LAPPEENRANNAN TEKNILLINEN YLIOPISTO
School of Engineering Science
Laskennallisen tekniikan koulutusohjelma
Kandidaatintyö

*Arna Hyvärinen*

**Fysiikkasimulaatiot videopeleissä**

Ohjaaja: Jouni Sampo

**TIIVISTELMÄ**

Lappeenrannan teknillinen yliopisto

School of Engineering Science

Laskennallisen tekniikan koulutusohjelma

Arna Hyvärinen

**Fysiikkasimulaatiot videopeleissä**

Kandidaatintyö

2017

20 sivua, 5 kuvaa, 0 taulukkoa, 2 liitettä

Ohjaaja: Jouni Sampo

Avainsanat: fysiikkamoottorit; simulaatio; videopelit; mekaniikka; virtauslaskenta;

Työssä kartoitetaan saatavilla olevaa tietoa fysiikkamoottoreista, ja miten niitä käytetään videopeleissä. Aihe on rajattu mekaanisen liikkeen ja virtauksen simuloimiseen.

Fysiikkamoottori on videopelikontekstissa pelimoottorin osaohjelma, joka laskee pelissä tapahtuvia fysikaalisia ilmiöitä numeerisesti. Esimerkiksi kappaleiden yhteentörmäys ja sen seurakset lasketaan fysiikkamoottorilla. Pelimoottorin täytyy pystyä reagoimaan pelaajan toimintaan realliajassa. Tämän takia laskenta täytyy tehdä numeerisesti, ja suunnitella mahdollisimman kevyeksi suorittaa. Se, kuinka tarkkaan simulaatio vastaa todellisuutta, on suunniteltava pelin tarpeiden kohtaisesti.

Työtä varten koodataan kaksi omaa simulaatiota. Ensimmäinen on yksinkertainen mekaniikkasimulaattori, jossa joukko palloja kimpoilee laatikossa. Toinen simulaatio kuvaa virtausta, simuloimalla nestettä partikkelijoukkona. Simulaation vaikeuksia on törmäyksestä irtautuminen, ja etenkin virtaussimulaation laskentateho. Hyvä partikkelisimulaatio vaatii niin suuren määrän palloja, että tämän ohjelman ajaminen tavanomaisella tietokoneella on epämiellyttävän hidasta. Raskaudesta johtunee se, että peleissä nestettä harvoin simuloidaan realistisesti.

# Sisältö

## Liitteet

Liite 1: Palloja laatikossa MATLAB-koodi

Liite 2: Malja MATLAB-koodi

**Symboli- ja lyhenneluettelo**

| | |
|---|---|
| $\boldsymbol{a}$ | kiihtyvyys |
| $\boldsymbol{F}$ | voima |
| $\boldsymbol{g}$ | putoamiskiihtyvyys |
| $\boldsymbol{I}$ | impulssi |
| $m$ | massa |
| $p$ | paine |
| $s$ | paikka |
| $t$ | aika |
| $\boldsymbol{v}$ | nopeus |
| $\boldsymbol{v}'$ | nopeus törmäyksen jälkeen |
| $\boldsymbol{x}$ | paikkakoordinaatti |
| | |
| $\nu$ | kinemaattinen viskositeetti |
| $\rho$ | tiheys |
| $\omega$ | vaimennuskerroin |
| | |
| SPH | Smoothed Particle Hydrodynamics |

# 1 JOHDANTO

## 1.1 Fysiikkasimulaatioista yleisesti

Fysiikkasimulaatiot ovat erittäin hyödyllisiä todellisessa maailmassa tekniikan alalla, sillä niiden avulla voidaan ennustaa fysikaalisten ilmiöiden käyttäytymistä ja optimoida toimintaa. Videopeleissä simulaatiot eivät ole yhtä realistisia kuin oikean maailman simuloimiseen tarkoitetut mallit. Ne täytyy suunnitella tarpeeksi kevyiksi laskea, jotta peli voi reagoida pelaajaan reaaliajassa.

Tärkein aspekti pelien fysiikkasimulaatioissa on pelin miellyttävyys: realistisia simulaatioita käyttämällä voidaan tehdä realistisia pelejä. Tämä on kuitenkin kaksiteräinen miekka: realistinen peli ei välttämättä ole miellyttävä pelata. Esimerkiksi biljardivideopelissä itse lajiin perehtymätön pelaaja odottaa intuitiivisesti pallojen kimpoilevan seinistä peilikulmassa. Todellisuudessa näin ei ole, vaan mm. pallon kierre vaikuttaa kulmaan. Mutta jos pallo käyttäytyy eri tavalla kuin pelaaja odottaa, peli ei tunnu responsiiviselta, ja pelin hallinta tuntuu pelaajasta huonolta. Toisaalta jos pelaaja räjäyttää pommin ruohikossa, ja heinät heilahtavat ilmavirran mukana, pelaaja kokee vaikuttuneensa maailmaan vahvemmin kuin jos heinä pysyisi staattisesti pystyssä. Tämän lisäksi esimerkiksi taustalla todellisuuden mukaisesti virtaava vesi lisää pelimaailman realistisuutta, jolloin siihen on helpompi uppoutua. Ja tietenkin alan harrastajat ihailevat esteettisesti miellyttävää simulaatiota.

## 1.2 Tutkimusongelma, tavoitteet ja rajaus

Tavoitteena työssä on kartoittaa vapaasti saatavilla olevaa tietoa videopelien fysiikkasimulaattoreista. Työ keskittyy kappaleen mekaanisen liikkeen simulaatioon sekä veden ja virtauksen simulaatioon, tässä järjestyksessä. Työssä perehdytään siihen, miten nämä voidaan toteuttaa, sekä teoriamielessä että käytännössä koodaamalla.

## 1.3 Tutkimusmetodologia ja -järjestelyt

Tässä työssä simulidaan muutamia fysiikan perusilmiöitä.

Käytännön kokemuksen saamiseksi työn lopuksi toteutetaan yksinkertainen, kaksiulotteinen mekaniikkamoottori, jolla simuloidaan pistemäisten kappaleiden liikettä. Myös nesteen lii-

kettä simuloidaan. Moottori tehdään MATLAB-ohjelmistolla.

Tavoitteena on tutkia fysiikkasimulaatioita muutaman esimerkin kautta. Nämä ovat pistemäisen kappaleen mekaniikka sekä veden virtaus. Tarkoituksena on selvittää yhtälöt, joilla edellä mainittujen ilmiöiden mallintaminen tapahtuu, ja miten nämä yhtälöt ratkaistaan. Sen jälkeen esitellään oma malli ja sen toimintaa demonstroidaan.

# 2 MALLIN TAUSTA/TEORIA

## 2.1 Fysiikkamoottori

Fysiikkamoottori on pelimoottorin osaohjelma, joka mallintaa fysiikkaa esimerkiksi laskemalla simuloitavien kappaleiden liikkeet. Pelifirmat yleensä eivät kehitä pelimoottoreitaan itse, koska resursseja käytetään mielluummin itse pelin tekemiseen. Tunnettuja pelimoottoreita ovat Unity (Unity 2018), Unreal Engine (UnrealEngine 2018) ja Godot (Godot 2018).

