliderRychlost=uicontrol('Units','normalized', ...
            'Style','Slider',...
            'Position',[0.90 0.25 0.03 0.65],...
            'Tag','sliderRychlost',...
            'Min',0,'Max',8,...
            'SliderStep',[1/80 1/8],...
            'Callback','kulecnik(''sliderRychlost'')');

        sliderUhel=uicontrol('Units','normalized', ...
            'Style','Slider',...
            'Position',[0.78 0.25 0.03 0.65],...
            'Tag','sliderUhel',...
            'Min',0,'Max',360,...
            'SliderStep',[1/360 10/360],...
            'Callback','kulecnik(''sliderUhel'')');

        uicontrol('Units','normalized', ...
            'Style','text',...
            'Position',[0.775 0.91 0.04 0.03],...
            'String','Uhel',...
            'BackgroundColor',get(fig,'Color'));

        uicontrol('Units','normalized', ...
            'Style','text',...
            'Position',[0.885 0.91 0.06 0.03],...
            'String','Rychlost',...
            'BackgroundColor',get(fig,'Color'));

        editSila=uicontrol('Units','normalized', ...
            'Style','edit',...
            'Position',[0.889 0.18 0.055 0.05],...
```

```
    'String','0',...
    'Tag','editRychlost',...
    'String',num2str(round(get(sliderRychlost,'Value'))),...
    'Callback','kulecnik(''editRychlost'')');

editUhel=uicontrol('Units','normalized', ...
    'Style','edit',...
    'Position',[0.769 0.18 0.055 0.05],...
    'String','90',...
    'Tag','editUhel',...
    'String',num2str(round(get(sliderUhel,'Value'))),...
    'Callback','kulecnik(''editUhel'')');

%inicializace pole kouli
koule(1).color='white'; %barva
koule(1).x=0; %souradnice x
koule(1).y=0; %souradnice y
koule(1).v0=[0 0]; %pocatecni rychlost
koule(1).x0=0; %pocatecni poloha
koule(1).y0=0; %pocatecni poloha
koule(1).i=1; %pocet kroku od posledního odrazu
koule(1).handle=0; %handle na 'graf' dané koule

for i=2:15
    if(mod(i,2)==0)
        koule(i).color='red';
    else
        koule(i).color='yellow';
    end;
    koule(i).x=0;
    koule(i).y=0;
    koule(i).v0=[0 0];
    koule(i).x0=0;
    koule(i).y0=0;
    koule(i).i=1;
    koule(i).handle=0;
end;

koule(16).color='black';
koule(16).x=0;
koule(16).y=0;
koule(16).v0=[0 0];
koule(16).x0=0;
koule(16).y0=0;
koule(16).i=1;
koule(16).handle=0;

%pocatecni poloha kouli
koule(1).x0=50;
koule(1).y0=93.5;

koule(2).x0=205.9;
koule(2).y0=93.5;

koule(3).x0=221.6;
koule(3).y0=87.3;

koule(4).x0=237.3;
koule(4).y0=106;

koule(5).x0=221.6;
koule(5).y0=99.8;

koule(6).x0=237.3;
koule(6).y0=81;

koule(7).x0=253;
koule(7).y0=112.3;
```

```
            koule(8).x0=253;
            koule(8).y0=99.8;

            koule(9).x0=253;
            koule(9).y0=87.3;

            koule(10).x0=253;
            koule(10).y0=74.8;

            koule(11).x0=268.7;
            koule(11).y0=81;

            koule(12).x0=268.7;
            koule(12).y0=106;

            koule(13).x0=268.7;
            koule(13).y0=118.5;

            koule(14).x0=268.7;
            koule(14).y0=93.5;

            koule(15).x0=268.7;
            koule(15).y0=68.5;

            koule(16).x0=237.3;
            koule(16).y0=93.5;

            %inicializace dulezitych promenych
            konstanty.t=[0:0.01:100];%cas
            konstanty.xi=0.18;%soucinitel odporu
            konstanty.m=0.21;%hmotnost koule
            konstanty.k=konstanty.xi/konstanty.m;%konstanta
            konstanty.r=0.06; %polomer koule
            konstanty.upper_diag=triu(ones(size(koule,2))*20);%pomocna matice pro
detekci kolizi

            %pomocne promenne pro vykreslovani kouli
            konstanty.xx=sin(0:0.2:2*pi);
            konstanty.yy=cos(0:0.2:2*pi);

            konstanty.v=tf(1,[1 konstanty.k]);%prenos rychlosti
            konstanty.p=tf(1,[1 konstanty.k 0]);%prenos polohy

            %vychozi impulzni charakteristiky
            konstanty.xref=impulse(konstanty.p,konstanty.t);
            konstanty.vref=impulse(konstanty.v,konstanty.t);

            %struktura pro vykreslovani pomocne usecky pro vizualizaci
            %nastaveni parametru 'stouchu'
            miritko.delka=get(sliderRychlost,'Value')*(100/4);
            miritko.uhel=get(sliderUhel,'Value')*((2*pi)/360);
            [miritko.x miritko.y]=pol2cart(miritko.uhel, miritko.delka);

