Programozás alapjai 2

NHF Specifikáció(*Bővített*)

Fótyék Róbert

DNHR9M

Leírás: A program egy részecskék közötti kölcsönhatásokat modellező egyszerű, kétdimenziós szimuláció. A kezdeti állapot beolvasás után képes tovább léptetni a szimulációt tetszőleges időtartamig. A nagyházi követelményeit figyelembe véve *nem* lesz grafikus felülete, a kimenete a részecskék pillanatnyi tulajdonságai, melyet a program egy fájlba ír.

I/O: A program szimuláció közben minden időegységben egy, az alábbi formátumú fájlba írja a részecskék adatait (ez lehetővé teszi egy grafikus ”visszajátszó” program elkészítését). A bemenet egy ugyanilyen formátumú fájl, és a program a fájlban található utolsó időegységről folytatja a szimulációt, tovább írva a fájlt.

Feature-ök: egy részecske deklarálásakor mindig megadandó a tömege, a töltése, illetve beállatható hogy részt vesz-e a gravitációs kölcsönhatásban. Ezen konstansokon kívül beállítható kezdetleges pozíciója és koordinátái. Ilyen részecskékből tetszőleges számú megadható. Ez a kezdeti állapot megadható fájlban, vagy a program CLI-jén keresztül.

Fájl formátum:

{

Tömeg,

PozícióX PozícióY,

SebességX SebességY,

Töltés,

GravitációsBoolean,

Tömeg,

PozícióX PozícióY,

SebességX SebességY,

Töltés,

GravitációsBoolean,

…

…

…

}

{

…

}

.

.

.

Így egy bracket {} egy időegység, és azon belül a részecskék a szabvány alapján következnek egymás után. A sablonban az új sor csak a szépség miatt van, egyébként minden whitespace ugyanaz.

Program futtatása:

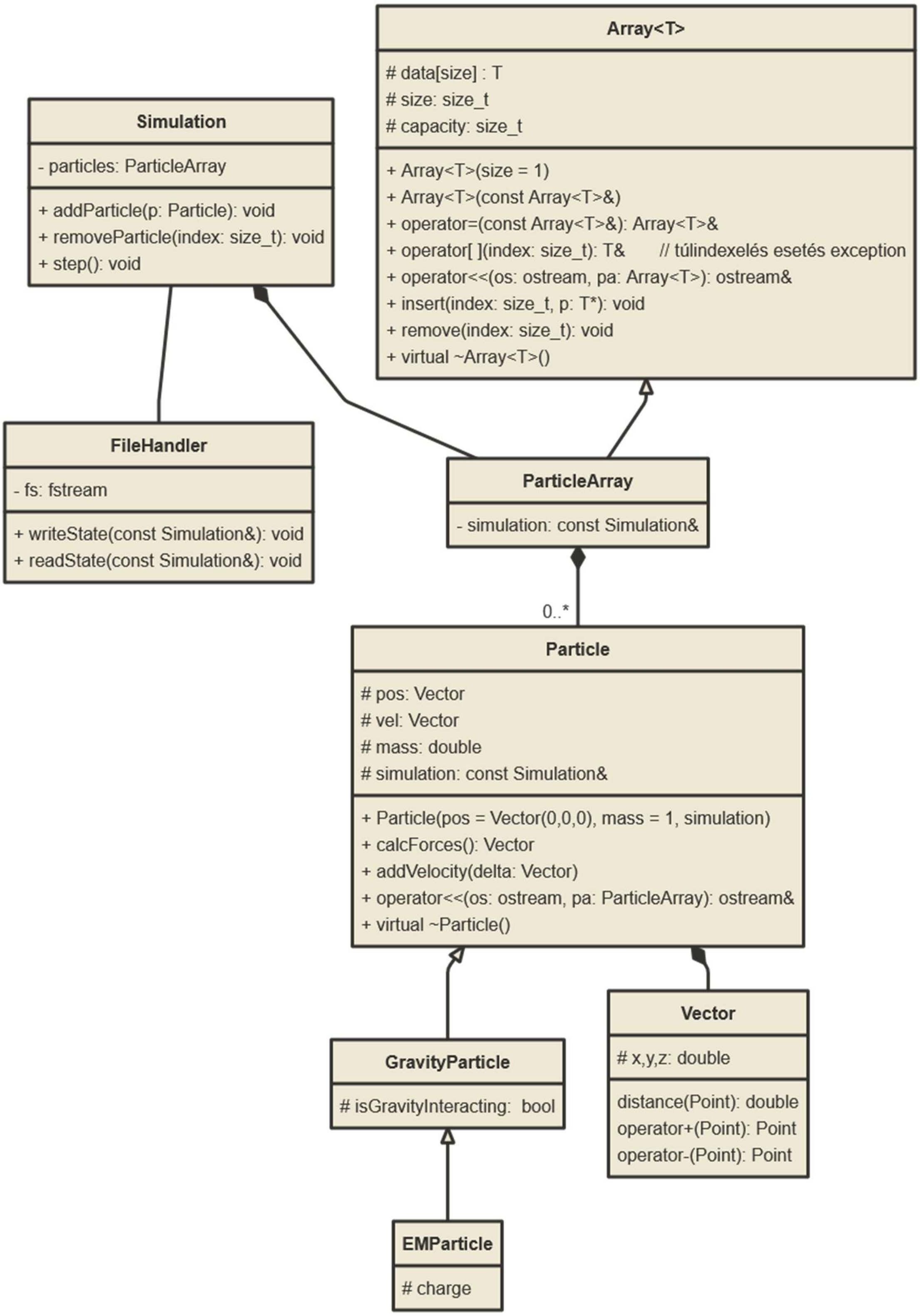
A program argumentumként megkapja az input fájl-t, majd onnan menürendszerből kezelhető.