Tärkeä ominaisuus on tehokkuus, jossa vastaan tuleekin pelimoottoreiden suunnittelun keskeinen pulma: mitä tarkemmin fysiikkaa mallinnetaan, sitä raskaampaa sitä on laskea. Koska peleissä tarvitaan tilan reaaliaikaista laskentaa, mallinnusta on pakko yksinkertaistaa laitteiden tehon takia.

Pelimoottorin ei ole järkevää laskea tapahtumaketjua analyyttisesti jatkuvana, koska sen täytyy ottaa huomioon käyttäjän toiminta. Numeerinen ratkaisu on kevyempi. Moottori laskee tapahtumia ”peliluuppeina”: toistuvina kierroksina, joissa tekoälyn toiminta, pelilogiikka, fysiikkasimulaatio ynnä muu lasketaan ja päivitetään (Gregory 2009).

On yleistä, että pelien kehittäjien pelimoottorit käyttävät toisaalta ostettua väliohjelmistoa monimutkaisten asioiden simuloimiseksi, sillä hyvän ohjelmiston luominen itse veisi liikaa resursseja ja vaatisi liian syvällistä asiantuntemusta (Nilson et al. 2007). Tällaisia ohjelmistoja on esimerkiksi fysiikkamoottorit, kappaleiden hajoamista käsittelevät tuhoutumismoottorit ja vaikeasti simuloitavat yksityiskohdat kuten kankaan liike ja repeäminen (Havok 2017).

Se, kuinka realistista mekaniikaa fysiikkamoottorin tarvitsee simuloida, riippuu pelistä. Jos pelaaja ampuu pelissä vihollista haulikolla, ja vihollinen lentää ilmaan, epärealistinen tapahtumaketju ruokkii voimafantasiaa. Tosiaalta, jos fysiikkapohjaisessa pulmapelissä (*Portal* on hyvä esimerkki) kappaleet käyttäytyvät arvaamattomasti, peliä on vain ärsyttävää pelata.

## 2.2 Mekaniikka

Mekaniikka seuraa Newtonin lakeja, ja käsittelee kappaleiden keskinäistä vuorovaikutusta. Mekaniikkaan kuuluu erinäisiä alahaaroja, kuten dynamiikka, jossa tutkitaan kappaleiden välisiä voimia ja niiden vaikutusta kappaleiden liikkeeseen. Pelkkää liikkeen tarkastelua kutsutaan kinematiikaksi. Yksinkertaisuuden vuoksi kappaleita kuvataan usein massakeskipisteen avulla (Laine 2007).

Dynamiikan liikkeen määrittelemiseksi tarvitaan Newtonin lakeja. Newtonin ensimmäinen laki määrittelee massan hitauden: lepotilassa oleva kappale pyrkii pysymään lepotilassa, liiketilassa oleva kappale pyrkii pysymään liiketilassa. Muutosta liikkeessä taas kuvaa Newtonin toinen laki:

$$\boldsymbol{F} = m\boldsymbol{a}$$

jossa $\boldsymbol{F}$ on kappaleeseen vaikuttava kokonaisvoima, $m$ kappaleen massa ja $\boldsymbol{a}$ kappaleen kiihtyvyys. Näillä laeilla saadaan tehokkaasti kuvailtua yhden kappaleen liike. Yhtälöllä voidaan ottaa huomioon miten mm. painovoima, kitkavoima ja muut ulkoiset voimat vaikuttavat kappaleen liikkeeseen. Jos käsitellään useampaa kappaletta, tarvitaan Newtonin kolmas laki. Se on voiman ja vastavoiman laki, ja se on törmäystilanteessa keskeisin laki. Sen mukaan kahden kappaleen väliset voimat ovat yhtä suuria, mutta vastakkaissuuntaisia. Kahden kappaleen törmäyksessä tarvitaan lisäksi liikemäärän eli impulssin säilymisyhtälö. Liikemäärän muutoksen määritelmä on:

$$\boldsymbol{I} = m\Delta\boldsymbol{v} = \boldsymbol{F}\Delta t \tag{1}$$

jossa $\boldsymbol{I}$ on impulssi, $\boldsymbol{v}$ nopeus ja $t$ aika. Liikkeen yhtälö jossa paikkaa $s$ kuvataan suhteessa aikaan $t$ on

$$s(t) = s_0 + v(t)t + \frac{1}{2}a(t)t^2$$

Jos nopeus ja kiihtyvyys olisivat vakiot, tämä olisi ratkaistavissa kevyesti. Mutta nämä muuttuvat jatkuvasti. Törmäysten tapahtuessa $v(t)$ ja $a(t)$ muuttuvat, ja vuorovaikutuksen takia tarkasta yhtälöstä tulee liiallisen raskas. Onneksi tarkka yhtälö on useimmissa simuloiduissa tapauksessa tarpeeton. Pelimoottorin ei tarvitse tietää kappaleiden paikkaa absoluuttisen täsmällisesti. Summittainen, nopeasti laskettava malli on useimmissa tapauksessa parempi kuin tarkka mutta hidas tai liian suurta laskutehoa vaativa malli. Siksi yhtälö yksinkertaistaan numeeriseen versioonsa. Ensin johdetaan paikan muutos.

Nopeuden määritelmästä

$$v = \frac{ds}{dt}$$

saadaan paikan muutos kun nopeus on vakio:

$$ds = vdt$$
$$\int_0^{\Delta s} ds = \int_0^{\Delta t} vdt$$
$$\Delta s \approx v\Delta t$$

Nopeuden muutos saadaan samalla tavalla kiihtyvyydestä:

$$a = \frac{dv}{dt}$$
$$dv = adt$$
$$\int_0^{\Delta v} dv = \int_0^{\Delta t} adt$$
$$\Delta v \approx a\Delta t$$

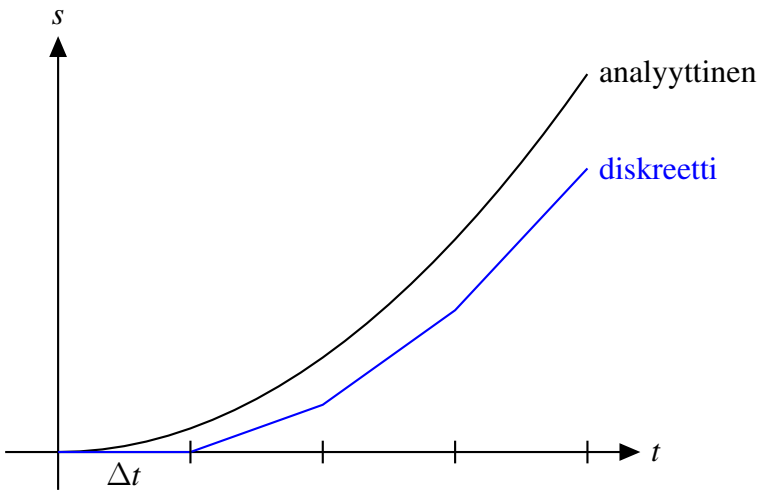
Tästä saadaan mallit kappaleen paikoille ja nopeuksille peräkkäisinä ajanhetkinä.

$$s_{n+1} \approx s_n + v_n\Delta t \tag{2}$$

$$v_{n+1} \approx v_n + a\Delta t \tag{3}$$

Näin saadaan perusnopeus, jonka avulla tavallinen liike saadaan kuvattua.