            %vykresleni pocatecniho stavu
            hold on;
            for k=1:size(koule,2)
                koule(k).x=koule(k).x0;
                koule(k).y=koule(k).y0;

                koule(k).handle=plot((6*konstanty.xx+koule(k).x),
(6*konstanty.yy+koule(k).y),koule(k).color,...
                    'LineWidth',1,...
                    'EraseMode', 'xor',...
                    'Tag', num2str(k));
            end

            %vykresleni pomocne usecky
```

```
            miritko.handle=line([koule(1).x koule(1).x+miritko.x],[koule(1).y
koule(1).y+miritko.y],'LineStyle', ':', 'EraseMode', 'xor');

            %nastaveni parametru grafu, do ktereho se koule vykresluji
            axis([0 356 0 187]);
            ax=gca;

            set(ax,'PlotBoxAspectRatio',[1.90374 1 1]);
            set(ax,'Color',[0 0.52 0]);
            box on;
            set(ax,'XTick',[]);
            set(ax,'YTick',[]);

            %ulozeni struktur z duvodu predavani parametru
            set(ax,'UserData',koule);
            set(sliderUhel,'UserData',miritko);
            set(fig,'UserData',konstanty);

        %callback cast pro editovaci pole Uhel. Nastavi se proslusni slider na
        %odpovidajici hodnotu (po kontrole zadane hodnoty) a upravi se
        %parametry pomocne usecky, ktera se prekresli
        case 'editUhel'
            val=str2double(get(findobj('Tag','editUhel'),'String'));
            if(isnumeric(val) && (val>=get(findobj('Tag','sliderUhel'),'Min')) &&
(val<=get(findobj('Tag','sliderUhel'),'Max')))
                set(findobj('Tag','sliderUhel'),'Value',val);
            end

            miritko=get(findobj('Tag','sliderUhel'), 'UserData');
            miritko.uhel=val*((2*pi)/360);
            [miritko.x miritko.y]=pol2cart(miritko.uhel, miritko.delka);
            graf=gca;
            koule=get(graf,'UserData');
            set(miritko.handle, 'XData',[koule(1).x koule(1).x+miritko.x],
'YData',[koule(1).y koule(1).y+miritko.y]);
            set(findobj('Tag','sliderUhel'),'UserData',miritko);

        %callback cast pro editovaci pole Rychlost. Obdobne operace jako v
        %predchozim pripade
        case 'editRychlost'
            val=str2double(get(findobj('Tag','editRychlost'),'String'));
            if(isnumeric(val) && (val>=get(findobj('Tag','sliderRychlost'),'Min'))
&& (val<=get(findobj('Tag','sliderRychlost'),'Max')))
                set(findobj('Tag','sliderRychlost'),'Value',val);
            end

            miritko=get(findobj('Tag','sliderUhel'), 'UserData');
            miritko.delka=val*(100/4);
            [miritko.x miritko.y]=pol2cart(miritko.uhel, miritko.delka);
            graf=gca;
            koule=get(graf,'UserData');
            set(miritko.handle, 'XData',[koule(1).x koule(1).x+miritko.x],
'YData',[koule(1).y koule(1).y+miritko.y]);
            set(findobj('Tag','sliderUhel'),'UserData',miritko);

         %callback cast pro slider Uhel. Zaokrouhlena hodnota se nastavi do
         %odpovidajiciho editacniho pole a zmeni se hodnoty pomocne usecky,
         %ktera se prekresli
        case 'sliderUhel'

set(findobj('Tag','editUhel'),'String',num2str(round(get(findobj('Tag','sliderUhel'
),'Value'))));

set(findobj('Tag','sliderUhel'),'Value',str2num(get(findobj('Tag','editUhel'),'Stri
ng')));

            miritko=get(findobj('Tag','sliderUhel'), 'UserData');
            miritko.uhel=get(findobj('Tag','sliderUhel'),'Value')*((2*pi)/360);
```

```
            [miritko.x miritko.y]=pol2cart(miritko.uhel, miritko.delka);
            graf=gca;
            koule=get(graf,'UserData');
            set(miritko.handle, 'XData',[koule(1).x koule(1).x+miritko.x],
'YData',[koule(1).y koule(1).y+miritko.y]);
            set(findobj('Tag','sliderUhel'),'UserData',miritko);

        %callback cast pro slider Rychlost.
        case 'sliderRychlost'

set(findobj('Tag','editRychlost'),'String',num2str(get(findobj('Tag','sliderRychlos
t'),'Value')));

set(findobj('Tag','sliderRychlost'),'Value',str2num(get(findobj('Tag','editRychlost
'),'String')));

            miritko=get(findobj('Tag','sliderUhel'), 'UserData');
            miritko.delka=get(findobj('Tag','sliderRychlost'),'Value')*(100/4);
            [miritko.x miritko.y]=pol2cart(miritko.uhel, miritko.delka);
            graf=gca;
            koule=get(graf,'UserData');
            set(miritko.handle, 'XData',[koule(1).x koule(1).x+miritko.x],
'YData',[koule(1).y koule(1).y+miritko.y]);
            set(findobj('Tag','sliderUhel'),'UserData',miritko);

        %callback pro tlacitko OK. Spousti se simulace
        case 'buttonOK'