Menüpontok:

- ListParticles
- AddParticle
- RemoveParticle
- Simulate *időtartam*
- *Exit*

A szimulált időtartam után visszalép a program a menübe.

Osztálydiagram


Array<T>
# data[size] : T
# size: size_t
# capacity: size_t
+ Array<T>(size = 1)
+ Array<T>(const Array<T>&)
+ operator=(const Array<T>&): Array<T>&
+ operator[ ](index: size_t): T& // túlindexelés esetés exception
+ operator<<(os: ostream, pa: Array<T>): ostream&
+ insert(index: size_t, p: T*): void
+ remove(index: size_t): void
+ virtual ~Array<T>()
Simulation
- particles: ParticleArray
+ addParticle(p: Particle): void
+ removeParticle(index: size_t): void
+ step(): void
FileHandler
- fs: fstream
+ writeState(const Simulation&): void
+ readState(const Simulation&): void
ParticleArray
- simulation: const Simulation&
0..*
Particle
# pos: Vector
# vel: Vector
# mass: double
# simulation: const Simulation&
+ Particle(pos = Vector(0,0,0), mass = 1, simulation)
+ calcForces(): Vector
+ addVelocity(delta: Vector)
+ operator<<(os: ostream, pa: ParticleArray): ostream&
+ virtual ~Particle()
GravityParticle
# isGravityInteracting: bool
Vector
# x,y,z: double
distance(Point): double
operator+(Point): Point
operator-(Point): Point
EMParticle
# charge

Megjegyzések:

- A ParticleArray illetve Particle osztályoknak komponense a Simulation osztály, hogy ne lehessen részecske hozzá tartozó szimuláció nélkül.
- Accesorokat, triviális konstruktorokat, destruktorokat nem jelöltem.

Fő algoritmus: szimulációs lépés kiszámítása

```
For each Particle i in ParticleArray:
     Vector force(0,0,0)
     For each Particle j != i in ParticleArray:
          Vector force += calculateEMforce(i, j)
     i.addVelocity(force/i.mass())
```

Particular

Generated by Doxygen 1.13.2

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

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

Chapter 3

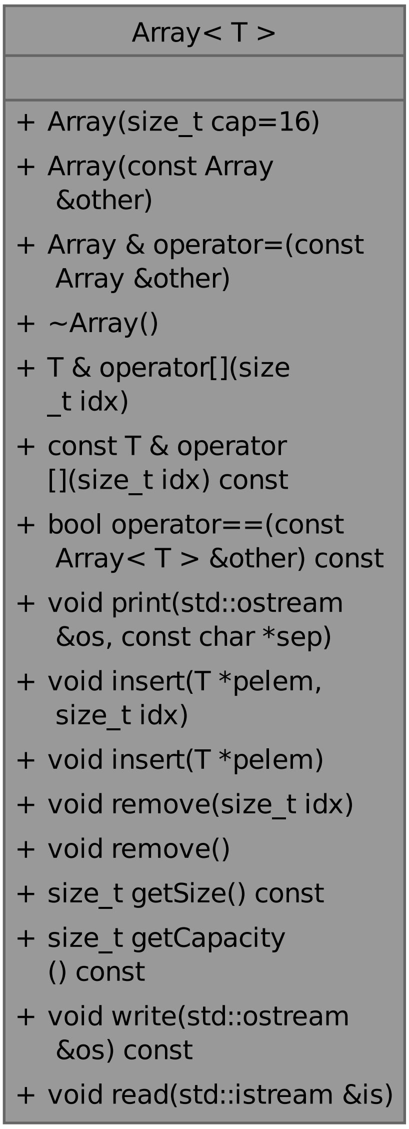
Class Documentation

3.1 Array< T > Class Template Reference

Generic dynamic array class template.

```
#include <Array.hpp>
```

Collaboration diagram for Array< T >:


Array< T >
+ Array(size_t cap=16)
+ Array(const Array &other)
+ Array & operator=(const Array &other)
+ ~Array()
+ T & operator[](size_t idx)
+ const T & operator[](size_t idx) const
+ bool operator==(const Array< T > &other) const
+ void print(std::ostream &os, const char *sep)
+ void insert(T *pelem, size_t idx)
+ void insert(T *pelem)
+ void remove(size_t idx)
+ void remove()
+ size_t getSize() const
+ size_t getCapacity() const
+ void write(std::ostream &os) const
+ void read(std::istream &is)

Public Member Functions

- Array (size_t cap=16)

    Constructor.
- Array (const Array &other)

    Copy constructor.
- Array & operator= (const Array &other)

    Copy assignment.
- ∼**Array** ()

    Destructor.