Tälläinen diskretisointi vääristää paikkaa. Kuten kuva (1) näyttää, jokaisella askeleella $\Delta t$ diskreetisti laskettu paikka jää hieman jälkeen analyyttisesti lasketusta paikasta. Mutta jos $\Delta t$ on tarpeeksi pieni, diskreetti malli on tarpeeksi tarkka.




**Kuva 1.** Paikka vakiokiihtyvyydellä, analyyttinen vs. diskreetti ratkaisu

Seuraavaksi määritellään, mitä tapahtuu kappaleiden kimmotessaan toisistaan. Elastisessa törmäyksessä liikemäärä (1) ja liike-energia pysyvät samoina. Tarkastellaan tilannetta ensin yksiulotteisesti. Liikemäärä $p$ ja liike-energia $E_{kin}$ ovat:

$$p = mv \tag{4}$$

$$E_{kin} = \frac{1}{2}mv^2 \tag{5}$$

Elastisen törmäyksen liikemäärä säilyy. Törmäyksen jälkeistä nopeutta merkitään tässä $v'$.

$$(4) \quad m_1v_1 + m_2v_2 = m_1v_1' + m_2v_2' \tag{6}$$

$$(5) \quad \frac{1}{2}mv_1^2 + \frac{1}{2}mv_2^2 = \frac{1}{2}mv_1'^2 + \frac{1}{2}mv_2'^2 \tag{7}$$

$$\Rightarrow mv_1^2 + mv_2^2 = mv_1'^2 + mv_2'^2 \tag{8}$$

Tästä epälineaarisesta yhtälöryhmästä voidaan ratkaista törmäyksen jälkeiset nopeudet $v_1'$ ja $v_2'$.

$$\begin{aligned}(6) \quad v_2' &= \frac{1}{m_2}(m_1v_1 + m_2v_2 - m_1v_1') \\ &= v_2 + \frac{m_1}{m_2}(v_1 - v_1')\end{aligned}$$

Ensin $v_2'$ ratkaistaa muiden muuttujien avulla, jonka jälkeen sijotetaan yhtälöön (8), jolloin saadaan toisen asteen yhtä nopeudelle $v_1'$.

$$\begin{aligned}(8) \quad m_1v_1 + m_2v_2^2 &= m_1v_1'^2 + m_2\left(v_2 + \frac{m_1}{m_2}(v_1 - v_1')\right)^2 \\ &= m_1v_1'^2 + m_2\left(v_2^2 + 2v_2\frac{m_1}{m_2}(v_1 - v_1') + \left(\frac{m_1}{m_2}(v_1 - v_1')\right)^2\right) \\ &= m_1v_1'^2 + m_2v_2^2 + 2v_2m_1(v_1 + v_1') + \frac{m_1^2}{m_2}\left(v_1^2 - 2v_1v_1' + v_1'^2\right) \\ &= m_1v_1'^2 + m_2v_2^2 + 2v_2m_1v_1 - 2v_2m_1v_1' + \frac{m_1^2}{m_2}v_1^2 - 2\frac{m_1^2}{m_2}v_1v_1' + \frac{m_1^2}{m_2}v_1'^2 \\ &= v_1'^2\left(m_1 + \frac{m_1^2}{m_2}\right) + v_1'\left(-2v_2m_1 - 2\frac{m_1^2}{m_2}v_1\right)^2 + m_2v_2 + 2v_2m_1v_1 + \frac{m_1^2}{m_2}v_1^2 \\ m_1v_1^2 &= v_1'^2\left(m_1 + \frac{m_1^2}{m_2}\right) + v_1'\left(-2v_2m_1 - 2\frac{m_1^2}{m_2}v_1\right)^2 + 2v_2m_1v_1 + \frac{m_1^2}{m_2}v_1^2\end{aligned}$$

$$\begin{aligned}0 &= v_1'^2\left(m_1 + \frac{m_1^2}{m_2}\right) - 2v_1'\left(v_2m_1 + \frac{m_1^2}{m_2}v_1\right) + 2v_2m_1v_1 + \frac{m_1^2}{m_2}v_1^2 - m_1v_1^2 \\ &= v_1'^2\left(1 + \frac{m_1}{m_2}\right) - 2v_1'\left(v_2 + \frac{m_1}{m_2}v_1\right) + 2v_2v_1 + v_1^2\left(\frac{m_1}{m_2} - 1\right)\end{aligned}$$

Tästä ratkaistaan $v_1'$.

$$v_1' = \frac{2\left(v_2 + \frac{m_1}{m_2}v_1\right) \pm \sqrt{\left(-2\left(v_2\frac{m_1}{m_2}v_1\right)\right)^2 - 4\left(1+\frac{m_1}{m_2}\right)\left(2v_2v_1 + v_1^2\left(\frac{m_1}{m_2}-1\right)\right)}}{2\left(1+\frac{m_1}{m_2}\right)}$$

Selkeyden vuoksi tarkastellaan neliöjuuren alla olevaa hetki erikseen:

$$\begin{aligned}
&4\left(v_2^2 + 2v_2\frac{m_1}{m_2}v_1\left(\frac{m_1}{m_2}\right)^2 v_1^2\right) - 4\left(2v_2v_1 + v_1^2\frac{m_1}{m_2} - v_1^2 + \frac{m_1}{m_2}2v_2v_1 + v_1^2\left(\frac{m_1}{m_2}\right)^2 + v_1^2\frac{m_1}{m_2}\right)\\
=&4\left(v_2^2 + 2v_2\frac{m_1}{m_2}v_1 + \left(\frac{m_1}{m_2}\right)^2 v_1^2 - 2v_2v_1 - v_1^2\frac{m_1}{m_2} + v_1^2 - \frac{m_1}{m_2}2v_2v_1 - v_1^2\left(\frac{m_1}{m_2}\right)^2 + v_1^2\frac{m_1}{m_2}\right)\\
=&4\left(v_2^2 - 2v_2v_1 + v_1^2\right)\\
=&4\left((v_2 - v_1)^2\right)
\end{aligned}$$

Josta seuraa

$$\begin{aligned}
\Rightarrow v_1' &= \frac{2\left(v_2\frac{m_1}{m_2}v_1\right) \pm \sqrt{4}(v_2 - v_1)}{2\left(1+\frac{m_1}{m_2}\right)}\\
&= \frac{m_2v_2 + m_1v_1 \pm m_2(v_2 - v_1)}{m_2 + m_1}\\
v_{1+}' &= \frac{1}{m_2+m_1}(2m_2v_2 + v_1(m_1 - m_2))\\
v_{1-}' &= \frac{1}{m_2+m_1}(v_1(m_1 + m_2)) = v_1
\end{aligned}$$

Tästä näkee, että $v_{1-}'$ ei käy, sillä nopeus ei muuttuisi, eli kappaleet menisivät toistensa läpi.