            %zasednuti vsech ovladacih prvku. V rubehu simulace nejsou potreba
            set(findobj('Tag','buttonOK'),'Enable','off');
            set(findobj('Tag','sliderUhel'),'Enable','off');
            set(findobj('Tag','sliderRychlost'),'Enable','off');
            set(findobj('Tag','editUhel'),'Enable','off');
            set(findobj('Tag','editRychlost'),'Enable','off');

            graf=gca;
            fig=get(graf,'Parent');

            %prevzeti predavanych parametru
            koule=get(graf,'UserData');
            miritko=get(findobj('Tag','sliderUhel'), 'UserData');
            konstanty=get(fig, 'UserData');

            %zjisteny vyhozi hodnoty ruchlosti pro bilou kouli
            [koule(1).v0(1), koule(1).v0(2)]=pol2cart(miritko.uhel,
miritko.delka*4);

            %pomocna usecka neni v prubehu simulace potreba
            set(miritko.handle, 'XData',[], 'YData',[]);

            %vypocet aktualnich vzdalenosti kouli (pro pozdejsi porovnavani)
            [x_stejne y_stejne]=meshgrid(cat(2,koule(1,:).x),cat(2,koule(1,:).y));
            x_stejne=x_stejne';
            distance2=sqrt(((x_stejne'-x_stejne).^2)+((y_stejne'-y_stejne).^2));
            distance2=tril(distance2)+konstanty.upper_diag;

            %vlastní simulacni smycka
            while(1)
                %pro kazdou kouli...
                for k=1:size(koule,2)
                    %...se vypocte nova poloha...

koule(k).x=koule(k).x0+koule(k).v0(1)*konstanty.xref(koule(k).i);

koule(k).y=koule(k).y0+koule(k).v0(2)*konstanty.xref(koule(k).i);

                    %...a hned se zmeni data v grafu...
                    set(koule(k).handle,'XData',(6*konstanty.xx+koule(k).x),...
```

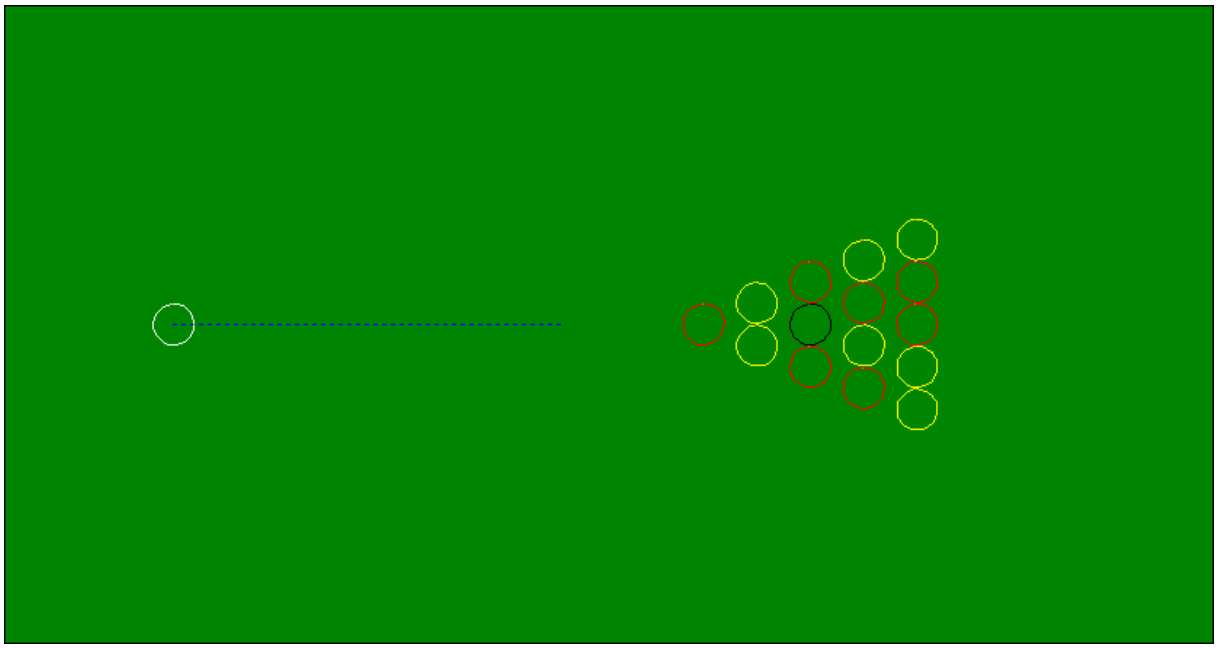
```
                        'YData',(6*konstanty.yy+koule(k).y));

                    %...dale probehne detekce kolizi ze stenou a propadne
                    %reseni teto kolize...

