Particle Simulator

Arany Mátyás Nagy Házi

Generated by Doxygen 1.10.0

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

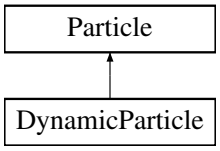
Class Documentation

4.1 DynamicParticle Class Reference

Dynamic Particle that move by the laws of physics.

`#include <particles.h>`

Inheritance diagram for DynamicParticle:



Public Member Functions

- **DynamicParticle** (float x, float y, unsigned int mass, int charge, float velx=0, float vely=0, ParticleType type=None)

  Creates object with the given attributes /summary>
- void calcGravity (Particle const &other)

  Calculates force of gravity, affecting the particle, caused by another particle.
- Collision calcCoulomb (Particle const &other)

  Calculates the Coulomb force affecting the particle, caused by another particle.
- bool refreshPos (float const rate)

  Refreshes the position of particle, determined by the forces and its velocity.
- bool refreshState ()

  Refreshes the state of the particle.
- MoveTypes getType () const

  Returns Dynamic type.
- Vector getVel () const

  Return the velocity of particle.

Public Member Functions inherited from Particle

- **Particle** (float x, float y, unsigned int mass, int charge, ParticleType type=None)

  Constructor with default particle type of 'None'.
- **Particle** (Vector pos, unsigned int mass, int charge, ParticleType type=None)

  Constructor with default particle type of 'None'.
- Vector **getPos** () const

  Returns the position vector of the particle.
- unsigned int **getMass** () const

  Returns the mass of the particle.
- int **getCharge** () const

  Returns the charge of the particle.
- bool **isOutOfView** () const

  Returns whether the particle is OutOfView = far out of screen.
- bool **isAccelerating** () const

  Returns whether the particle is Accelerating.
- bool **isBalanced** () const

  Returns whether the particle is Balanced.
- bool **isCalm** () const

  Returns whether the particle is Calm.
- State **getState** () const

  Returns the state of the particle.
- std::string **ParticleString** () const

  Provides the string description of a particle.
- ParticleType **getPType** () const

  Returns the particle type of the particle.
- sf::CircleShape **getShape** () const

  Returns with the shape of the particle.

Additional Inherited Members

Protected Attributes inherited from Particle

- Vector **pos**

  Position of the particle.
- unsigned int **mass**

  Mass of the particle.
- int **charge**

  Charge of the particle.
- State **state**

  State of the particle.
- ParticleType **type**

  Predefined type of particle.
- sf::CircleShape **circ**

  SFML Circleshape representation of the particle.

4.1.1 Detailed Description

Dynamic Particle that move by the laws of physics.

4.1.2 Member Function Documentation

4.1.2.1 calcCoulomb()

```
Collision DynamicParticle::calcCoulomb (
            Particle const & other )  [virtual]
```

Calculates the Coulomb force affecting the particle, caused by another particle.

Returns

The type of collision if the two Particle collided

Implements Particle.

4.1.2.2 calcGravity()

```
void DynamicParticle::calcGravity (
            Particle const & other )  [virtual]
```

Calculates force of gravity, affecting the particle, caused by another particle.

Implements Particle.

4.1.2.3 getType()

```
MoveTypes DynamicParticle::getType ( ) const  [inline], [virtual]
```

Returns Dynamic type.

Implements Particle.

4.1.2.4 getVel()

```
Vector DynamicParticle::getVel ( ) const  [inline], [virtual]
```

Return the velocity of particle.

Returns

Implements Particle.

4.1.2.5 refreshPos()

```
bool DynamicParticle::refreshPos (
            float const rate )  [virtual]
```

Refreshes the position of particle, determined by the forces and its velocity.

Parameters

rate	The speed of the particle (delta T)

Returns

Whether there was a change in state

Implements Particle.

4.1.2.6 refreshState()

```
bool DynamicParticle::refreshState ( )  [virtual]
```

Refreshes the state of the particle.

Returns

If there was a change in state

Implements Particle.