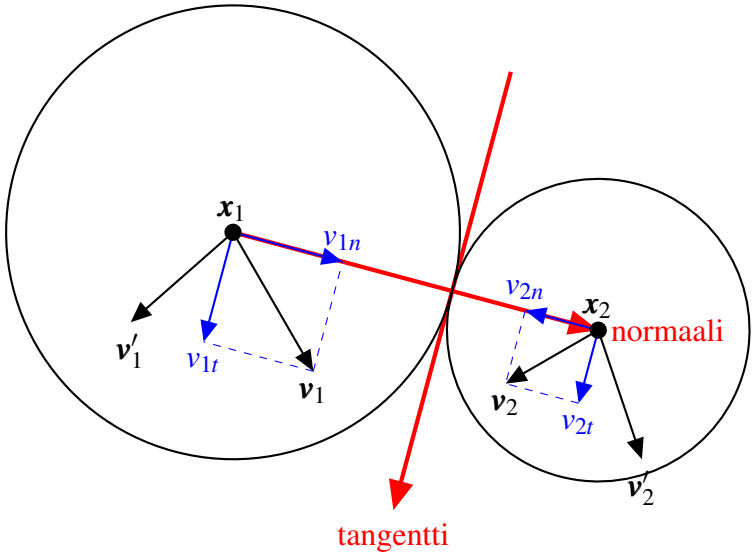
Kaksiulotteisessa törmäyksessä nopeus jaetaan normaalin ja tangentin suuntaisiin projektioihin, mutta laskutoimitus pysyy muuten samana. Kuva (2) näyttää törmäyksen.

Kun törmäystä käsitellään kaksiulotteisesti, se voidaan nähdä kahtena yksiulotteisena törmäyksenä, eli kappaleen nopeus jaetaan kahteen komponenttiin. Toinen ”törmäys” on kappaleiden keskipisteiden jännittämän normaalin suuntainen, toinen kohtisuorassa normaaliin. Nopeuksia merkitään kuvan mukaisesti: $v_n$ on normaalin suuntainen, $v_t$ tangentin suuntainen. Normaali määritellään yksikkövektorina kappaleiden keskikohtien perusteella:

$$\boldsymbol{n} = \frac{\boldsymbol{x}_1 - \boldsymbol{x}_2}{|\boldsymbol{x}_1 - \boldsymbol{x}_2|} \tag{9}$$

Tangentti on myös yksikkövektori, ja kohtisuorassa normaaliin:

$$\boldsymbol{t} = (-n_y, n_x) \tag{10}$$




**Kuva 2.** Kahden pallon törmäys, projektiot sinisellä

Nyt nopeudet voidaan jakaa komponentteihin pistetulon avulla seuraavanlaisesti:

$$v_{1n} = \boldsymbol{n} \cdot \boldsymbol{v}_1$$

$$v_{1t} = \boldsymbol{t} \cdot \boldsymbol{v}_1$$

$$v_{2n} = \boldsymbol{n} \cdot \boldsymbol{v}_2$$

$$v_{2t} = \boldsymbol{t} \cdot \boldsymbol{v}_2$$

Tangentin suuntaisesti nopeus ei muutu, normaalin suuntaisesti muutos tapahtuu kuten johdettiin edellä.

$$\begin{aligned} v'_{1t} &= v_{1t} \\ v'_{2t} &= v_{2t} \\ v'_{1n} &= \frac{1}{m_1 + m_2}(2m_2 v_{2n} + v_{1n}(m_1 - m_2)) \\ v'_{2n} &= \frac{1}{m_1 + m_2}(2m_1 v_{1n} + v_{2n}(m_2 - m_1)) \end{aligned}$$

Pitää ottaa huomioon että nämä ovat skalaariarvoja. Ne saadaan oikeaoppiseen vektorimuotoon kertomalla normaalin suuntaiset komponentit normaalivektorilla ja tangentin suuntaiset komponentit tangenttivektorilla. Uusi nopeus on komponenttivektoreiden summa

$$\boldsymbol{v}'_1 = \boldsymbol{v}'_{1n} + \boldsymbol{v}'_{1t}$$

$$\boldsymbol{v}'_2 = \boldsymbol{v}'_{2n} + \boldsymbol{v}'_{2t}$$

Pistetulon (Berchek 2009) kanssa kokonaan auki kirjoitettuna:

$$\boldsymbol{v}_1' = \boldsymbol{v}_1 - \frac{2m_2}{m_1+m_2}\frac{(\boldsymbol{v}_1-\boldsymbol{v}_2)\cdot(\boldsymbol{x}_1-\boldsymbol{x}_2)}{|\boldsymbol{x}_1-\boldsymbol{x}_2|^2}(\boldsymbol{x}_1-\boldsymbol{x}_2) \tag{11}$$

$$\boldsymbol{v}_2' = \boldsymbol{v}_2 - \frac{2m_1}{m_1+m_2}\frac{(\boldsymbol{v}_2-\boldsymbol{v}_1)\cdot(\boldsymbol{x}_2-\boldsymbol{x}_1)}{|\boldsymbol{x}_2-\boldsymbol{x}_1|^2}(\boldsymbol{x}_2-\boldsymbol{x}_1) \tag{12}$$

## 2.3 Virtaus

Nesteen virtauksen laskeminen yleisessä tapauksessa on analyyttisesti lähes mahdotonta ratkaista, mutta sen voi tehdä numeerisesti. Tärkeitä yhtälöitä virtausmekaniikassa ovat massan säilymistä kuvaava yhtälö ja liikemääräyhtälö. Heikosti kokoonpuristuvan virtauksen jatkuvuusyhtälö on

$$\frac{d\rho}{dt} = -\rho\nabla\cdot\boldsymbol{v},$$

missä $\rho$ on tiheys ja $\boldsymbol{v}$ nopeus. Navier–Stokes-yhtälö eli liikemäärän säilymistä kuvaava yhtälö kokoonpuristumattomalle virtaukselle on

$$\frac{d\boldsymbol{v}}{dt} = -\frac{1}{\rho}\nabla p + \nu\nabla^2\boldsymbol{v} + \boldsymbol{g}$$

missä $g$ on putoamiskiihtyvyys, $p$ on paine ja $\nu$ kinemaattinen viskositeetti (nesteen aineominaisuus). Koska näissä kahdessa yhtälössä on kolme tuntematonta muuttujaa, $\rho$, $p$ ja $\boldsymbol{v}$, tarvitaan vielä kolmas yhtälö yhtälöryhmän ratkaisemiseksi, eli jokin tilanyhtälö:

$$\rho = \rho(p)$$

Näitä on useita, esimerkiksi Taitin tilanyhtälö (Korhonen 2016).

Näiden yhtälöiedn ratkeiseminen numeerisesti on mahdollista, mutta hyvin vaikeaa. Veden (sekä muiden nesteiden) virtausta voidaan kuitenkin simuloida yksinkertaisemmin kohtelemalla nestettä joukkona pieniä palloja, jotka liikkuvat pintaa kuvaavan kalvon alla. Tekninen termi tälle on Smoothed Particle Hydrodynamics eli SPH. Tässä menetelmässä jokainen hiukkanen kuvaa vakiomassaista nestetilavuutta, sillä tällöin massan säilyminen on taattu, eli jatkuvuusyhtälö on voimassa. Diskretoidun liikeyhtälön johtaminen on silti huomattavan monimutkaista (Korhonen 2016). Tämän lisäksi toimiva SPH vaatii hyvin suuren määrän partikkeleita, pinnan kalvon laskeminen ja stabiilin simulaation luominen voi olla hyvin vaikeaa (Catto 2012).

Työn lopussa esitellään nestesimulaatio, jossa vettä simuloidaan suurena joukkona palloja. Pallot liikkuvat yksittäisen pallon mekaniikan avulla, joka on kuvattu luvussa 2.2. Työssä ei yritetä ratkaista virtausongelmaa oikeaoppisesti, koska tarkoitus on tutkia peleissä toteutettavia metodeja, eikä realistisia simulaatioita.