                    %leva a prava stena
                    if((koule(k).x>(356-
(100*konstanty.r)))||(koule(k).x<(100*konstanty.r)))
                        koule(k).v0(1)=koule(k).v0(1)*konstanty.vref(koule(k).i);
                        koule(k).v0(2)=koule(k).v0(2)*konstanty.vref(koule(k).i);

                        koule(k).x0=koule(k).x;
                        koule(k).y0=koule(k).y;
                        koule(k).v0(1)=(-1)*koule(k).v0(1);

                        koule(k).i=1;
                    end

                    %horni a dolni stena
                    if((koule(k).y>=(187-
(100*konstanty.r)))||(koule(k).y<=(100*konstanty.r)))
                        koule(k).v0(1)=koule(k).v0(1)*konstanty.vref(koule(k).i);
                        koule(k).v0(2)=koule(k).v0(2)*konstanty.vref(koule(k).i);

                        koule(k).x0=koule(k).x;
                        koule(k).y0=koule(k).y;
                        koule(k).v0(2)=(-1)*koule(k).v0(2);

                        koule(k).i=1;
                    end
                end

                %...dále se testuje kolize kouli...

                %...spocita se matice vzajemnych vzdalenosti kouli...
                [x_stejne
y_stejne]=meshgrid(cat(2,koule(1,:).x),cat(2,koule(1,:).y));
                x_stejne=x_stejne';
                distance=sqrt(((x_stejne'-x_stejne).^2)+((y_stejne'-y_stejne).^2));
                distance=tril(distance)+konstanty.upper_diag;

                %...a zjistí se jestli se nejake koule k sobe nepriblizili na
                %mene, nez dvojnasobek polomeru...
                [x_found y_found]=find(distance<(2*konstanty.r*100));

                %...vypocte se diference poloh ve dvou po sobe nasledujicich
                %krocich limulace...
                d_dist=distance-distance2;

                %...pro kazdou kolizi se spocita odraz
                for k=1:size(x_found)
                    if(d_dist(x_found(k),y_found(k))>=0)
                        %...jestlize je diference predchazejiciho a tohoto kroku
                        %vetsi, nebo rovna nule, tak koule vuci sobe bud stoji,
                        %a nebo se vzdaluji. V tom pripade odraz nenastava...
                        continue;
                    end

                    %...jestlize odraz skutecne nastal, tak se spocitaji veskere
                    %parametry odrazu a vypoctou se jednotlive ryclosti obou
                    %kouli...

v01x=koule(x_found(k)).v0(1)*konstanty.vref(koule(x_found(k)).i);

v01y=koule(x_found(k)).v0(2)*konstanty.vref(koule(x_found(k)).i);


v02x=koule(y_found(k)).v0(1)*konstanty.vref(koule(y_found(k)).i);
```

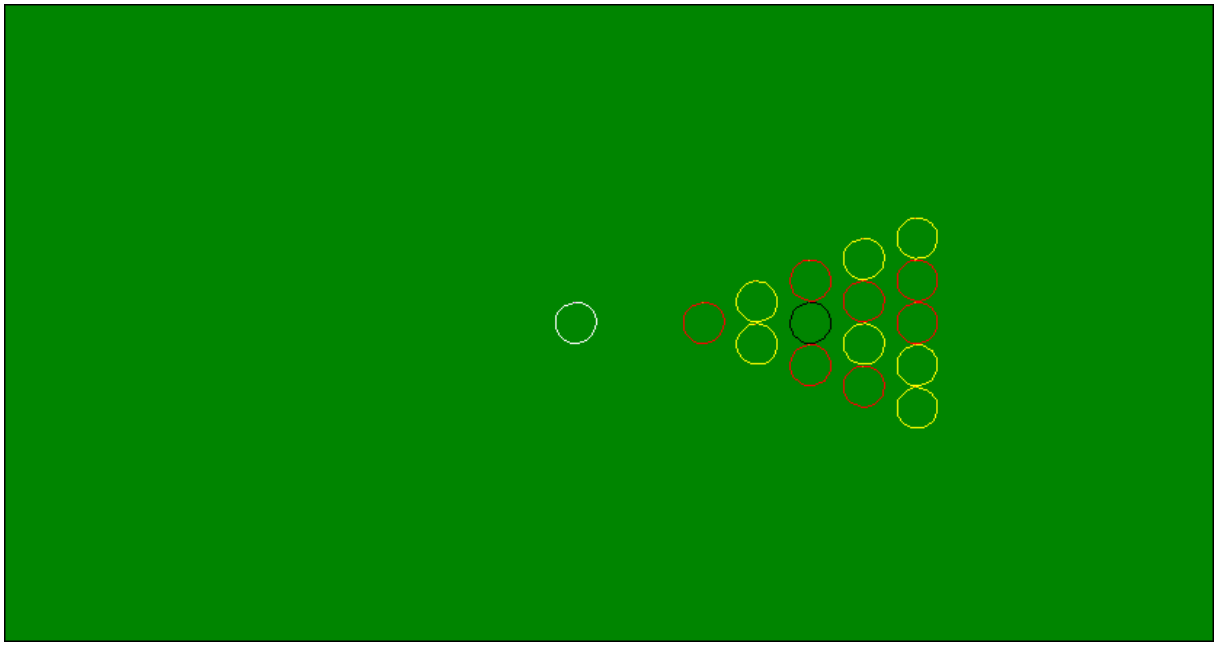
```
v02y=koule(y_found(k)).v0(2)*konstanty.vref(koule(y_found(k)).i);

                    v01=sqrt(v01x^2+v01y^2);
                    v02=sqrt(v02x^2+v02y^2);