The documentation for this class was generated from the following files:

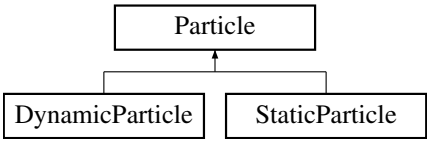
- X:/Saját/prog/BME/prog2/Nagyhazi/proba2/AranyMatyasNH/particles.h
- X:/Saját/prog/BME/prog2/Nagyhazi/proba2/AranyMatyasNH/particles.cpp

4.2 Particle Class Reference

Abstract class for any kind of particle.

```
#include <particles.h>
```

Inheritance diagram for Particle:



Public Member Functions

- **Particle** (float x, float y, unsigned int mass, int charge, ParticleType type=None)

  Constructor with default particle type of 'None'.
- **Particle** (Vector pos, unsigned int mass, int charge, ParticleType type=None)

  Constructor with default particle type of 'None'.
- Vector **getPos** () const

  Returns the position vector of the particle.
- unsigned int **getMass** () const

  Returns the mass of the particle.
- int **getCharge** () const

  Returns the charge of the particle.
- bool **isOutOfView** () const

  Returns whether the particle is OutOfView = far out of screen.
- bool **isAccelerating** () const

  Returns whether the particle is Accelerating.
- bool **isBalanced** () const

  Returns whether the particle is Balanced.
- bool **isCalm** () const

  Returns whether the particle is Calm.
- State **getState** () const

  Returns the state of the particle.
- std::string **ParticleString** () const

  Provides the string description of a particle.
- virtual MoveTypes getType () const =0

  Returns Dynamic / Static.
- virtual Vector getVel () const =0

  Returns the velocity vector of the particle.
- ParticleType **getPType** () const

  Returns the particle type of the particle.
- sf::CircleShape **getShape** () const

  Returns with the shape of the particle.
- virtual void calcGravity (Particle const &other)=0

  Calculates force of gravity, affecting the particle, caused by another particle.
- virtual Collision calcCoulomb (Particle const &other)=0

  Calculates the Coulomb force affecting the particle, caused by another particle.
- virtual bool refreshPos (float const rate)=0

  Refreshes the position of particle, determined by the forces and its velocity.
- virtual bool refreshState ()=0

  Refreshes the state of a particle, determined by its velocity, and acceleration.

Protected Attributes

- Vector **pos**

  Position of the particle.
- unsigned int **mass**

  Mass of the particle.
- int **charge**

  Charge of the particle.
- State **state**

  State of the particle.
- ParticleType **type**

  Predefined type of particle.
- sf::CircleShape **circ**

  SFML Circleshape representation of the particle.

4.2.1 Detailed Description

Abstract class for any kind of particle.

4.2.2 Member Function Documentation

4.2.2.1 calcCoulomb()

```
virtual Collision Particle::calcCoulomb (
            Particle const & other )  [pure virtual]
```

Calculates the Coulomb force affecting the particle, caused by another particle.

Implemented in StaticParticle, and DynamicParticle.

4.2.2.2 calcGravity()

```
virtual void Particle::calcGravity (
            Particle const & other )  [pure virtual]
```

Calculates force of gravity, affecting the particle, caused by another particle.

Implemented in StaticParticle, and DynamicParticle.

4.2.2.3 getType()

```
virtual MoveTypes Particle::getType ( ) const  [pure virtual]
```

Returns Dynamic / Static.

Implemented in StaticParticle, and DynamicParticle.

4.2.2.4 getVel()

```
virtual Vector Particle::getVel ( ) const  [pure virtual]
```

Returns the velocity vector of the particle.

Implemented in StaticParticle, and DynamicParticle.

4.2.2.5 refreshPos()

```
virtual bool Particle::refreshPos (
            float const rate )  [pure virtual]
```

Refreshes the position of particle, determined by the forces and its velocity.

///

Returns

Whether there was a change in state

Implemented in DynamicParticle, and StaticParticle.

4.2.2.6 refreshState()

```
virtual bool Particle::refreshState ( )  [pure virtual]
```

Refreshes the state of a particle, determined by its velocity, and acceleration.

Returns

Whether there was a change in state

Implemented in StaticParticle, and DynamicParticle.

The documentation for this class was generated from the following files:

- X:/Saját/prog/BME/prog2/Nagyhazi/proba2/AranyMatyasNH/particles.h
- X:/Saját/prog/BME/prog2/Nagyhazi/proba2/AranyMatyasNH/particles.cpp

4.3 ParticleContainer Class Reference

Class for and array of Particle, can refresh the attributes of every item in it.

```
#include <container.h>
```

Public Member Functions

- void **insert** (Particle *item)

  Adds the preallocated particle to the container.
- void newItem (bool isDynamic, float x, float y, unsigned int mass, int charge, ParticleType type=None, float velx=0, float vely=0)

  Adds new particle to the container.
- void deleteItem (size_t const idx)

  Deletes item from the container.
- void calculateCoulombs (float const speed) const

  Calculates every coulomb of between every particle in the container.
- bool refreshPositions (float const speed)

  Refreshes positions of every particle in the system.
- void **refreshState** ()

  Refreshes the overall state of the container, determined by the state of its members.
- void read (const char *filename)

  Builds an array of particle from a file.
- Particle * operator[ ] (size_t idx) const
- size_t **getSize** () const

  Returns the size of the container.
- State **getState** () const

  Returns the overall state of the container.
- void **destroy** ()

  Frees the memory allocated for the container, resetting it.
- **ParticleContainer** (ParticleContainer const &)

  Copy constructor not designed to be used.