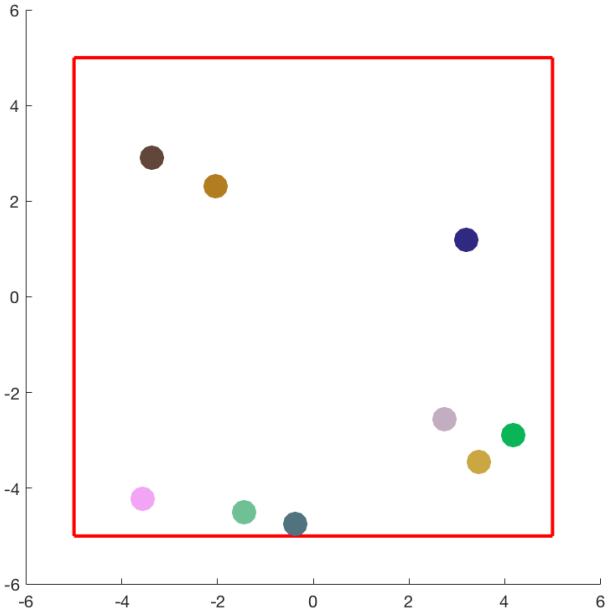
# 3 SIMULAATIOESIMERKIT

## 3.1 Oma simulaatio: Palloja laatikossa

Esmierkkisimullaationa työssä on toteutettu yksinkertainen mekaaninen fysiikkasimulaatio, jossa $n$ määrä palloja lähtee liikkeelle satunnaisnopeuksilla, ja kimpolee sekä laatikon seinistä että toisistaan. Simulaatio on toteutettu MATLAB-ohjelmointikielellä.

Simulaatiossa pallot liikkuvat kaksiulotteisessa kitkattomassa tilassa, mutta ne menettävät liikemäärää kimmotessaan. Pallot ”liukuvat”, pyörimisliikettä ei ole otettu huomioon. Törmäyksen jälkeistä liikemäärää pienennetään kertoimella, joka on ohjelmoidessa valittu, esimerkiksi $\omega = 0.99$.

$$\omega(m_1 v_1 + m_2 v_2) = m_1 v_1' + m_2 v_2' \tag{6}$$




**Kuva 3.** Palloja laatikossa

Pallojen liikettä kuvataan aiemmin eritellyllä tavalla. Tässä malli tosin yksinkertaistuu kiihtyvyyden takia: Koska laatikkomallissa palloihin ei kohdistu sisäisiä voimia, $a = 0$, joten (3) muuttuu muotoon $v_{n+1} = v_n$, eli sitä ei tarvitse huomioida, koska muutosta ei tapahdu. Monimutkaisempi malli jossa painovoimaa käsitellään on esitetty kohdassa 3.2.

Tässä tapauksessa oletetaan, että kaikki pallot ovat samanpainoisia, joten elastisen törmäysyhtälön massatermi muuttuu ykköseksi, eli siitä ei tarvitse välittää.

### 3.1.1 Koodi

Tarkka MATLAB-koodi on liitteenä. Ohjelma toimii pääpiirteissään seuraavanlaisesti:

Ensiksi alustetaan kuinka monta palloa halutaan, eli kuinka suuri $n$ on. Sitten palloille annetaan satunnaiset lähtönopeudet ja -paikat, ja jos $n < 9$, niille määritellään biljardpallojen mukaiset värit. Jos palloja on enemmän, värit arvotaan satunnaisesti. Satunnaisessa paikkasijoituksessa syntyy ongelmia, jos palloja sijoitetaan vahingossa sisäkkäin. Siksi asia tarkistetaan, ja jos kaksi palloa ovat sisäkkäin, koodi nostaa näille palloille liput. Liputettu pallo ei voi kimmota muista palloista. Tämän on tarkoitus estää sisäkkäin jumiin jäämistä, tilannetta jossa pallot jäävät kimpoilemaan toisiinsa eivätkä pääse tarpeeksi kauas toisistaan ennen kuin ne kimpoavat taas toisiaan päin. Myös vakiot, kuten aika-askeleen pituus ja kimmokerroin, määritellään.

Kun alustus on tehty, alkaa ajastettu while-luuppi. Seuraavan aika-askeleen sijoituspaikka lasketaan kaavan (2) mukaan. Sitten tarkistetaan seiniin osuminen: jos pallo osuu seinän rajalla, se kimpoaa peilikulmassa, eli seinään kohtisuorassa oleva nopeusvektorin komponentti muuttuu erimerkkiseksi. Tämän lisäksi nostetaan lippu jonka mukaan pallo on seinässä, tarkoitus on jälleen varmistaa, ettei se jää kimpolemaan seinän sisään. Jos pallo on tarpeeksi kaukana seinästä, lippu vapautetaan.

Seuraavaksi testataan pallojen toisistaan kimpoilemista. Tämä toimii samalla periaatteella: jos kahden pallon välinen etäisyys on pienempi tai yhtä suuri kuin pallon halkaisija, pallot kimpoavat. Nopeus lasketaan kaavojen (11) ja (12) mukaan ja kerrotaan kimmokertoimella. Pallot liputetaan, ja jos kaksi liputettua palloa eivät ole sisäkkäin, lipput vapautetaan. Lopuksi pallot piirretään.

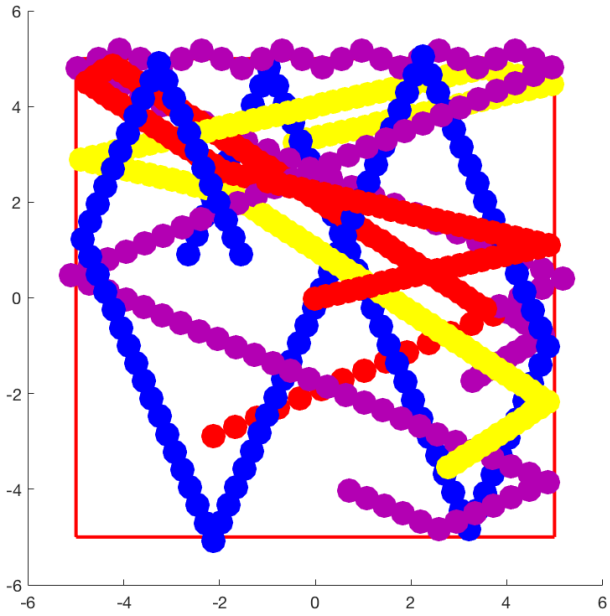
### 3.1.2 Ongelmat

Näinkin yksinkertaisessa simullatiossa ei voi välttyä ongelmilta. Kuten aiemmin kuvailtiin, määriteltiin, että jos pallon keskipisteen ja seinän välinen etäisyys on pienempi kuin pallon säde, pallo kimpoaa peilikulmassa. Ongelma syntyy, jos pallo ei seuraavan aika-askeleen aikana pääse pois seinän sisästä, jolloin pallo yrittää jälleen kimmota peilikulmassa, päätyen vain takaisin seinään. Tästä seuraa se, että pallo jää kimpoilemaan seinän sisään. Samanlai-

nen ongelma kävi myös pallojen keskinäisen liikkeen kanssa: toisinaan pallot jäivät jumiin sisäkkäin.