                    %uhly pro kouli 1
                    deltax1=koule(y_found(k)).x-koule(x_found(k)).x;
                    deltay1=koule(y_found(k)).y-koule(x_found(k)).y;

                    alfa1=cart2pol(deltax1,deltay1);%argument osy stretu
                    alfa1=mod(alfa1,2*pi);

                    alfa2=cart2pol(v01x,v01y);%argument rychlosti koule 1
                    alfa2=mod(alfa2,2*pi);

                    alfa=alfa1-alfa2;
                    alfa=sign(alfa)*alfa;

                    switch(sign(alfa2-alfa1))
                        case 1
                            alfa3=alfa1+pi/2;% uhel rychlosti v1

                        case 0
                            alfa3=0;

                        case -1
                            alfa3=2*pi-(pi/2-alfa1);
                    end

                    %uhly pro kouli 1
                    deltax2=koule(x_found(k)).x-koule(y_found(k)).x;
                    deltay2=koule(x_found(k)).y-koule(y_found(k)).y;

                    beta1=cart2pol(deltax2,deltay2);%argument osy stretu
                    beta1=mod(beta1,2*pi);

                    beta2=cart2pol(v02x,v02y);%argument rychlosti koule2
                    beta2=mod(beta2,2*pi);

                    beta=beta1-beta2;
                    beta=sign(beta)*beta;

                    switch(sign(beta2-beta1))
                        case 1
                            beta3=beta1+pi/2;

                        case 0
                            beta3=0;

                        case -1
                            beta3=2*pi-(pi/2-beta1);
                    end

                    %rychlst v1 pro kouli 1
                    v11x=v01*sin(alfa)*cos(alfa3);
                    v11y=v01*sin(alfa)*sin(alfa3);

                    %rychlst v2 pro kouli 1
                    v21x=v01*cos(alfa)*cos(alfa1);
                    v21y=v01*cos(alfa)*sin(alfa1);

                    %rychlst v1 pro kouli 2
                    v12x=v02*sin(beta)*cos(beta3);
                    v12y=v02*sin(beta)*sin(beta3);

                    %rychlst v2 pro kouli 2
                    v22x=v02*cos(beta)*cos(beta1);
```

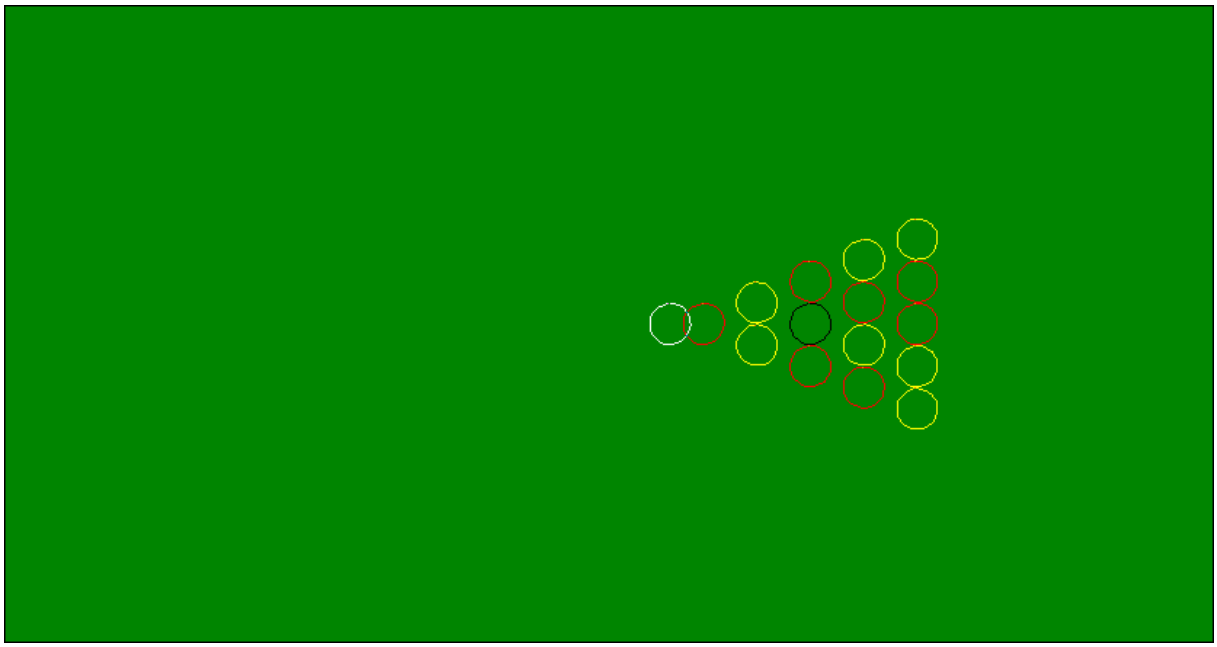
```
                    v22y=v02*cos(beta)*sin(beta1);

                    koule(y_found(k)).v0(1)=v12x+v21x;
                    koule(y_found(k)).v0(2)=v12y+v21y;

                    koule(x_found(k)).v0(1)=v22x+v11x;
                    koule(x_found(k)).v0(2)=v22y+v11y;

