libpedsim

2.2

Generated by Doxygen 1.7.3

Wed May 23 2012 00:22:23

Contents

1 Todo List

Member Ped::Tagent::addWaypoint(Twaypoint ∗wp) Add a flag to change the waypoint queue behavior of the Tagents.

Member Ped::Tagent::desiredForce() move this destination handling into a separate method called by move(). then mark this method as const

Member Ped::Tagent::getFollow() const Add a method that returns a Tagent∗

Member Ped::Tagent::move(double h) Make momentum factor (0.75) settable by the user

Member Ped::Tagent::setFollow(int id) Add a method that takes a Tagent∗ as argument

Member Ped::Tobstacle::rotate(double x, double y, double phi) Use the original points (saved) and cache the total phi or something.

Member Ped::Tscene::Tscene(double left, double up, double width, double height) Get rid of that limitation. A dynamical outer boundary algorithm would be nice.

Member Ped::Ttree::getAgents() const This might be not very efficient, since all childs are checked, too. And then the set (set of pointer, but still) is being copied around.

2 Class Index

2.1 Class List

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

3 Class Documentation

3.1 Ped::Tagent Class Reference

```
#include <ped_agent.h>
```

Public Member Functions

- Tagent ()
- virtual ~Tagent ()
- virtual void move (double h)
- virtual Tvector socialForce () const
- virtual Tvector obstacleForce () const
- virtual Tvector desiredForce ()
- virtual Tvector lookaheadForce (Tvector desired) const
- virtual Tvector myForce (Tvector desired) const
- virtual void **print** () const
- void setPosition (double px, double py, double pz)
- void **setType** (int t)
- void setFollow (int id)
- void setVmax (double vmax)
- int getFollow () const
- int **getid** () const
- int **gettype** () const
- const Tvector & **getPosition** () const
- const Tvector & **getVelocity** () const
- const Tvector & **getAcceleration** () const
- double **getx** () const
- double **gety** () const
- double **getz** () const
- double **getax** () const
- double **getay** () const
- double **getaz** () const
- double **getvx** () const
- double **getvy** () const
- double **getvz** () const
- void setfactorsocialforce (double f)
- void setfactorobstacleforce (double f)
- void setfactordesiredforce (double f)
- void setfactorlookaheadforce (double f)
- void addWaypoint (Twaypoint ∗wp)
- void assignScene (Tscene ∗s)

3.1.1 Detailed Description

This is the main class of the library. It contains the Tagent, which eventually will move through the Tscene and interact with Tobstacle and other Tagent. You can use it as it is, and access the agent's coordinates using the getx() etc methods. Or, if you want to change the way the agent behaves, you can derive a new class from it, and overwrite the methods you want to change. This is also a convenient way to get access to internal variables not available though public methods, like the individual forces that affect the agent.

Author

chgloor

Date

2003-12-26

Examples:

example.cpp.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 Ped::Tagent::Tagent ( )

Default Constructor

Date

2003-12-29

3.1.2.2 Ped::Tagent::∼Tagent ( ) `[virtual]`

Default destructor

Date

2012-02-04

3.1.3 Member Function Documentation

3.1.3.1 void Ped::Tagent::addWaypoint (Twaypoint ∗ *wp*)

Adds a TWaypoint to an agent's list of waypoints. Twaypoints are stored in a cyclic queue, the one just visited is pushed to the back again. There will be a flag to change this behavior soon.

Todo

Add a flag to change the waypoint queue behavior of the Tagents.

Author

chgloor

Date

2012-01-19

Examples:

example.cpp.

3.1.3.2 void Ped::Tagent::assignScene (Ped::Tscene $*$ *s*)

Assigns a Tscene to the agent. Tagent uses this to iterate over all obstacles and other agents in a scene. The scene will invoke this function when Tscene::addAgent() is called.

Date

2012-01-17

Warning

Bad things will happen if the agent is not assigned to a scene. But usually, Tscene takes care of that.

Parameters

$*s$	A valid Tscene initialized earlier.

Referenced by Ped::Tscene::addAgent().

3.1.3.3 Ped::Tvector Ped::Tagent::desiredForce ( ) `[virtual]`

Calculates the force between this agent and the next assigned waypoint. If the waypoint has been reached, the next waypoint in the list will be selected. At the moment, a visited waypoint is pushed back to the end of the list, which means that the agents will visit all the waypoints over and over again. In a later release, this behavior can be controlled by a flag.

Date

2012-01-17

Todo

move this destination handling into a separate method called by move(). then mark this method as const

Returns

Tvector: the calculated force

3.1.3.4 int Ped::Tagent::getFollow ( ) const

Gets the ID of the agent this agent is following.