Ongelmaa voidaan yrittää ratkaista liputtamalla palloja kimpoamisen jälkeen. Tarkistetaan, ovatko juuri kimmonneet pallot vielä sisäkkäin, tai määrittelemällä että pallo ei voi kimmota samaan kappaleeseen seuravalla peliloopilla. Nämä tarkistukset tosin hidastavat ohjelman ajamista, varsinkin suurella määrällä palloja, ja varma korjaus vaatisi niin lyhyitä peliloppeja että ohjelman ajamisen miellyttävyys kärsii.

Yksi odottamaton pullonkaula oli piirtäminen. Ohjelma oli määritelty ajettavan tietyn ajan. Kokeilumielessä mitattiin montako pelilooppia ohjelma ehtii tehdä tavallisesti[1], ja silloin kun piirtämisfuntio otettiin väliaikaisesti pois. Ilman piirtämistä kierroksien määrä noin kolminkertaisutui, 30 s ajolla kahdellakymmenellä pallolla 546:sta loopista 1550:een. Hidastava piirtäminen on ohjelmiston ongelma, ja sille ei ohjelmoija voi mitään. Tässä simulaatiossa muun laskennan optimointi ei ole mielekästä, koska graffikan tuottaminen vie valtaosan ajasta.




**Kuva 4.** Neljän pallon liikeradat: Violetti pallo jäi yläseinään

[1]Looppilaskin otettiin pois koodin liitteenä olevasta versiosta

## 3.2 Oma simulaatio: Virtaus

Kuten aiemmin mainittiin, virtausta voi simuloida joukolla pieniä palloja. Käytännön kokeilua varten sovellettiin edellisen simulaation koodia. Simulaatiossa edellinen laatikko katsottiin maljaksi, jossa pallot vellovat. Alkusysäyksen aikaansaamiseksi kaikki pallot laitettiin maljan sivuun, ja päästettiin putoamaan. Pisarapallojen lisäksi pirrettiin punaisella massakeskipiste, jotta paremmin voitaisiin seurata yleistä kayttäytymistä.

### 3.2.1 Koodi

Virtaussimulaation koodi perustuu vahvasti Palloja laatikossa -koodiin, sillä tarkoituksena on toteuttaa simuaatio partikkelidynamiikan avulla. Erona on pallojen määrän lisäksi sofistikoituneempi käyttäytyminen: siinä missä laatikon pallot kimpoilivat kuin biljardipallot, maljan pallot putoavat painovoiman takia.

### 3.2.2 Ongelmat

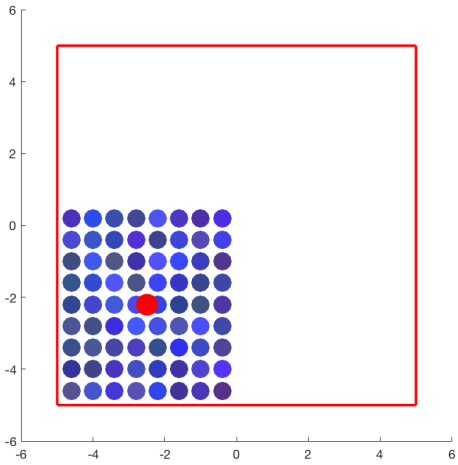
Tässä simulaatiossa syntyi huomattavasti enemmän ongelmia. Painovoiman painamina pallot jäivät herkästi päällekkäin kimpoamatta. Myös massakeskipiste näyttää, että alkuallon jälkeen vesi alkaa käyttäytyä hyvin tasaisesti.

Tässä vastaan tuli myös silkka laskentateho. Paremman simulaation saisi suurella määrällä pienenpieniä palloja, mutta jo mallissa käytetyt vaivaiset 72 palloa saa animaation näyttämään nykivältä.

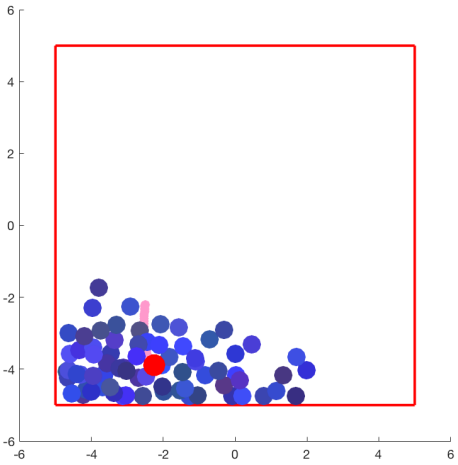
Jo näinkin yksinkertainen simulaatio näyttää miksi videopeleissä pelaaja harvemmin pääsee realistisesti käyttäytyvään veteen, ja miksi vesi on yleensä hyvin tyyliteltyä: realistinen vesisimulaatio vie liikaa laskentatehoa.

# 4 KESKUSTELU

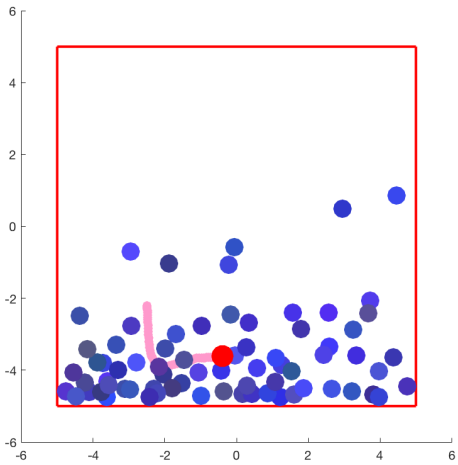
Työtä aloitettaessa suunnitelmana oli vertailla kaupallisia moottoreita, mutta selvisi, että niihin on mahdotonta päästä käsiksi. On ymmärrettävää, että kehittäjät haluavat varjella tuotteitaan. Yleistä kirjallisuutta oli hankala löytää, mistä voi päätellä alan olevan vielä suhteellisen nuori.



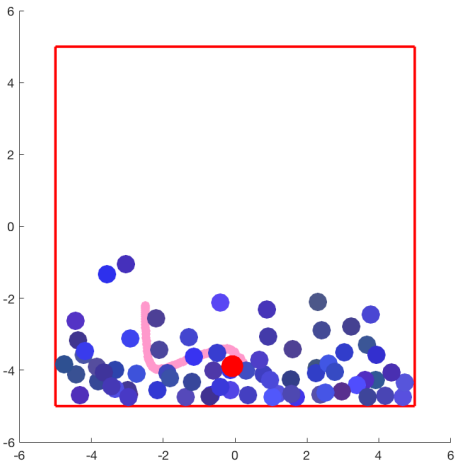

(a) Alkuasetelma $t = 0$




(b) Pallot ovat lähteneet liikkeelle, $t = 5$s




(c) Pallot aaltona oikealle laidalle, $t = 10$s




(d) Pallot rauhoittuvat maljan pohjalle, pientä lainehtimista lukuun ottamatta, $t = 30$s

**Kuva 5.** Lainehtiva laatikko, jossa punaisella merkitty massakeskipiste, vaaleanpunaisella massakeskipisteen rata

Omia simulaattoreita tehdessä selvisi monia asioita. Niinkin yksinkertaista simulaatiota kuin pallojen laatikossa törmäily tehdessä täytyy ottaa huomioon paljon asioita, ja virheitä tulee helposti. Liikkeen määrittäminen oli helppoa, jopa törmäyksen jälkeen nopeutta oli helppo muokata oikeaoppisesti. Mutta usean kappaleen keskinäinen reagointi tuotti ongelmia. Järkevän virheenkäsittelytoiminnan tekeminen on haastavaa. Virtaussimulaatiossa ongelmat korostuivat. Oli yllättävää huomata kuinka huono simulaatio oli, vaikka se periaatteessa noudatti parhaaksi nähtyjä prinsiippejä. Olisi mielenkiintoista nähdä miten algoritmi toimisi suuremmalla määrällä pienempiä palloja, ja tehokkaammassa laitteistossa.