4.3.1 Detailed Description

Class for and array of Particle, can refresh the attributes of every item in it.

4.3.2 Member Function Documentation

4.3.2.1 calculateCoulombs()

```
void ParticleContainer::calculateCoulombs (
            float const speed ) const
```

Calculates every coulomb of between every particle in the container.

Parameters

speed	The speed of the rate of change

4.3.2.2 deleteItem()

```
void ParticleContainer::deleteItem (
            size_t const idx )
```

Deletes item from the container.

Parameters

idx	The index of the item to delete

4.3.2.3 newItem()

```
void ParticleContainer::newItem (
            bool isDynamic,
            float x,
            float y,
            unsigned int mass,
            int charge,
            ParticleType type = None,
            float velx = 0,
            float vely = 0 )
```

Adds new particle to the container.

Parameters

isDynamic	If the new particle should be dynamic
x	starting X coordinate
y	starting Y coordinate
mass	The mass of the new particle

Parameters

charge	The charge of the new particle
type	The predefined type of the new particle
velx	The starting x velocity of new particle (ignored if static)
vely	The starting y velocity of new particle (ignored if static)

4.3.2.4 operator[]()

```
Particle * ParticleContainer::operator[] (
            size_t idx ) const
```

Parameters

idx	Index of the particle to be returned

Returns

Returns the particle at the given index

4.3.2.5 read()

```
void ParticleContainer::read (
            const char * filename )
```

Builds an array of particle from a file.

Parameters

filename	Name of file to be read from

4.3.2.6 refreshPositions()

```
bool ParticleContainer::refreshPositions (
            float const speed )
```

Refreshes positions of every particle in the system.

Parameters

speed	The speed of the rate of change

If there was a change in the state of some Particle of the system: returns true

Returns

The documentation for this class was generated from the following files:

- X:/Saját/prog/BME/prog2/Nagyhazi/proba2/AranyMatyasNH/container.h
- X:/Saját/prog/BME/prog2/Nagyhazi/proba2/AranyMatyasNH/container.cpp

4.4 RunningSimulation Class Reference

Class directly for running an already existing simulation Singleton class with only static members.

`#include <runsimulation.h>`

Static Public Member Functions

- static bool **saveMode** ()

  Returns what the current mode is: saving or no saving.
- static void **saveMode** (bool mode)

  Sets whether the simulation should save.
- static void **maxIteration** (long int maxit)

  Sets the limit of iterations.
- static void **setup** (Simulation *sim, bool resp=false, long int maxit=10000000)

  Sets up the simulation to later run.
- static bool **reachedMaxIt** ()

  Returns whether the simulation has reached maximum iterations.
- static Simulation * **getSimulation** ()

  Returns the pointer of the current simulation.
- static State startNonVisual (bool displaytext=true)

  Starts running the simulation, with no visuals, until it ends.
- static State startVisual ()

  Starts running the simulation with SFML, until it has ended.
- static void changeSimulation (Simulation *newsim, bool resp=false)

  Changes the simulation associated with this class, resetting all its attributes.
- static void newSystem (const char *filename)

  Changes the simulation to simulate a new system from a new file.
- static void **deleteSim** ()

4.4.1 Detailed Description

Class directly for running an already existing simulation Singleton class with only static members.

4.4.2 Member Function Documentation

4.4.2.1 changeSimulation()

```
static void RunningSimulation::changeSimulation (
            Simulation * newsim,
            bool resp = false )  [inline], [static]
```

Changes the simulation associated with this class, resetting all its attributes.

Parameters

newsim	Pointer to the simulation to connect
resp	True if the simulation was dinamically allocated

4.4.2.2 newSystem()

```
static void RunningSimulation::newSystem (
            const char * filename )  [inline], [static]
```

Changes the simulation to simulate a new system from a new file.

Parameters

filename	Name of file to be read from

summary> If responsibility is true, then it deletes the current simulation and then changes it to nullptr Resets everything /summary>

4.4.2.3 startNonVisual()

```
State RunningSimulation::startNonVisual (
            bool displaytext = true )  [static]
```

Starts running the simulation, with no visuals, until it ends.