                    %nastavení novych vychozich hodnot pro obe koule
                    koule(x_found(k)).i=1;
                    koule(y_found(k)).i=1;

                    koule(x_found(k)).x0=koule(x_found(k)).x;
                    koule(x_found(k)).y0=koule(x_found(k)).y;

                    koule(y_found(k)).x0=koule(y_found(k)).x;
                    koule(y_found(k)).y0=koule(y_found(k)).y;
                end

                %...pro vypocet diference se uchova momentalni matice vzdalenosti
                %mezi koulemi...
                distance2=distance;

                %...dale se pocita celkova velikost rychlosti cele soustavy,
                %aby se dal detekovat konec simulace...
                v_celkova=0;
                for k=1:size(koule,2)
                    vx=koule(k).v0(1)*konstanty.vref(koule(k).i);
                    vy=koule(k).v0(2)*konstanty.vref(koule(k).i);
                    koule(k).i=koule(k).i+1;
                    v_celkova=v_celkova+sqrt(vx^2+vy^2);
                end

                %...jestlize klesne celkova rychlost pod danou mez, tak je
                %simulace zastavena...
                if(v_celkova<5)
                    disp('Konec simulace');

                    %pro kazdou kouli se ulozi aktualni hodnoty polohy a
                    %rychlost se nastavi na nuly...
                    for k=1:size(koule,2)
                        koule(k).x0=koule(k).x;
                        koule(k).y0=koule(k).y;
                        koule(k).v0=[0 0];
                        koule(k).i=1;
                    end

                    %...znovu se aktivuje GUI...
                    set(findobj('Tag','buttonOK'),'Enable','on');
                    set(findobj('Tag','sliderUhel'),'Enable','on');
                    set(findobj('Tag','sliderRychlost'),'Enable','on');
                    set(findobj('Tag','editUhel'),'Enable','on');
                    set(findobj('Tag','editRychlost'),'Enable','on');

                    %...zobrazi se pomocna usecka...
                    set(miritko.handle, 'XData',[koule(1).x koule(1).x+miritko.x],
'YData',[koule(1).y koule(1).y+miritko.y]);

                    %...a pole kouli se ulozi...
                    set(graf,'UserData',koule);