Perimmäinen ongelma numeerisissa simulaatioissa lienee se, että jatkuvaa maailmaa kuvataan epäjatkuvalla tavalla. Numeerisessa mallissa voi olla että törmäys rekisteröidään vasta kun kappaleet ovat jo sisäkkäin, tai nopeasti liikkuvat kappaleet hyppäävät toistensa yli koskematta.

# 5 YHTEENVETO

Työn tarkoituksena oli kartoittaa tunnettua tietoa fysiikkamoottoreista, keskittyen siihen, miten niitä kehitetään ja sovelletaan videopeleissä. Aihetta lähestyttiin mekaniikan ja virtauksen simuloimisen kautta. Työ aloitettiin kartoittamalla simuloitavien järjestemien perusteet. Osoittautui, että kaupallisista moottoreista on melko mahdotonta saada yksityiskohtaista tietoa, sillä yhtiöt suojelevat omaisuuttaan tarkasti.

Omia simulaatioita tehtiin kaksi: ensimmäinen kuvasi satunnaisesti laatikossa kimpolevaa joukkoa palloja, toinen sovelsi ensimmäistä virtauksen simuloimiseksi kuvaamalla nesteen liikettä partikkelijoukkona. Näissä simulaatioissa ongelmiksi osoittautui törmäyksen jälkeinen liike, ja jälkimmäisessä tapauksessa matala laskentateho esti paremman simulaation luomisen. Vaikka törmäyksenjälkeisen liikkeen suunnan ja suuruuden laskenta oli helppoa, pallot tahtoivat jäädä kiinni toisiinsa virheellisesti. Hyvä virtaussimulaatio vaatisi niin suuren määrän partikkeleja, että ohjelman ajaminen muuttuisi epämiellyttävän hitaaksi.

# Lähteet


Berchek, Chad (2009). *2-Dimensional Elastic Collisions without Trigonometry*. Published online, URL: www.vobarian.com/collisions/2dcollisions2.pdf.

Catto, Erin (2012). "Physics for Game Programmers". Teoksessa: *Game Developers Conference 2012*.

Godot (2018). Published online, URL: www.godotengine.org.

Gregory, Jason (2009). *Game Engine Architecture*. CRC Press.

Havok (2017). Published online, URL: www.havok.com.

Korhonen, Joonas (2016). "Smoothed Particle Hydrodynamics virtausmekaniikan simulaatiossa". Tutkielma. Jyväskylän yliopisto.

Laine, Ville (2007). "Dynamiikan simulointi 3D-animaatiossa". Tutkielma. Lahden ammattikorkeakoulu.

Nilson, Björn ja Söderberg, Martin (2007). "Game Engine Architecture". Tutkielma. Mälardalen University.

Unity (2018). Published online, URL: www.unity3d.com.

UnrealEngine (2018). Published online, URL: www.unrealengine.com/en-US/blog.

# Kuvat

# Liite 1: Palloja laatikossa MATLAB-koodi

```
% Code by Arna Hyvarinen

clc
clear all
close all
cla reset
hold on

% laatikon reunat
r1 = plot([-5 -5],[-5 5],'r','LineWidth',2);
r2 = plot([5 5],[-5 5],'r','LineWidth',2);
r3 = plot([-5 5],[-5 -5],'r','LineWidth',2);
r4 = plot([-5 5],[5 5],'r','LineWidth',2);

axis equal
axis([-6 6 -6 6]);

n = 20; % montako palloa

% satunnaiset lahtonopeudet ja -paikat
v0 = 25*rand(n,2);
p0 = 7*rand(n,2) - 3.5;

% maaritellaan vakioarvoja
dt = 0.01;
kimmo = 0.99; % kimmoisuus
radius = 0.25; % palloje koko
halkaisija = 2*radius;
time = 30.00001; % kuinka pitkaan ajetaan

% tarkistusliput: onko pallo seinan sisalla, tai toisen pallon sisalla
palloSeinassa = zeros(n,1);
palloPallossa = zeros(n,1);

piste = zeros(n,1); % piirtamis-muuttuja

```

```
37 p = zeros(n,2); % sijainti
38 v = zeros(n,2); % nopeus
39 % ensimmainen sijainti ja nopeus alustetaan
40 for (i = 1:n)
41     p(i,:) = p0(i,:);
42 end
43
44 for (i = 1:n)
45     v(i,:) = v0(i,:);
46 end
47
48 % varit: jos palloja on saman verran tai vahemman kuin biljardissa,
      biljardipallojen varit
49 % yellow, blue, red, magenta, orange, green, maroon, black
50 if n < 9
51     vari = [1, 1, 0
52             0, 0, 1
53             1, 0, 0
54             0.7, 0, 0.7
55             0.65, 0, 0
56             0, 1, 0
57             1, 0.5, 0
58             0, 0, 0
59             ];
60 else
61     vari = rand(n,3); % satunnaiset rgb-varit
62 end
63
64 % testataa, spawnaavatko pallot paallekkain. Jos kylla, nosta lippu
65 for (i = 1:n)
66     for (j = (i+1):n)
67         dp = abs(p0(i,:)-p0(j,:));
68         osuu = (sqrt(dp(1)^2 + dp(2)^2) <= halkaisija);
69         if osuu
70             palloPallossa(i) = 1;
71             palloPallossa(j) = 1;
72         end
73     end
```

```
74 end
75
76 % toiminta alkaa
77 tic
78 while(toc < time)
79
80     % lasketaan pallojen seuraava sijainti
81     for (i = 1:n)
82         p(i,:) = p(i,:) + v(i,:)*dt;
83         piste(i) = plot(p(i,1),p(i,2),'.','Color',vari(i,:),'MarkerSize'
               ,50);
84     end
85
86     % reunoista kimpoaminen
87     % jos pallo on seinassa, ei kimpoamista
88     for (i = 1:n)
89         if ((5-abs(p(i,1)) <= radius)) && palloSeinassa(i) == 0
90             v(i,1) = -v(i,1);
91             v(i,1) = kimmo*v(i,1);
92             v(i,2) = kimmo*v(i,2);
93             palloSeinassa(i) = 1;
94         elseif ((5-abs(p(i,2)) <= radius)) && palloSeinassa(i) == 0
95             v(i,2) = -v(i,2);
96             v(i,1) = kimmo*v(i,1);
97             v(i,2) = kimmo*v(i,2);
98             palloSeinassa(i) = 1;
99         elseif ((5-abs(p(i,1)) > radius)) && palloSeinassa(i) == 1
100            palloSeinassa(i) = 0;
101        elseif ((5-abs(p(i,2)) > radius)) && palloSeinassa(i) == 1
102            palloSeinassa(i) = 0;
103        end
104    end
105
106
107    % pallojen toisistaan kimpoaminen
108    % jos pallot ovat sisakkain, ei kimpoamista
109    for (i = 1:n)
110        for (j = (i+1):n)
```

```
111             dp = abs(p(i,:)-p(j,:));
112             osuu = (dp(1)^2 + dp(2)^2 <= halkaisija^2);
113             if osuu && (palloPallossa(i) == 0) && (palloPallossa(j) == 0)
114                 v1 = v(i,:);
115                 v2 = v(j,:);
116                 x1 = p(i,:);
117                 x2 = p(j,:);
118                 v(i,:) = kimmo* v1 - dot(v1-v2,x1-x2)/(norm(x1-x2))^2 *(
                        x1-x2);
119                 v(j,:) = kimmo* v2 - dot(v2-v1,x2-x1)/(norm(x2-x1))^2 *(
                        x2-x1);
120                 palloPallossa(i) = 1;
121                 palloPallossa(j) = 1;
122             elseif not(osuu) && (palloPallossa(i) == 1) && (palloPallossa
                    (j) == 1)
123                 palloPallossa(i) = 0;
124                 palloPallossa(j) = 0;
125             end
126         end
127     end
128
129     % piirtaminen
130     drawnow;
131     for (i = 1:n)
132         delete(piste(i));
133     end
134
135 end
136
137 % viimeisten pisteiden piirtaminen
138 for (i = 1:n)
139      plot(p(i,1),p(i,2),'.','Color',vari(i,:),'MarkerSize',50);
140 end
141
142 %eof
```