Returns

Returns the end state of the simulation

4.4.2.4 startVisual()

```
State RunningSimulation::startVisual ( )  [static]
```

Starts running the simulation with SFML, until it has ended.

Returns

Returns the end state of the simulation

SFML font for the displayable text

User interface info text

The documentation for this class was generated from the following files:

- X:/Saját/prog/BME/prog2/Nagyhazi/proba2/AranyMatyasNH/runsimulation.h
- X:/Saját/prog/BME/prog2/Nagyhazi/proba2/AranyMatyasNH/runsimulation.cpp

4.5 Simulation Class Reference

Simulation for systems of Particle.

`#include <simulation.h>`

Public Member Functions

- **Simulation** (const char *filename=nullptr, const float basespeed=0.008f)

  Constructs the simulation from the filename given.
- const Particle * operator[ ] (size_t idx) const

  Index operator.
- size_t getSize () const

  Returns the size of the simulation, the number of Particle in it.
- std::string getName () const

  Returns the name of the simulation.
- State getEndState () const

  Returns the end state of the simulation, calculated by its members' states.
- bool **isValid** () const

  If the simulation is valid and ready.
- void print (std::ostream &os) const

  Prints out the array of Particle of the simulation.
- void **reduceSpeed** ()

  Reduces the speed of the simulation.
- void **increaseSpeed** ()

  Increases the speed of the simulation.
- bool iterate ()

  Iterates the simulation by one step, refreshing the position, velocity and state of every particle.
- void readFromFile (const char *filename)

  Reads the particle data from a file, to start the simulation with.
- void **resetSim** ()

  Resets the object, allowing it to simulate new systems.
- void saveToFile (const char *filename=nullptr) const

  Writes the exact state of the system in a file, which is able to be read back, to start the simulation again.
- void **addNewParticle** ()

  Adds a new particle to the simulation, with random attributes.
- **operator int** () const

  Casting int returns the iteration number.
- sf::Text const getText () const

  Returns the SFML text of the simulation to be drawn.
- sf::CircleShape const getShape (size_t idx) const

  Returns the SFML shape of the indexed particle.

4.5.1 Detailed Description

Simulation for systems of Particle.

4.5.2 Member Function Documentation

4.5.2.1 getEndState()

```
State Simulation::getEndState ( ) const
```

Returns the end state of the simulation, calculated by its members' states.

Returns

4.5.2.2 getName()

```
std::string Simulation::getName ( ) const  [inline]
```

Returns the name of the simulation.

Returns

4.5.2.3 getShape()

```
sf::CircleShape const Simulation::getShape (
            size_t idx ) const  [inline]
```

Returns the SFML shape of the indexed particle.

Parameters

idx	Index of needed particle

Returns

4.5.2.4 getSize()

```
size_t Simulation::getSize ( ) const  [inline]
```

Returns the size of the simulation, the number of Particle in it.

Returns

4.5.2.5 getText()

```
sf::Text const Simulation::getText ( ) const
```

Returns the SFML text of the simulation to be drawn.

Returns

4.5.2.6 iterate()

```
bool Simulation::iterate ( )
```

Iterates the simulation by one step, refreshing the position, velocity and state of every particle.

Returns

If there was a change in state

4.5.2.7 operator[]()

```
const Particle * Simulation::operator[] (
            size_t idx ) const  [inline]
```

Index operator.

Parameters

idx	Index

Returns

returns the particle in the particle given by its index

4.5.2.8 print()

```
void Simulation::print (
            std::ostream & os ) const  [inline]
```

Prints out the array of Particle of the simulation.

Parameters

os	

4.5.2.9 readFromFile()

```
void Simulation::readFromFile (
            const char * filename )  [inline]
```

Reads the particle data from a file, to start the simulation with.

Parameters

filename	Name of file to be read from

4.5.2.10 saveToFile()

```
void Simulation::saveToFile (
            const char * filename = nullptr ) const
```

Writes the exact state of the system in a file, which is able to be read back, to start the simulation again.

Parameters

filename	Name of file to be written in

The documentation for this class was generated from the following files:

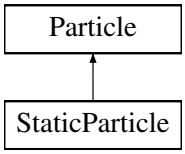
- X:/Saját/prog/BME/prog2/Nagyhazi/proba2/AranyMatyasNH/simulation.h
- X:/Saját/prog/BME/prog2/Nagyhazi/proba2/AranyMatyasNH/simulation.cpp

4.6 StaticParticle Class Reference

Static Particle that don't move.