                    %tim simulace konci a uzivatel muze zadavat dalsi 'stouch'
                    break;
                end
                pause(0.01);
            end
    end
    end
```

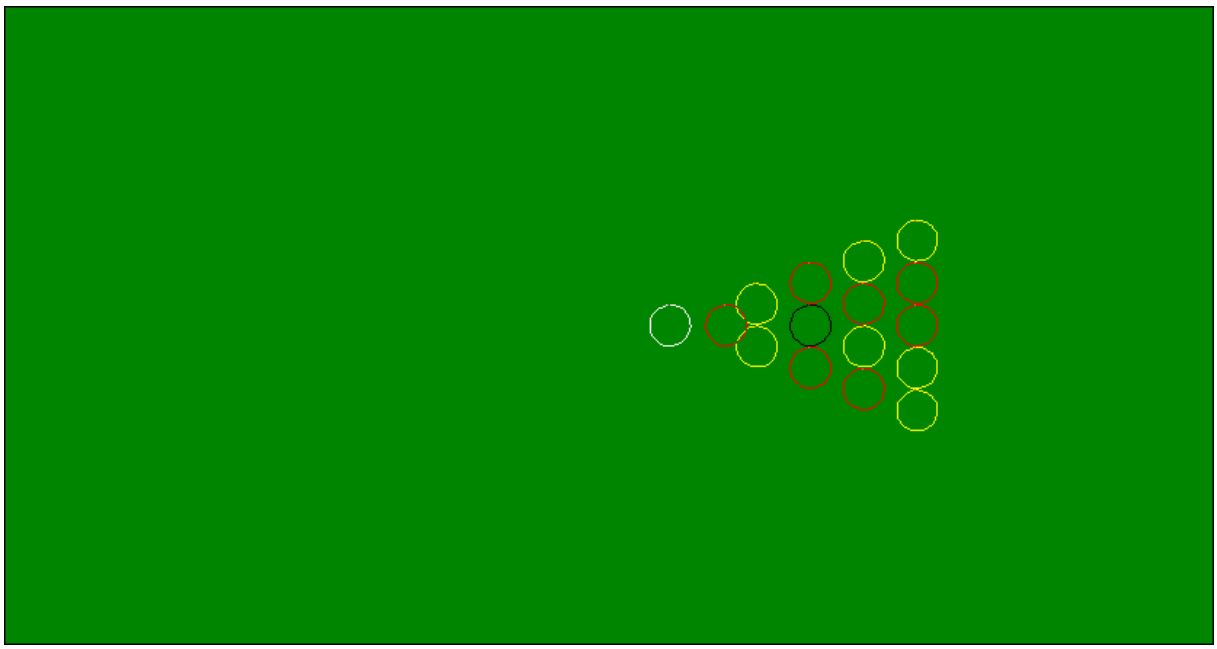
# Příloha B

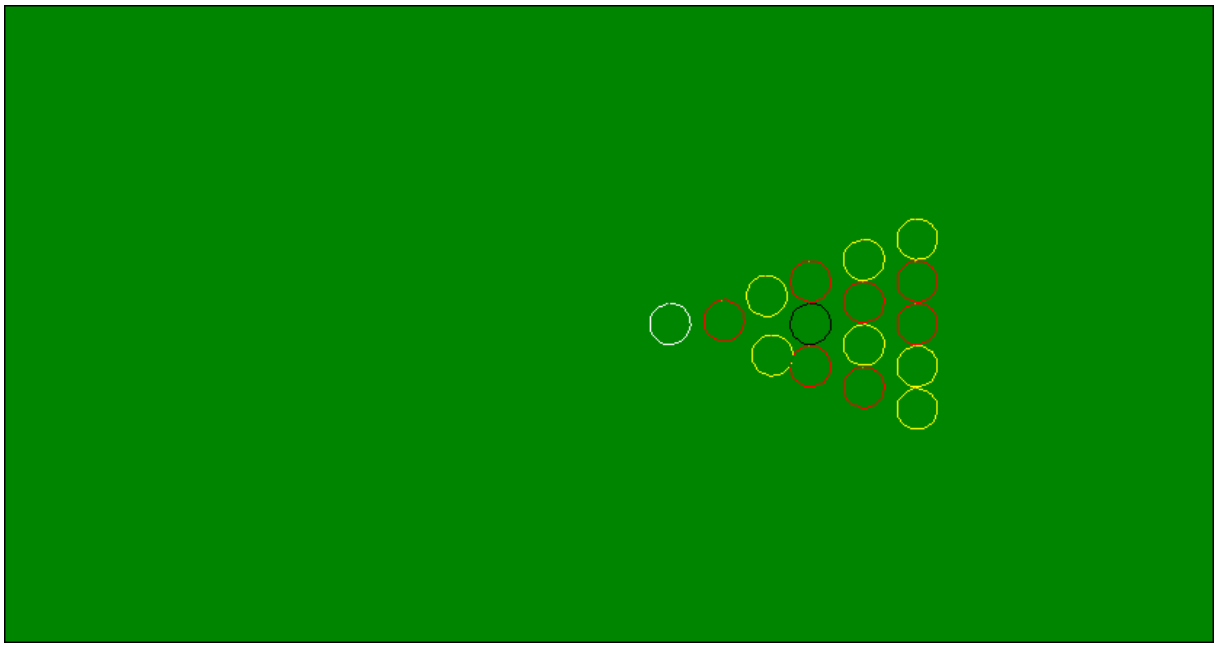
Několik obrázků z průběhu složitější simulace.