Date

2012-01-18

Returns

int, the agent id of the agent

Todo

Add a method that returns a Tagent*

3.1.3.5 Ped::Tvector Ped::Tagent::lookaheadForce (Ped::Tvector *e*) const `[virtual]`

Calculates the mental layer force of the strategy "look ahead". It is implemented here in the physical layer because of performance reasons. It iterates over all Tagents in the Tscene, complexity O(N^2).

Date

2012-01-17

Returns

Tvector: the calculated force

Parameters

e	is a vector defining the direction in which the agent should look ahead to. Usually, this is the direction he wants to walk to.

3.1.3.6 void Ped::Tagent::move (double *h*) `[virtual]`

Does the agent dynamics stuff. Calls the methods to calculate the individual forces, adds them to get the total force aggecting the agent. This will then be translated into a velocity difference, which is applied to the agents velocity, and then to its position.

Date

2003-12-29

Parameters

h	This tells the simulation how far the agent should proceed (also known as Tau in literature). 1 = 1 unit.

Todo

Make momentum factor (0.75) settable by the user

Note

Is the momentum factor (0.75) dependent of h?? think so --chgloor 2012-01-15

Referenced by Ped::Tscene::moveAgents().

3.1.3.7 Ped::Tvector Ped::Tagent::myForce (Ped::Tvector *e*) const `[virtual]`

myForce() is a method that returns an "empty" force (all components set to 0). This method can be overridden in order to define own forces. It is called in move() in addition to the other default forces.

Date

2012-02-12

Returns

Tvector: the calculated force

Parameters

e	is a vector defining the direction in which the agent wants to walk to.

3.1.3.8 Ped::Tvector Ped::Tagent::obstacleForce ( ) const `[virtual]`

Calculates the force between this agent and the nearest obstacle in this scene. Iterates over all obstacles == O(N).

Date

2012-01-17

Returns

Tvector: the calculated force

3.1.3.9 void Ped::Tagent::setfactordesiredforce (double *f*)

Sets the factor by which the desired force is multiplied. Values between 0 and about 10 do make sense.

Date

2012-01-20

Parameters

f	The factor

3.1.3.10 void Ped::Tagent::setfactorlookaheadforce (double *f*)

Sets the factor by which the look ahead force is multiplied. Values between 0 and about 10 do make sense.

Date

2012-01-20

Parameters

f	The factor

3.1.3.11 void Ped::Tagent::setfactorobstacleforce (double *f*)

Sets the factor by which the obstacle force is multiplied. Values between 0 and about 10 do make sense.

Date

2012-01-20

Parameters

f	The factor

3.1.3.12 void Ped::Tagent::setfactorsocialforce (double *f*)

Sets the factor by which the social force is multiplied. Values between 0 and about 10 do make sense.

Date

2012-01-20

Parameters

f	The factor

3.1.3.13 void Ped::Tagent::setFollow (int *id*)

Sets the agent ID this agent has to follow. If set, the agent will ignore its assigned waypoints and just follow the other agent.

Date

2012-01-08

Parameters

id	is the agent to follow (must exist, obviously)

Todo

Add a method that takes a Tagent∗ as argument

3.1.3.14 void Ped::Tagent::setPosition (double *px*, double *py*, double *pz*)

Sets the agent's position. This, and other getters returning coordinates, will eventually changed to returning a Tvector.

Date

2004-02-10

Parameters

px	Position x
py	Position y
pz	Position z

Examples:

example.cpp.

3.1.3.15 void Ped::Tagent::setVmax (double *pvmax*)

Sets the maximum velocity of an agent (vmax). Even if pushed by other agents, it will not move faster than this.

Date

2012-01-08

Parameters

pvmax	The maximum velocity. In scene units per timestep, multiplied by the simulation's precision h.

3.1.3.16 Ped::Tvector Ped::Tagent::socialForce ( ) const [virtual]

Calculates the social force between this agent and all the other agents belonging to the same scene. It iterates over all agents inside the scene, has therefore the complexity O(N^2). A better agent storing structure in Tscene would fix this. But for small (less than 10000 agents) scenarios, this is just fine.

Date

2012-01-17

Returns

Tvector: the calculated force

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

- ped_agent.h

- ped_agent.cpp

3.2 Ped::Tobstacle Class Reference

```
#include <ped_obstacle.h>
```

Public Member Functions

- Tobstacle ()
- Tobstacle (double ax, double ay, double bx, double by)
- virtual void setPosition (double ax, double ay, double bx, double by)
- virtual Tvector obstacleforce (double p1, double p2)
- virtual void rotate (double x, double y, double phi)
- void **setType** (int t)
- int **getid** ()
- int **gettype** ()
- double **getax** ()
- double **getay** ()
- double **getbx** ()
- double **getby** ()

3.2.1 Detailed Description

Class that defines a Tobstacle object. An obstacle is, for now, always a wall with start and end coordinate.