`#include <particles.h>`

Inheritance diagram for StaticParticle:



Public Member Functions

- **StaticParticle** (float x, float y, unsigned int mass, int charge, ParticleType type=None)

  Creates object with the given attributes.
- void calcGravity (Particle const &)

  Does nothing for a static particle.

- Collision calcCoulomb (Particle const &)

  Does nothing for a static particle.
- bool refreshPos (float const)

  Does nothing for a static particle.
- bool refreshState ()

  Does nothing for a static particle.
- MoveTypes getType () const

  Returns Dynamic / Static.
- Vector getVel () const

  Return the velocity of particle.

Public Member Functions inherited from Particle

- **Particle** (float x, float y, unsigned int mass, int charge, ParticleType type=None)

  Constructor with default particle type of 'None'.
- **Particle** (Vector pos, unsigned int mass, int charge, ParticleType type=None)

  Constructor with default particle type of 'None'.
- Vector **getPos** () const

  Returns the position vector of the particle.
- unsigned int **getMass** () const

  Returns the mass of the particle.
- int **getCharge** () const

  Returns the charge of the particle.
- bool **isOutOfView** () const

  Returns whether the particle is OutOfView = far out of screen.
- bool **isAccelerating** () const

  Returns whether the particle is Accelerating.
- bool **isBalanced** () const

  Returns whether the particle is Balanced.
- bool **isCalm** () const

  Returns whether the particle is Calm.
- State **getState** () const

  Returns the state of the particle.
- std::string **ParticleString** () const

  Provides the string description of a particle.
- ParticleType **getPType** () const

  Returns the particle type of the particle.
- sf::CircleShape **getShape** () const

  Returns with the shape of the particle.

Additional Inherited Members

Protected Attributes inherited from Particle

- Vector **pos**

  Position of the particle.
- unsigned int **mass**

  Mass of the particle.
- int **charge**

Charge of the particle.

- State **state**

  State of the particle.

- ParticleType **type**

  Predefined type of particle.

- sf::CircleShape **circ**

  SFML Circleshape representation of the particle.

4.6.1 Detailed Description

Static Particle that don't move.

4.6.2 Member Function Documentation

4.6.2.1 calcCoulomb()

```
Collision StaticParticle::calcCoulomb (
            Particle const &  )  [inline], [virtual]
```

Does nothing for a static particle.

Implements Particle.

4.6.2.2 calcGravity()

```
void StaticParticle::calcGravity (
            Particle const &  )  [inline], [virtual]
```

Does nothing for a static particle.

Implements Particle.

4.6.2.3 getType()

```
MoveTypes StaticParticle::getType ( ) const  [inline], [virtual]
```

Returns Dynamic / Static.

Implements Particle.

4.6.2.4 getVel()

```
Vector StaticParticle::getVel ( ) const  [inline], [virtual]
```

Return the velocity of particle.

Returns

Implements Particle.

4.6.2.5 refreshPos()

```
bool StaticParticle::refreshPos (
            float const  )  [inline], [virtual]
```

Does nothing for a static particle.

Implements Particle.

4.6.2.6 refreshState()

```
bool StaticParticle::refreshState ( )  [inline], [virtual]
```

Does nothing for a static particle.

summary> Return Stacic type /summary>

Implements Particle.

The documentation for this class was generated from the following file:

- X:/Saját/prog/BME/prog2/Nagyhazi/proba2/AranyMatyasNH/particles.h

4.7 Vector Class Reference

Vector class used for representing physics related vector values, with X and Y coordinates.

```
#include <vector.h>
```

Public Member Functions

- Vector (float x=0, float y=0)
  Constructor for vectors, usable as default constructor.
- float **getX** () const
  Returns the X coordinate of a vector.
- float **getY** () const
  Returns the Y coordinate of a vector.
- float **abs** () const
  Returns the abs valuse of a vector.
- Vector **operator+** (Vector const &rhs) const
  Adds two vectors together.
- Vector **operator-** (Vector const &rhs) const
  Substracts two vectors.
- float operator% (Vector const &rhs) const
  Gives the distance between two vectors.
- float **distance** (Vector const &rhs) const
  Returns the distance between two vectors.
- Vector **operator**∗ (float rhs) const
  Multiplies vector by a float.