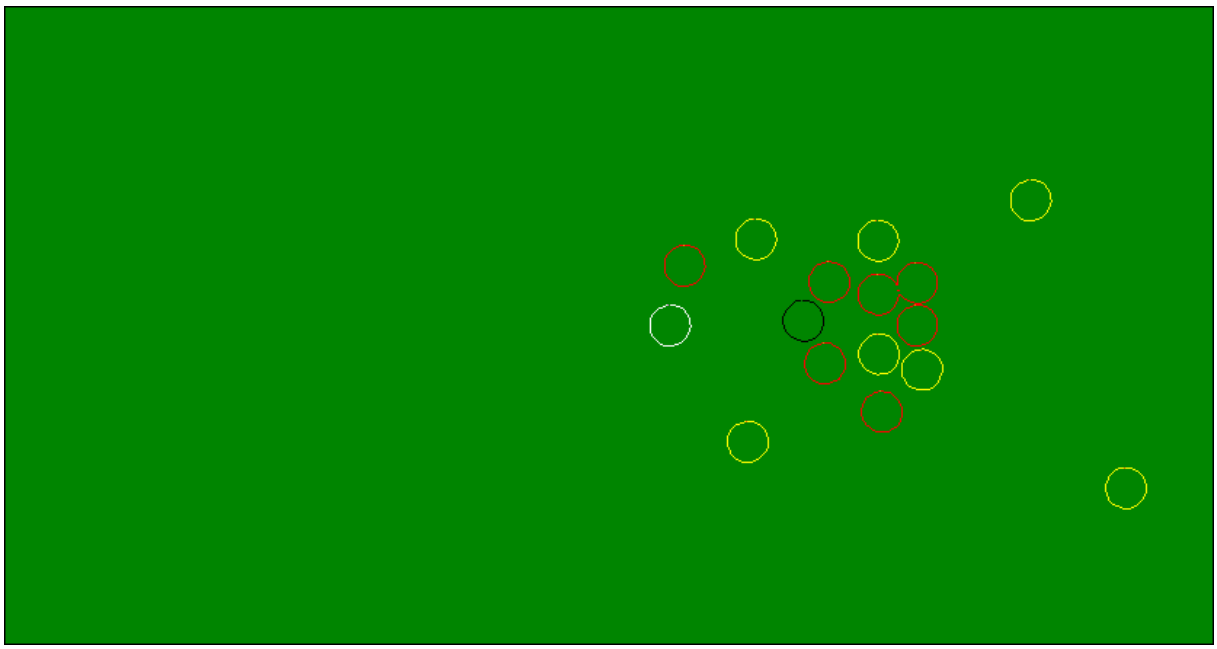


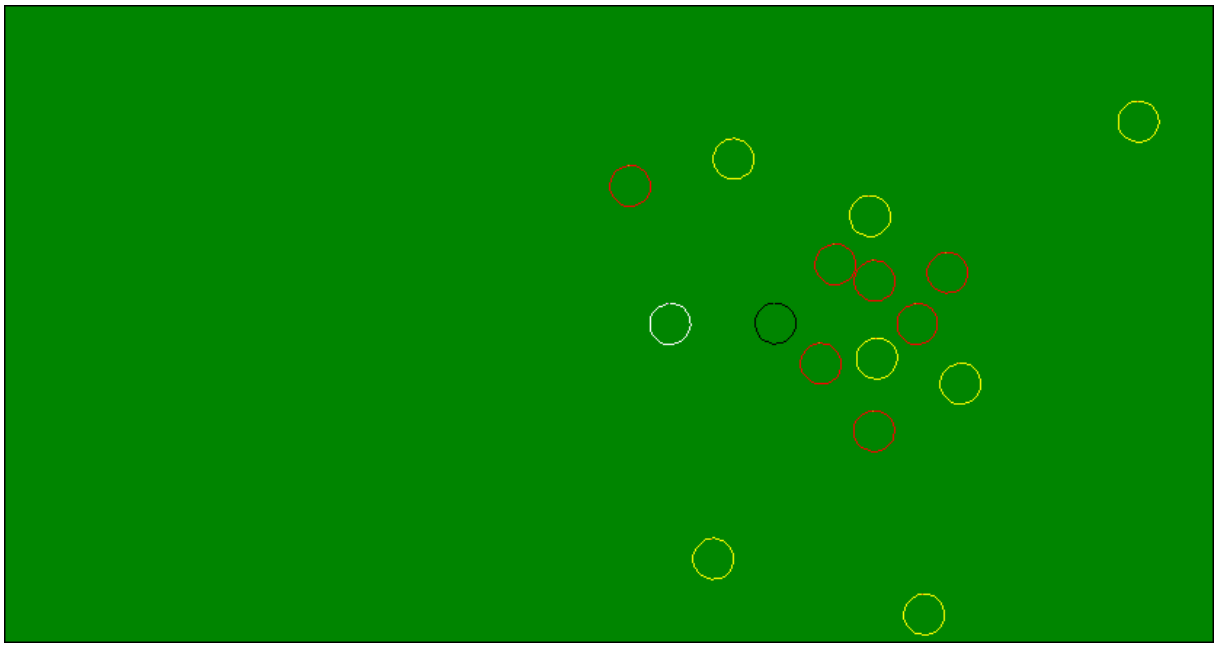


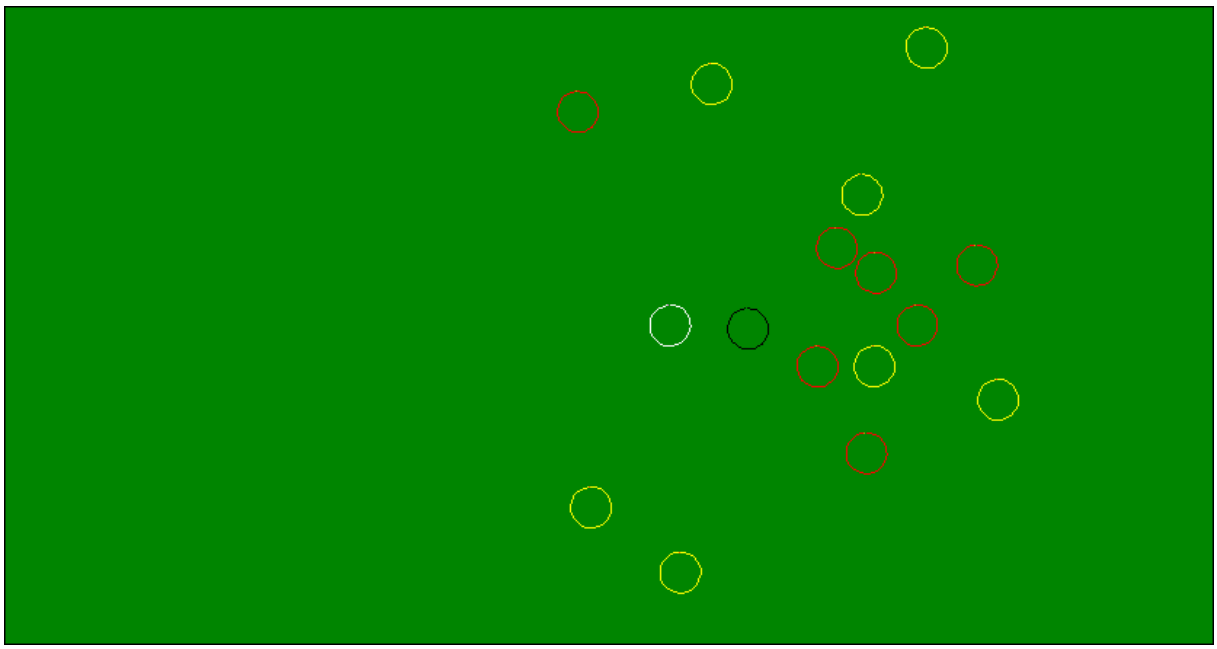












# Příloha C

Přílohou C je přiložené CD. Na něm je kompletní zdrojový kód programu, kompletní sekvence demonstračních obrázků a elektronická podoba této práce.