# Liite 2: Malja MATLAB-koodi

```
% Code by Arna Hyvarinen

clc
clear all
close all
cla reset
hold on

% laatikon reunat
r1 = plot([-5 -5],[-5 5],'r','LineWidth',2);
r2 = plot([5 5],[-5 5],'r','LineWidth',2);
r3 = plot([-5 5],[-5 -5],'r','LineWidth',2);
r4 = plot([-5 5],[5 5],'r','LineWidth',2);

axis equal
axis([-6 6 -6 6]);

n = 72; % montako palloa

% maaritellaan vakioarvoja
dt = 0.01;
kimmo = 0.99; % kimmoisuus
radius = 0.25; % palloje koko
halkaisija = 2*radius;
time = 30.00001; % kuinka pitkaan ajetaan
g = 100; % painovoima

mkp = 0; % massakeskipiste

% pieni satunnainen lahtonopeus
k = 5;
v0 = -k/2 + k*rand(n,2);

% alkupaikat, aseteltuna
x = linspace(-4.6,-0.4,8);
px = ones(n/3,1);
```

```
pya = [3.4:0.6:5];
py = [];
for i = 1:length(x)
    px(i*3-2:i*3) = x(i);
    py = [py pya];
end
py = -py';
p0 = [px py
      px py+1.8
      px py+1.8*2];

% tarkistusliput: onko pallo seinan sisalla, tai toisen pallon sisalla
palloSeinassa = zeros(n,1);
palloPallossa = eye(n,n);

piste = zeros(n,1); % piirtamis-muuttuja

p = zeros(n,2); % sijainti
v = zeros(n,2); % nopeus
% ensimmainen sijainti ja nopeus alustetaan
for (i = 1:n)
    p(i,:) = p0(i,:);
end

for (i = 1:n)
    v(i,:) = v0(i,:);
end

vari = 0.5 + (1-0.5)*rand(n,3); % satunnaiset siniset rgb-varit
vari(:,1) = vari(:,1)*0.35;
vari(:,2) = vari(:,2)*0.35;

% toiminta alkaa
tic
while(toc < time)

    % lasketaan pallojen seuraava sijainti ja nopeus
    for (i = 1:n)
```

```
75          v(i,:) = v(i,:) - [0,g]*dt; % puotoamiskiihtyvyys on alas ja
                vakio
76          p(i,:) = p(i,:) + v(i,:)*dt;
77          piste(i) = plot(p(i,1),p(i,2),'.','Color',vari(i,:),'MarkerSize'
                ,50);
78      end
79
80      % piirretaan massakeskipiste
81      plot( (sum(p(:,1))/n), (sum(p(:,2))/n),'.','Color',[1 0.6 0.8],'
            MarkerSize',25 );
82      mkp = plot( (sum(p(:,1))/n), (sum(p(:,2))/n),'r.','MarkerSize',60 );
83
84      % reunoista kimpoaminen
85      % jos pallo on seinassa, ei kimpoamista
86      for (i = 1:n)
87          if ((5-abs(p(i,1)) <= radius)) && palloSeinassa(i) == 0
88              v(i,1) = -v(i,1);
89              v(i,1) = kimmo*v(i,1);
90              v(i,2) = kimmo*v(i,2);
91              palloSeinassa(i) = 1;
92          elseif ((5-abs(p(i,2)) <= radius)) && palloSeinassa(i) == 0
93              v(i,2) = -v(i,2);
94              v(i,1) = kimmo*v(i,1);
95              v(i,2) = kimmo*v(i,2);
96              palloSeinassa(i) = 1;
97          elseif ((5-abs(p(i,1)) > radius)) && palloSeinassa(i) == 1
98              palloSeinassa(i) = 0;
99          elseif ((5-abs(p(i,2)) > radius)) && palloSeinassa(i) == 1
100             palloSeinassa(i) = 0;
101         end
102
103         % jos pallo paatyy maljan ulkopuolelle, se sijoitetaan takaisin
                sen
104         % sisalle
105         if p(i,1) < -5 + radius
106             p(i,1) = -5 + radius;
107         end
108         if p(i,1) > 5 - radius
```

```
109             p(i,1) = 5 - radius;
110         end
111         if p(i,2) < -5 + radius
112             p(i,2) = -5 + radius;
113         end
114         if p(i,2) > 5 - radius
115             p(i,2) = 5 - radius;
116         end
117     end
118
119
120     % pallojen toisistaan kimpoaminen
121     % jos pallot ovat sisakkain, ei kimpoamista
122     for (i = 1:n)
123         for (j = (i+1):n)
124             dp = abs(p(i,:)-p(j,:));
125             osuu = (dp(1)^2 + dp(2)^2 <= halkaisija^2);
126             if osuu && (palloPallossa(i,j) == 0)
127                 v1 = v(i,:);
128                 v2 = v(j,:);
129                 x1 = p(i,:);
130                 x2 = p(j,:);
131                 v(i,:) = kimmo* v1 - dot(v1-v2,x1-x2)/(norm(x1-x2))^2 *(
                        x1-x2);
132                 v(j,:) = kimmo* v2 - dot(v2-v1,x2-x1)/(norm(x2-x1))^2 *(
                        x2-x1);
133                 palloPallossa(i,j) = 1;
134                 palloPallossa(j,i) = 1;
135             elseif not(osuu) && (palloPallossa(i,j) == 1)
136                 palloPallossa(i,j) = 0;
137                 palloPallossa(j,i) = 0;
138             end
139         end
140     end
141
142     % piirtaminen
143     drawnow;
144     for (i = 1:n)
```

```
145         delete(piste(i));
146     end
147     delete(mkp)
148 end
149 
150 % viimeisten pisteiden ja masakeskipisteen piirtaminen
151 for (i = 1:n)
152      plot(p(i,1),p(i,2),'.','Color',vari(i,:),'MarkerSize',50);
153 end
154 plot( (sum(p(:,1))/n), (sum(p(:,2))/n),'r.','MarkerSize',60 );
155 
156 %eof
```