- Vector **operator**/ (float rhs) const

  Divides vector by a float.
- Vector **operator**/ (int rhs) const

  Divides vector by an int.
- Vector **operator**/ (unsigned int rhs) const

  Divides vector by an unsigned int.
- Vector **operator**∗ (int rhs) const

  Multiplies vector by an int.
- Vector **operator**∗ (unsigned int rhs) const

  Multiplies vector by an unsigned int.
- bool **operator==** (const Vector &rhs) const

  Checks if two vectors are close enough to each other.
- void rotate (double phi)

  Rotates the vector.

Public Attributes

- float **x**

  X coordinate of a vector.
- float **y**

  Y coordinate of a vector.

4.7.1 Detailed Description

Vector class used for representing physics related vector values, with X and Y coordinates.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 Vector()

```
Vector::Vector (
            float x = 0,
            float y = 0 )  [inline]
```

Constructor for vectors, usable as default constructor.

Parameters

x	X coordinate
y	Y coordinate

4.7.3 Member Function Documentation

4.7.3.1 operator%()

```
float Vector::operator% (
            Vector const & rhs ) const
```

Gives the distance between two vectors.

Parameters

rhs	

Returns

Distance between the two points

4.7.3.2 rotate()

```
void Vector::rotate (
            double phi )
```

Rotates the vector.

Parameters

phi	Angle to be rotated by

The documentation for this class was generated from the following files:

- X:/Saját/prog/BME/prog2/Nagyhazi/proba2/AranyMatyasNH/vector.h
- X:/Saját/prog/BME/prog2/Nagyhazi/proba2/AranyMatyasNH/vector.cpp

Chapter 5

File Documentation

5.1 container.h

```
00001 #ifndef CONTAINER_H
00002 #define CONTAINER_H
00003
00004 #include "particles.h"
00005 #include <iostream>
00006
00010 class ParticleContainer
00011 {
00013     Particle** part;
00015     size_t size;
00017     size_t mem;
00019     State state;
00020
00024     void increaseMemory();
00025
00026 public:
00027
00028     ParticleContainer();
00029
00033     void insert(Particle* item) {
00034         part[size++] = item;
00035         increaseMemory();
00036     }
00037
00049     void newItem(bool isDynamic, float x, float y, unsigned int mass, int charge, ParticleType type =
      None, float velx = 0, float vely = 0 );
00050
00055     void deleteItem(size_t const idx);
00056
00061     void calculateCoulombs(float const speed) const;
00062
00069     bool refreshPositions(float const speed);
00070
00074     void refreshState();
00075
00080     void read(const char* filename);
00081
00087     Particle* operator[](size_t idx) const;
00088
00092     size_t getSize() const { return size; }
00093
00097     State getState() const { return state; }
00098
00102     void destroy();
00103
00104     ~ParticleContainer();
00105
00109     ParticleContainer(ParticleContainer const&) : ParticleContainer() {  std::cout « "Nincs masolo
      konstruktor!" « std::endl; }
00110
00111 };
00112
00119 std::ostream& operator«(std::ostream& os, ParticleContainer const& rhs);
00120
00121
00122 #endif
```