- T & operator[ ] (size_t idx)

  Element access (non-const)
- const T & operator[ ] (size_t idx) const

  Element access (const)
- bool operator== (const Array< T > &other) const

  element by element comparison
- void **print** (std::ostream &os, const char ∗sep)
- void insert (T ∗pelem, size_t idx)

  Insert an element at a specific index.
- void insert (T ∗pelem)

  Insert an element at the end of the array.
- void remove (size_t idx)

  Remove element at a specific index.
- void **remove** ()

  Remove the last element.
- size_t getSize () const

  Get the current size.
- size_t getCapacity () const

  Get the current capacity.
- void write (std::ostream &os) const

  array write to ostream
- void read (std::istream &is)

  array read from istream

3.1.1 Detailed Description

template<typename T>
class Array< T >

Generic dynamic array class template.

Template Parameters

T	Type of elements stored

3.1.2 Constructor & Destructor Documentation

3.1.2.1 Array() [1/2]

```
template<typename T>
Array< T >::Array (
            size_t cap = 16)  [inline]
```

Constructor.

Parameters

cap	Initial capacity

3.1.2.2 Array() [2/2]

```
template<typename T>
Array< T >::Array (
             const Array< T > & other)  [inline]
```

Copy constructor.

Parameters

other	Array to copy

3.1.3 Member Function Documentation

3.1.3.1 getCapacity()

```
template<typename T>
size_t Array< T >::getCapacity () const  [inline]
```

Get the current capacity.

Returns

size_t Allocated capacity

3.1.3.2 getSize()

```
template<typename T>
size_t Array< T >::getSize () const  [inline]
```

Get the current size.

Returns

size_t Number of elements

3.1.3.3 insert() [1/2]

```
template<typename T>
void Array< T >::insert (
             T * pelem)  [inline]
```

Insert an element at the end of the array.

Parameters

pelem	Element to insert

3.1.3.4 insert() [2/2]

```
template<typename T>
void Array< T >::insert (
             T * pelem,
             size_t idx)  [inline]
```

Insert an element at a specific index.

Exceptions

std::out_of_range	when index is outside used array

Parameters

pelem	allocated element pointer (usage: arr.insert(new T);)
idx	Index at which to insert

3.1.3.5 operator=()

```
template<typename T>
Array & Array< T >::operator= (
            const Array< T > & other)   [inline]
```

Copy assignment.

Parameters

other	Array to copy

Returns

Array& Reference to this

3.1.3.6 operator==()

```
template<typename T>
bool Array< T >::operator== (
            const Array< T > & other) const   [inline]
```

element by element comparison

Parameters

other	array to compare

Returns

bool

3.1.3.7 operator[]() [1/2]

```
template<typename T>
T & Array< T >::operator[] (
            size_t idx)   [inline]
```

Element access (non-const)

Exceptions

std::out_of_range	when index is outside used array

Parameters

idx	Index

Returns

T& Reference to element

3.1.3.8 operator[]() [2/2]

```
template<typename T>
const T & Array< T >::operator[ ] (
             size_t idx) const  [inline]
```

Element access (const)

Exceptions

std::out_of_range	when index is outside used array

Parameters

idx	Index

Returns

const T& Const reference to element

3.1.3.9 read()

```
template<typename T>
void Array< T >::read (
             std::istream & is)  [inline]
```

array read from istream

Parameters

is	

3.1.3.10 remove()

```
template<typename T>
void Array< T >::remove (
             size_t idx)  [inline]
```

Remove element at a specific index.

Exceptions

std::out_of_range	when index is outside used array

Parameters

idx	Index to remove

3.1.3.11 write()

```
template<typename T>
void Array< T >::write (
             std::ostream & os) const  [inline]
```

array write to ostream

Parameters

os	ostream to write to

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

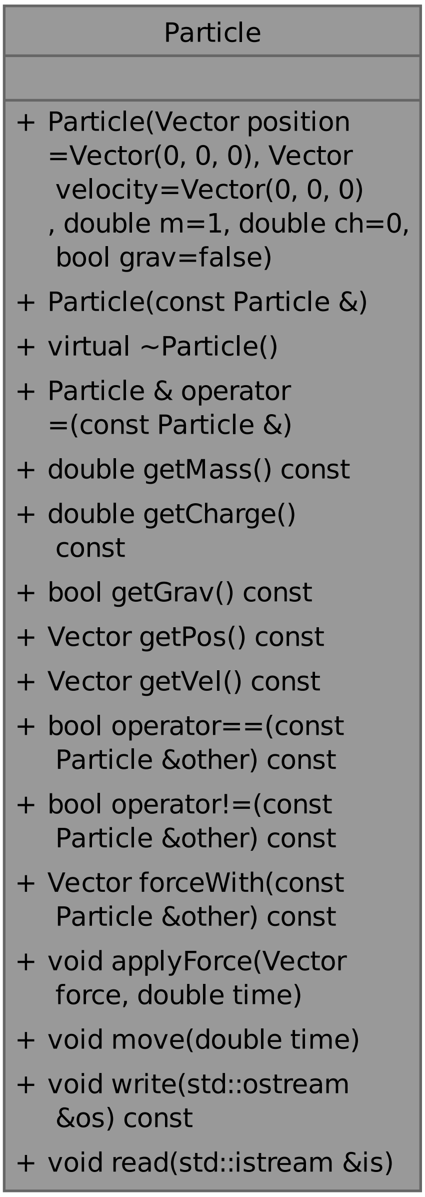
- inc/Array.hpp

3.2 Particle Class Reference

non-interacting particle class.

```
#include <Particle.hpp>
```

Collaboration diagram for Particle:


Particle
+ Particle(Vector position =Vector(0, 0, 0), Vector velocity=Vector(0, 0, 0) , double m=1, double ch=0, bool grav=false)
+ Particle(const Particle &)
+ virtual ~Particle()
+ Particle & operator =(const Particle &)
+ double getMass() const
+ double getCharge() const
+ bool getGrav() const
+ Vector getPos() const
+ Vector getVel() const
+ bool operator==(const Particle &other) const
+ bool operator!=(const Particle &other) const
+ Vector forceWith(const Particle &other) const
+ void applyForce(Vector force, double time)
+ void move(double time)
+ void write(std::ostream &os) const
+ void read(std::istream &is)

Public Member Functions

- Particle (Vector position=Vector(0, 0, 0), Vector velocity=Vector(0, 0, 0), double m=1, double ch=0, bool grav=false)

  constructor
- **Particle** (const Particle &)

  copy constructor
- virtual ∼**Particle** ()

  virtual destructor

- Particle & **operator=** (const Particle &)

  copy assignment
- double getMass () const

  mass getter
- double getCharge () const

  charge getter
- bool getGrav () const

  grav getter
- Vector getPos () const

  position getter
- Vector getVel () const

  velocity getter
- bool operator== (const Particle &other) const

  equality operator
- bool **operator!=** (const Particle &other) const
- Vector forceWith (const Particle &other) const

  calculates force between two particles.
- void applyForce (Vector force, double time)

  applies force to particle
- void move (double time)

  move the particle based on its velocity vector
- void write (std::ostream &os) const

  write to ostream
- void read (std::istream &is)

  read from ostream

3.2.1 Detailed Description

non-interacting particle class.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 Particle()

```
Particle::Particle (
             Vector position = Vector(0,0,0),
             Vector velocity = Vector(0,0,0),
             double m = 1,
             double ch = 0,
             bool grav = false)
```

constructor

Parameters

position	position of particle
velocity	velocity of particle
m	mass of particle

3.2.3 Member Function Documentation

3.2.3.1 applyForce()

```
void Particle::applyForce (
             Vector force,
             double time)
```

applies force to particle

Parameters

force	force to apply
time	time to apply force for

3.2.3.2 forceWith()

```
Vector Particle::forceWith (
             const Particle & other) const
```

calculates force between two particles.

Parameters

other	other particle

Returns

Vector force vector

3.2.3.3 getCharge()

```
double Particle::getCharge () const  [inline]
```

charge getter

Returns

double charge

3.2.3.4 getGrav()

```
bool Particle::getGrav () const  [inline]
```

grav getter

Returns

grav

3.2.3.5 getMass()

```
double Particle::getMass () const  [inline]
```

mass getter

Returns

double mass attribute

3.2.3.6 getPos()

```
Vector Particle::getPos () const  [inline]
```

position getter

Returns

Vector position attribute

3.2.3.7 getVel()

```
Vector Particle::getVel () const  [inline]
```

velocity getter

Returns

Vector vel attribute

3.2.3.8 move()

```
void Particle::move (
             double time)
```

move the particle based on its velocity vector

Parameters

time	how much time to move the particle for

3.2.3.9 operator==()

```
bool Particle::operator== (
             const Particle & other) const
```

equality operator

Parameters

other	vector to compare

3.2.3.10 read()

```
void Particle::read (
             std::istream & is)
```

read from ostream

Parameters

is	std::istream to read from

3.2.3.11 write()

```
void Particle::write (
             std::ostream & os) const
```

write to ostream

Parameters

os	std::ostream to write to

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

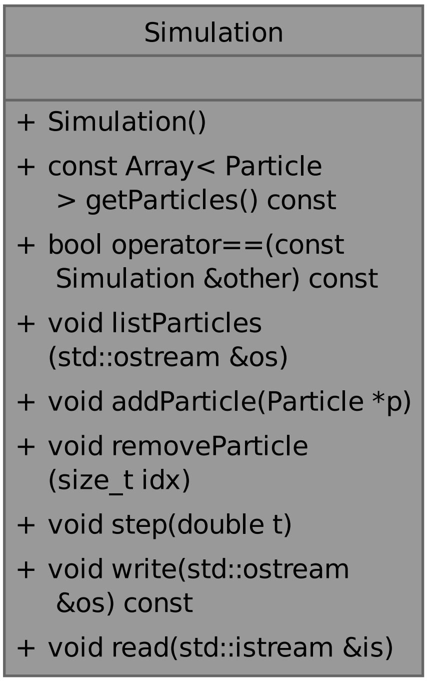
- inc/Particle.hpp
- src/Particle.cpp

3.3 Simulation Class Reference

wrapper and manager for all simulation entities (currently only particle array)

```
#include <Simulation.hpp>
```

Collaboration diagram for Simulation:


Simulation
+ Simulation()
+ const Array< Particle > getParticles() const
+ bool operator==(const Simulation &other) const
+ void listParticles (std::ostream &os)
+ void addParticle(Particle *p)
+ void removeParticle (size_t idx)
+ void step(double t)
+ void write(std::ostream &os) const
+ void read(std::istream &is)

Public Member Functions

- **Simulation** ()

  simulation constructor
- const Array< Particle > getParticles () const

  Get the Particles object (const)
- bool operator== (const Simulation &other) const

  comparison
- void listParticles (std::ostream &os)

  list particles to std::ostream
- void addParticle (Particle ∗p)

  add a particle to simulation
- void removeParticle (size_t idx)

  remove a particle from simulation
- void step (double t)

  steps the simulation a set time
- void write (std::ostream &os) const

  write sim to ostream
- void read (std::istream &is)

  read sim from istream

3.3.1 Detailed Description

wrapper and manager for all simulation entities (currently only particle array)

3.3.2 Member Function Documentation

3.3.2.1 addParticle()

```
void Simulation::addParticle (
             Particle * p)
```

add a particle to simulation

Parameters

p	particle to add

3.3.2.2 getParticles()

```
const Array< Particle > Simulation::getParticles () const  [inline]
```

Get the Particles object (const)

Returns

const Array<Particle>

3.3.2.3 listParticles()

```
void Simulation::listParticles (
             std::ostream & os)
```

list particles to std::ostream

Parameters

os	std::ostream to list

3.3.2.4 operator==()

```
bool Simulation::operator== (
             const Simulation & other) const
```

comparison

Parameters

other	other simulation

3.3.2.5 read()

```
void Simulation::read (
             std::istream & is)
```

read sim from istream

Parameters

is	istream

3.3.2.6 removeParticle()

```
void Simulation::removeParticle (
             size_t idx)
```

remove a particle from simulation

Parameters

idx	index of particle to remove

3.3.2.7 step()

```
void Simulation::step (
             double t)
```

steps the simulation a set time

Parameters

t	time to step

3.3.2.8 write()

```
void Simulation::write (
             std::ostream & os) const
```

write sim to ostream

Parameters

os	ostream

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

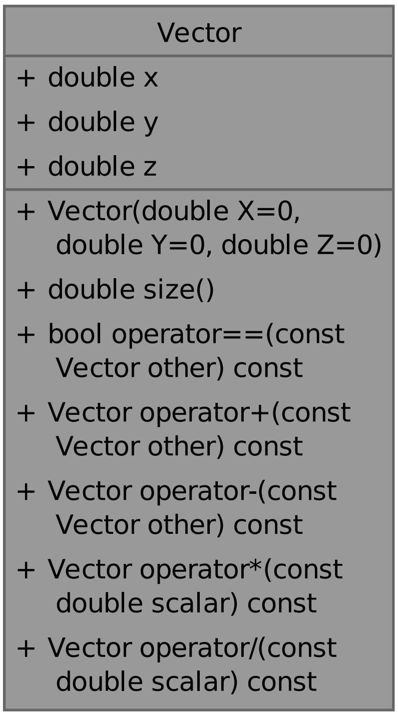
- inc/Simulation.hpp
- src/Simulation.cpp

3.4 Vector Struct Reference

physics type vector3d

```
#include <Vector.h>
```

Collaboration diagram for Vector:


Vector
+ double x
+ double y
+ double z
+ Vector(double X=0, double Y=0, double Z=0)
+ double size()
+ bool operator==(const Vector other) const
+ Vector operator+(const Vector other) const
+ Vector operator-(const Vector other) const
+ Vector operator*(const double scalar) const
+ Vector operator/(const double scalar) const

Public Member Functions

- Vector (double X=0, double Y=0, double Z=0)

  vector constructor
- double size ()

  norm of vector (pythagoras)
- bool operator== (const Vector other) const

  field by field equality
- Vector operator+ (const Vector other) const

  vector addition
- Vector operator- (const Vector other) const

  vector subtraction
- Vector operator∗ (const double scalar) const

  scalar multiplication
- Vector operator/ (const double scalar) const

  scalar division

Public Attributes

- double **x**
- double **y**
- double **z**

3.4.1 Detailed Description

physics type vector3d

3.4.2 Constructor & Destructor Documentation

3.4.2.1 Vector()

```
Vector::Vector (
             double X = 0,
             double Y = 0,
             double Z = 0)
```

vector constructor

Parameters

X	x coordinate
Y	y coordinate
Z	z coordinate

3.4.3 Member Function Documentation

3.4.3.1 operator∗()

```
Vector Vector::operator* (
             const double scalar) const
```

scalar multiplication

Parameters

scalar	scalar to multiply vector by

Returns

Vector scaled vector

3.4.3.2 operator+()

```
Vector Vector::operator+ (
          const Vector other) const
```

vector addition

Parameters

other	vector to sum with

Returns

Vector sum of vectors

3.4.3.3 operator-()

```
Vector Vector::operator- (
          const Vector other) const
```

vector subtraction

Parameters

vector	to subtract

Returns

Vector difference of vectors

3.4.3.4 operator/()

```
Vector Vector::operator/ (
          const double scalar) const
```

scalar division

Parameters

scalar	scalar to divide by

Returns

Vector scaled vector

3.4.3.5 operator==()

```
bool Vector::operator== (
             const Vector other) const
```

field by field equality

Parameters

other	vector to compare

Returns

bool

3.4.3.6 size()

```
double Vector::size ()
```

norm of vector (pythagoras)