5.2 CPORTA.h

```
00001
00002 //#define CPORTA
00003
00004 //#define TESTMODE
```

5.3 particles.h

```
00001 #ifndef Particle_H
00002 #define Particle_H
00003
00004 #include "CPORTA.h"
00005 #include "window.h"
00006 #include <cmath>
00007 #include "systemstates.h"
00008
00009 #ifndef CPORTA
00010 #include <SFML/Graphics.hpp>
00011 #endif
00012
00013 #include "vector.h"
00014
00018 class Particle {
00019 protected:
00021     Vector pos;
00023     unsigned int mass;
00025     int charge;
00027     State state;
00029     ParticleType type;
00030 #ifndef CPORTA
00032     sf::CircleShape circ;
00033 #endif
00034
00035 public:
00039     Particle(float x, float y, unsigned int mass, int charge, ParticleType type = None);
00040
00044     Particle(Vector pos, unsigned int mass, int charge, ParticleType type = None) :
      Particle(pos.x,pos.y, mass, charge, type) {}
00045
00049     Vector getPos() const { return pos; }
00050
00054     unsigned int getMass() const { return mass; }
00055
00059     int getCharge() const { return charge; }
00060
00064     bool isOutOfView() const { return state == OutOfView; }
00065
00069     bool isAccelerating() const { return state == Accelerating; }
00070
00074     bool isBalanced() const { return state == Balanced; }
00075
00079     bool isCalm() const { return state == Calm; }
00080
00084     State getState() const { return state; }
00085
00089     std::string ParticleString() const;
00090
00094     virtual MoveTypes getType() const = 0;
00095
00099     virtual Vector getVel() const = 0;
00100
00104     ParticleType getPType() const { return type; }
00105 #ifndef CPORTA
00109     sf::CircleShape getShape() const { return circ; }
00110 #endif
00111
00115     virtual void calcGravity(Particle const& other)=0;
00116
00120     virtual Collision calcCoulomb(Particle const& other)=0;
00121
00126     virtual bool refreshPos(float const rate) = 0;
00127
00132     virtual bool refreshState() = 0;
00133
00134     virtual ~Particle() {}
00135
00136 };
00137
00141 class StaticParticle :public Particle {
00142
00143 public:
```

```
00147     StaticParticle(float x, float y, unsigned int mass, int charge, ParticleType type = None) :
      Particle(x, y, mass, charge, type) {
00148         state = Calm;
00149         if (abs(pos.x) > resX * outofviewratio || abs(pos.y) > resY * outofviewratio) state =
      OutOfView;
00150     }
00151
00155     void calcGravity(Particle const&) {}
00156
00160     Collision calcCoulomb(Particle const&) { return NoCollision; }
00161
00165     bool refreshPos(float const) { return state == OutOfView; }
00166
00170     bool refreshState() { return false;  }
00171
00175     MoveTypes getType() const{ return Static; }
00176
00181     Vector getVel() const { return Vector(0,0); }
00182
00183 };
00184
00188 class DynamicParticle :public Particle {
00190     Vector vel;
00192     Vector force;
00193
00194 public:
00198     DynamicParticle(float x, float y, unsigned int mass, int charge, float velx = 0, float vely = 0,
      ParticleType type = None) : Particle(x, y, mass, charge, type), vel(Vector(velx, vely)),
      force(Vector(0.0,0.0)) {}
00199
00203     void calcGravity(Particle const& other);
00204
00209     Collision calcCoulomb(Particle const& other);
00210
00216     bool refreshPos(float const rate);
00217
00222     bool refreshState();
00223
00227     MoveTypes getType() const { return Dynamic; }
00228
00233     Vector getVel() const { return vel; }
00234
00235 };
00236
00237
00238 #endif
```

5.4 runsimulation.h

```
00001 #ifndef RUNSIMULATION_H
00002 #define RUNSIMULATION_H
00003
00004 #include "simulation.h"
00005 #include <chrono>
00006
00007
00012 class RunningSimulation
00013 {
00015     static Simulation* simulation;
00017     static bool pause;
00019     static bool end;
00021     static long int maxiteration;
00023     static bool reachedTooMany;
00025     static bool responsibility;
00027     static bool savemode;
00028
00029 #ifndef CPORTA
00030
00035     static bool userInput(sf::Event& event);
00036 #endif
00040     static void resetAttributes() {
00041         pause = false; end = false; reachedTooMany = false;
00042     }
00043
00047     RunningSimulation();
00048 public:
00049
00053     static bool saveMode() { return savemode; }
00054
00058     static void saveMode(bool mode) { savemode = mode; }
00059
00063     static void maxIteration(long int maxit) { maxiteration = maxit; }
00064
```

```
00068     static void setup(Simulation* sim, bool resp = false,long int maxit = 10000000);
00069
00073     static bool reachedMaxIt() { return reachedTooMany; }
00074
00078     static Simulation* getSimulation() { return simulation; }
00079
00084     static State startNonVisual(bool displaytext = true);
00085
00086 #ifndef CPORTA
00091     static State startVisual();
00092 #endif
00098     static void changeSimulation(Simulation* newsim, bool resp = false) {
00099         if (responsibility) delete simulation;
00100         responsibility = resp;
00101         simulation = newsim;
00102         resetAttributes();
00103     }
00104
00109     static void newSystem(const char* filename)
00110     {
00111         if (simulation == nullptr) throw "No simulation to change!";
00112         simulation->resetSim();
00113         simulation->readFromFile(filename);
00114         resetAttributes();
00115     }
00116
00121     static void deleteSim();
00122
00123 };
00124
00125
00126 #endif
```

5.5 simulation.h

```
00001 #ifndef SIMULATION_H
00002 #define SIMULATION_H
00003
00004 #include "container.h"
00005
00009 class Simulation {
00011     bool valid;
00013     std::string name;
00015     float speed;
00017     const float basespeed;
00019     long int iteration;
00021     int sinceLastChange;
00023     State currentState;
00025     State endState;
00027     ParticleContainer container;
00028 #ifndef CPORTA
00030     sf::Text text;
00032     sf::Font font;
00033
00037     void refreshText();
00038 #endif
00042     void refreshEndState();
00043
00047     void handleCollision(Collision type, size_t i, size_t j);
00048
00049 public:
00053     Simulation(const char* filename = nullptr, const float basespeed = 0.008f);
00054
00058     const
00059         Particle* operator[](size_t idx) const { return container[idx]; }
00060
00065     size_t getSize() const { return container.getSize(); }
00066
00071     std::string getName() const { return name; }
00072
00077     State getEndState() const;
00078
00082     bool isValid() const { return valid; }
00083
00088     void print(std::ostream& os) const { os << container; }
00089
00093     void reduceSpeed() { speed = static_cast<float>(0.5f * basespeed > speed * 0.9f ? 0.5f * basespeed
      : speed * 0.9f ); }
00094
00098     void increaseSpeed() { speed = static_cast<float>(2.0f * basespeed < speed * 1.111f ? 2.0f *
     basespeed : speed * 1.111f); }
00099
00104     bool iterate();
```

```
00105
00110     void readFromFile(const char* filename) {
00111         container.read(filename);
00112         if (filename != nullptr)
00113         {
00114             valid = true;
00115             name = filename;
00116             name = name.substr(0, name.find('.'));
00117         }
00118     #ifndef CPORTA
00119         refreshText();
00120     #endif
00121     }
00122
00126     void resetSim();
00127
00132     void saveToFile(const char* filename = nullptr) const;
00133
00137     void addNewParticle();
00138
00142     operator int() const { return iteration; }
00143
00144 #ifndef CPORTA
00149     sf::Text const getText() const;
00150
00156     sf::CircleShape const getShape(size_t idx) const { return container[idx]->getShape(); }
00157 #endif
00158
00159 };
00160
00161 std::ostream& operator<<(std::ostream& os, const Simulation& rhs);
00162
00163
00164 #endif
```

5.6 systemstates.h

```
00001 #ifndef SYSTEM_H
00002 #define SYSTEM_H
00003
00004 #include <iostream>
00005
00009 enum Collision { NoCollision, ProtonElectron, ProtonNeutron, PositronElectron };
00010
00014 enum State { OutOfView, Accelerating, Balanced, Calm, Orbit };
00015
00019 enum MoveTypes { Static, Dynamic };
00020
00024 enum ParticleType { None, Proton, Electron, Neutron, Positron, Neutrino, Deuterium };
00025
00031 const char* getStateString(State const state);
00032
00038 const char* getParticleTypeString(ParticleType const type);
00039
00040 std::istream& operator>>(std::istream& is, ParticleType& type);
00041
00042 std::ostream& operator<<(std::ostream& os, ParticleType const type);
00043
00044 std::ostream& operator<<(std::ostream& os, State const type);
00045
00046 #endif
```

5.7 vector.h

```
00001 #ifndef VECTOR_H
00002 #define VECTOR_H
00003
00007 class Vector {
00008 public:
00010     float x;
00012     float y;
00013
00019     Vector(float x = 0, float y = 0) : x(x), y(y) {}
00020
00022     float getX() const { return x; }
00024     float getY() const { return y; }
00026     float abs() const;
00027
00031     Vector operator+(Vector const& rhs) const;
```

```
00032
00036     Vector operator-(Vector const& rhs) const;
00037
00046     float operator%(Vector const& rhs) const;
00047
00051     float distance(Vector const& rhs) const { return this->operator%(rhs); }
00052
00056     Vector operator*(float rhs) const;
00057
00061     Vector operator/(float rhs) const;
00062
00066     Vector operator/(int rhs) const;
00067
00071     Vector operator/(unsigned int rhs) const;
00072
00076     Vector operator*(int rhs) const;
00077
00081     Vector operator*(unsigned int rhs) const;
00082
00086     bool operator==(const Vector& rhs) const;
00087
00092     void rotate(double phi);
00093 };
00094
00098 Vector operator*(const float& lhs, const Vector& rhs);
00099
00100
00101 #endif
```

5.8 window.h

```
00001 #ifndef WINDOW_H
00002 #define WINDOW_H
00003
00007 const int resX = 1000;
00011 const int resY = 1000;
00012
00016 const float epsilon = 0.005f;
00017
00021 const float outofviewratio = 3;
00022
00026 const float coulombscale = 6000;
00027
00031 const float accelerationscale = 350;
00032
00034 const float gravityscale = 0.001f;
00035
00036 #endif
```

Index