Returns

double norm

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

- inc/Vector.h
- src/Vector.cpp

Chapter 4

File Documentation

4.1 Array.hpp

```cpp
#ifndef ARRAY_HPP
#define ARRAY_HPP

#include <cstddef>
#include <iostream>
#include <stdexcept>

#include "memtrace.h"

template<typename T>
class Array {
private:
    T** data;
    size_t size;
    size_t capacity;

    void deepCopy(const Array& other)
    {
        data = new T*[capacity];
        for (size_t i = 0; i < size; ++i)
        {
            data[i] = new T(*other.data[i]);
        }
    }

    void doubleCapacity()
    {
        if (size < capacity) throw std::length_error("Too few elements to double capacity");
        T** temp = new T*[capacity * 2];

        for(size_t i = 0; i < size; i++)
            temp[i] = data[i];

        delete[] data;
        capacity *= 2;
        data = temp;
    }

    void halveCapacity()
    {
        if (size > capacity / 2) throw std::length_error("Too many elements to halve capacity");
        if (capacity <= 2) return;

        T** temp = new T*[capacity / 2];

        for(size_t i = 0; i < size; i++)
            temp[i] = data[i];

        delete[] data;
        data = temp;
    }


public:
    Array(size_t cap = 16)
        : data(nullptr), size(0), capacity(cap)
    {
        data = new T*[capacity];
    }

    Array(const Array& other)
        : data(nullptr), size(other.size), capacity(other.capacity)
    {
        deepCopy(other);
    }

    Array& operator=(const Array& other)
    {
        if (this == &other) return *this;

        for (size_t i = 0; i < size; i++)
            delete data[i];
        delete[] data;

        size = other.size;
        capacity = other.capacity;
        deepCopy(other);

        return *this;
    }

    ~Array()
    {
        for (size_t i = 0; i < size; i++)
            delete data[i];
        delete[] data;
    }

    T& operator[](size_t idx)
    {
        if (idx >= size) throw std::out_of_range("Index out of size");
        return *data[idx];
    }

    const T& operator[](size_t idx) const
    {
        if (idx >= size) throw std::out_of_range("Index out of size");
        return *data[idx];
    }

    bool operator==(const Array<T>& other) const
    {
        if (size != other.size) return false;

        for (size_t i = 0; i < size; i++)
            if (*data[i] != *other.data[i]) return false;

        return true;
    }

    void print(std::ostream& os, const char* sep)
    {
        for (size_t i = 0; i < size-1; i++)
            os « *data[i] « sep;
        os « *data[size-1];
    }

    void insert(T* pelem, size_t idx)
    {
        if (idx > size) throw std::out_of_range("Insert index out of bounds");
        if (size >= capacity)
            doubleCapacity();

        for (size_t i = size; i > idx; i--) {
            *data[i] = *data[i - 1];
        }

        data[idx] = pelem;
        size++;
    }

    void insert(T* pelem)
    {
        insert(pelem, size);
    }

    void remove(size_t idx)
    {
        if (idx >= size) throw std::out_of_range("Remove index out of bounds");
        if (size < capacity/2) halveCapacity();

        delete data[idx];

        for (size_t i = idx; i < size - 1; ++i) {
            data[i] = data[i + 1];
        }

        size--;
    }

    void remove()
    {
        remove(size - 1);
    }

    size_t getSize() const
    {
        return size;
    }

    size_t getCapacity() const
    {
        return capacity;
    }

    void write(std::ostream& os) const
    {
        os « size « std::endl;
        for(size_t i = 0; i < size; i++)
        {
            data[1]->write(os);
            os « std::endl;
        }
    }

    void read(std::istream& is)
    {
        for (size_t i = 0; i < size; i++)
            delete data[i];

        (is » size).ignore(1);
        for(size_t i = 0; i < size; i++)
        {
            data[i] = new T;
            data[i]->read(is);
            is.ignore(1);
        }
    }

};

template<typename T>
std::ostream& operator«(std::ostream& os, const Array<T>& arr)
{
    for (size_t i = 0; i < arr.getSize(); i++)
    {
        os « " index :" « i « std::endl;
        os « arr[i];
    }
    return os;
}

#endif // ARRAY_HPP
```

4.2 constant.h

```
#ifndef CONSTANT_H
#define CONSTANT_H

#define EPSILON 0.0001
#define G 1
#define K 1

#endif
```

4.3 memtrace.h

```
/*********************************
Memoriaszivargas-detektor
Keszitette: Peregi Tamas, BME IIT, 2011
            petamas@iit.bme.hu
Kanari:     Szeberenyi Imre, 2013.,
VS 2012:    Szeberényi Imre, 2015.,
mem_dump:   2016.
inclue-ok:  2017., 2018., 2019., 2021., 2022.
clang-mágia:Bodor András, 2025
*********************************/

#ifndef MEMTRACE_H
#define MEMTRACE_H

#if defined(MEMTRACE)

/*ha definiálva van, akkor a hibakat ebbe a fajlba írja, egyébkent stderr-re*/
/*#define MEMTRACE_ERRFILE MEMTRACE.ERR*/

/*ha definialva van, akkor futas kozben lancolt listat epit. Javasolt a hasznalata*/
#define MEMTRACE_TO_MEMORY

/*ha definialva van, akkor futas kozben fajlba irja a foglalasokat*/
/*ekkor nincs ellenorzes, csak naplozas*/
/*#define MEMTRACE_TO_FILE*/

/*ha definialva van, akkor a megallaskor automatikus riport keszul */
#define MEMTRACE_AUTO

/*ha definialva van, akkor malloc()/calloc()/realloc()/free() kovetve lesz*/
#define MEMTRACE_C

#ifdef MEMTRACE_C
    /*ha definialva van, akkor free(NULL) nem okoz hibat*/
    #define ALLOW_FREE_NULL
#endif

#ifdef __cplusplus
    /*ha definialva van, akkor new/delete/new[]/delete[] kovetve lesz*/
    #define MEMTRACE_CPP
#endif

#if defined(__cplusplus) && defined(MEMTRACE_TO_MEMORY)
    /*ha definialva van, akkor atexit helyett objektumot hasznal*/
    /*ajanlott bekapcsolni*/
    #define USE_ATEXIT_OBJECT
#endif

/******************************************/
/* INNEN NE MODOSITSD                     */
/******************************************/
#ifdef NO_MEMTRACE_TO_FILE
    #undef MEMTRACE_TO_FILE
#endif

#ifdef NO_MEMTRACE_TO_MEMORY
    #undef MEMTRACE_TO_MEMORY
#endif

#ifndef MEMTRACE_AUTO
    #undef USE_ATEXIT_OBJECT
#endif

#ifdef __cplusplus
    #define START_NAMESPACE namespace memtrace {
    #define END_NAMESPACE } /*namespace*/
    #define TRACEC(func) memtrace::func
    #include <new>
#else
    #define START_NAMESPACE
    #define END_NAMESPACE
    #define TRACEC(func) func
#endif

// THROW deklaráció változatai
#if defined(_MSC_VER)
  // VS rosszul kezeli az __cplusplus makrot
  #if _MSC_VER < 1900
    // * nem biztos, hogy jó így *
    #define THROW_BADALLOC
    #define THROW_NOTHING
  #else
    // C++11 vagy újabb
    #define THROW_BADALLOC noexcept(false)
    #define THROW_NOTHING noexcept
  #endif
#else
  #if __cplusplus < 201103L
    // C++2003 vagy régebbi
    #define THROW_BADALLOC throw (std::bad_alloc)
    #define THROW_NOTHING throw ()
  #else
```

```
    // C++11 vagy újabb
    #define THROW_BADALLOC noexcept(false)
    #define THROW_NOTHING noexcept
  #endif
#endif

START_NAMESPACE
    int allocated_blocks();
END_NAMESPACE

#if defined(MEMTRACE_TO_MEMORY)
START_NAMESPACE
    int mem_check(void);
    int poi_check(void*);
END_NAMESPACE
#endif

#if defined(MEMTRACE_TO_MEMORY) && defined(USE_ATEXIT_OBJECT)
#include <cstdio>
START_NAMESPACE
    class atexit_class {
        private:
            static int counter;
            static int err;
        public:
            atexit_class() {
#if defined(CPORTA) && !defined(CPORTA_NOSETBUF)
                if (counter == 0) {
                        setbuf(stdout, 0);
                        setbuf(stderr, 0);
                }
#endif
            counter++;
            }

            int check() {
                if(--counter == 0)
                    err = mem_check();
                return err;
            }

            ~atexit_class() {
                check();
            }
    };

static atexit_class atexit_obj;

END_NAMESPACE
#endif/*MEMTRACE_TO_MEMORY && USE_ATEXIT_OBJECT*/

/*Innentol csak a "normal" include eseten kell, kulonben osszezavarja a mukodest*/
#ifndef FROM_MEMTRACE_CPP
#include <stdlib.h>
#ifdef __cplusplus
    #include <iostream>
/* ide gyűjtjük a nemtrace-vel összeakadó headereket, hogy előbb legyenek */

    #include <fstream>  // VS 2013 headerjében van deleted definició
    #include <sstream>
    #include <vector>
    #include <list>
    #include <map>
    #include <algorithm>
    #include <functional>
    #include <memory>
    #include <iomanip>
    #include <locale>
    #include <typeinfo>
    #include <ostream>
    #include <stdexcept>
    #include <ctime>
    #include <random>
    #if __cplusplus >= 201103L
        #include <iterator>
        #include <regex>
    #endif
#endif
#ifdef MEMTRACE_CPP
    namespace std {
        typedef void (*new_handler)();
}
#endif

#ifdef MEMTRACE_C
START_NAMESPACE
    #undef malloc
    #define malloc(size) TRACEC(traced_malloc)(size,#size,__LINE__,__FILE__)
    void * traced_malloc(size_t size, const char *size_txt, int line, const char * file);

    #undef calloc
    #define calloc(count,size) TRACEC(traced_calloc)(count, size, #count","#size,__LINE__,__FILE__)
    void * traced_calloc(size_t count, size_t size, const char *size_txt, int line, const char * file);

    #undef free
    #define free(p) TRACEC(traced_free)(p, #p,__LINE__,__FILE__)
    void traced_free(void * p, const char *size_txt, int line, const char * file);

    #undef realloc
    #define realloc(old,size) TRACEC(traced_realloc)(old,size,#size,__LINE__,__FILE__)
    void * traced_realloc(void * old, size_t size, const char *size_txt, int line, const char * file);

    void mem_dump(void const *mem, size_t size, FILE* fp = stdout);

END_NAMESPACE
#endif/*MEMTRACE_C*/

#ifdef MEMTRACE_CPP
START_NAMESPACE
    #undef set_new_handler
    #define set_new_handler(f) TRACEC(_set_new_handler)(f)
    void _set_new_handler(std::new_handler h);

    void set_delete_call(int line, const char * file);
END_NAMESPACE

void * operator new(size_t size, int line, const char * file) THROW_BADALLOC;
void * operator new[](size_t size, int line, const char * file) THROW_BADALLOC;
void * operator new(size_t size) THROW_BADALLOC;
void * operator new[](size_t size) THROW_BADALLOC;
void operator delete(void * p)  THROW_NOTHING;
void operator delete[](void * p) THROW_NOTHING;

#if __cplusplus >= 201402L
// sized delete miatt: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3536.html
void operator delete(void * p, size_t)  THROW_NOTHING;
void operator delete[](void * p, size_t) THROW_NOTHING;
#endif

/* Visual C++ 2012 miatt kell, mert háklis, hogy nincs megfelelő delete, bár senki sem használja */
void operator delete(void *p, int, const char *) THROW_NOTHING;
void operator delete[](void *p, int, const char *) THROW_NOTHING;

// clang >= 3.1 esetén vannak warningok, amiket zavar, hogy redefiniálva van a new/delete
#if defined(__clang__) && (__clang_major__ > 3 || \
                           (__clang_major__ == 3 && __clang_minor__ > 0))
   // Csak nagyon drasztikus warning szint mellet jön elő, amikor van rekurzívnak tűnő makró.
   // Ilyenek a new és delete alább, hiszen olyan, mintha magukat hívnák, pedig nincs (ilyen)
   // rekurzió makró szinten.
#  pragma clang diagnostic ignored "-Wdisabled-macro-expansion"
   // Bármilyen kulcsszó újradefiniálása esetén pánikol.
#  pragma clang diagnostic ignored "-Wkeyword-macro"
#endif

#define new new(__LINE__, __FILE__)
#define delete memtrace::set_delete_call(__LINE__, __FILE__),delete

#ifdef CPORTA
#define system(...)  // system(__VA_ARGS__)
#endif

#endif /*MEMTRACE_CPP*/

#endif /*FROM_MEMTRACE_CPP*/
#else
#pragma message ( "MEMTRACE NOT DEFINED" )
#endif /*MEMTRACE*/

#endif /*MEMTRACE_H*/
```

4.4 Particle.hpp

```
#ifndef PARTICLE_HPP
#define PARTICLE_HPP

#include "Vector.h"
#include <ostream>

class Particle
```

```
{
    Vector pos;
    Vector vel;
    double mass;
    double charge;
    bool isGravityInteracting;

public:

    Particle(Vector position = Vector(0,0,0),
             Vector velocity = Vector(0,0,0),
             double m = 1,
             double ch = 0,
             bool grav = false);

    Particle(const Particle&);

    virtual ~Particle();

    Particle& operator=(const Particle&);

    double getMass() const { return mass; }

    double getCharge() const { return charge; }

    bool getGrav() const { return isGravityInteracting; }

    Vector getPos() const { return pos; }

    Vector getVel() const { return vel; }

    bool operator==(const Particle& other) const;

    bool operator!=(const Particle& other) const { return !operator==(other);}

    Vector forceWith(const Particle& other) const;

    void applyForce(Vector force, double time);

    void move(double time);

    void write(std::ostream& os) const;

    void read(std::istream& is);

};

std::ostream& operator«(std::ostream& os, const Particle& p);

#endif
```

4.5 Simulation.hpp

```
#ifndef SIMULATION_HPP
#define SIMULATION_HPP

#include "Array.hpp"
#include "Particle.hpp"

class Simulation {
    Array<Particle> particles;
public:

    Simulation();

    const Array<Particle> getParticles() const { return particles; }

    bool operator==(const Simulation& other) const;

    void listParticles(std::ostream& os);

    void addParticle(Particle* p);

    void removeParticle(size_t idx);

    void step(double t);

    void write(std::ostream& os) const;

    void read(std::istream& is);

};

```

```
std::ostream& operator«(std::ostream& os, const Simulation& sim);


#endif
```

4.6 Vector.h

```
#ifndef VECTOR_H
#define VECTOR_H

#include <iostream>


struct Vector {
    double x,y,z;

    Vector(double X = 0, double Y = 0, double Z = 0);

    double size();

    bool operator==(const Vector other) const;

    Vector operator+(const Vector other) const;

    Vector operator-(const Vector other) const;

    Vector operator*(const double scalar) const;

    Vector operator/(const double scalar) const;
};

//gtest
std::ostream& operator«(std::ostream& os, Vector v);


#endif // VECTOR_H
```

Index