Author

chgloor

Date

2012-01-17

Examples:

example.cpp.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 Ped::Tobstacle::Tobstacle ( )

Default constructor, places a wall from 0/0 to 1/1

Date

2012-01-07

3.2.2.2 Ped::Tobstacle::Tobstacle (double *pax*, double *pay*, double *pbx*, double *pby*)

Constructor used to set intial values.

Date

2012-01-07

Parameters

pax	x coordinate of the first corner of the obstacle.
pay	y coordinate of the first corner of the obstacle.
pbx	x coordinate of the second corner of the obstacle.
pby	y coordinate of the second corner of the obstacle.

3.2.3 Member Function Documentation

3.2.3.1 Ped::Tvector Ped::Tobstacle::obstacleforce (double *x*, double *y*) `[virtual]`

Calculates and returns the forces of the obstacle to a given point x/y. x/y can be the location of an agent, but it can also be anything else, for example a grid coordinate of the user interface, if you want to display the obstacle forces on the map.

Date

2012-01-17

Returns

Tvector forces

Parameters

double	x: The x coordinate of the point
double	y: The y coordinate of the point

3.2.3.2 void Ped::Tobstacle::rotate (double *x*, double *y*, double *phi*) `[virtual]`

rot phi around x/y

Author

chgloor

Date

2012-01-20

Warning

Due to rounding errors, this will fail after a while.

Todo

Use the original points (saved) and cache the total phi or something.

Parameters

x	The x coordinate of the point the obstacle will be rotated around.
y	The y coordinate of the point the obstacle will be rotated around.
r	The angle the obstacle will be rotated, where phi is given in radians

3.2.3.3 void Ped::Tobstacle::setPosition (double *pax,* double *pay,* double *pbx,* double *pby*) `[virtual]`

Moves the obstacle to a new position. Can be uses to simulate opening doors etc.

Date

2012-01-07

Parameters

pax	x coordinate of the first corner of the obstacle.
pay	y coordinate of the first corner of the obstacle.
pbx	x coordinate of the second corner of the obstacle.
pby	y coordinate of the second corner of the obstacle.

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

- ped_obstacle.h
- ped_obstacle.cpp

3.3 Ped::Tscene Class Reference

```
#include <ped_scene.h>
```

Public Member Functions

- Tscene ()
- Tscene (double left, double up, double width, double height)
- virtual ∼Tscene ()
- virtual void addAgent (Tagent ∗a)
- virtual void addObstacle (Tobstacle ∗o)
- virtual void cleanup ()
- virtual void moveAgents (double h)

- set< const Ped::Tagent ∗ > getNeighbors (double x, double y, double dist) const
- const vector< Tagent ∗ > & **getAllAgents** () const

Protected Attributes

- Ttree ∗ **tree**

Friends

- class Ped::Tagent
- class Ped::Ttree

3.3.1 Detailed Description

The Tscene class contains the spatial representation of the "world" the agents live in. Theoretically, in a continuous model, there are no boundaries to the size of the world. Agents know their position (the x/y co-ordinates). However, to find the nearest neighbors of an agent, it makes sense to put them in some kind of "boxes". In this implementation, the infinite world is divided by a dynamic quadtree structure. There are some CPU cycles required to update the structure with each agent position change. But the gain in looking up the neighbors is worth this. The quadtree structure only needs to be changed when an agent leaves its box, which migh only happen every 100th or 1000th timestep, depending on the box size. The Tscene class needs an outer boundary in order to construct the initial box of the quadtree. Agents are not allowd to go outside that boundary. If you do not know how far they will walk, choose a rather big boundary box. The quadtree algorythm will dynamically assign smaller sub-boxes within if required. If all (most) agents walk out of a box, it is no longer needed. It can be colleted. If there are some agents left, they will be assigned to the box above in the hierarchy. You must trigger this collection process periodically by calling cleanup() manually

Author

chgloor

Date

2010-02-12

Examples:

example.cpp.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 Ped::Tscene::Tscene ( )

Default constructor. If this constructor is used, there will be no quadtree created. This is faster for small scenarios or less than 1000 Tagents.

Date

2012-01-17

3.3.2.2 Ped::Tscene::Tscene (double *left,* double *up,* double *width,* double *height*)

Constructor used to create a quadtree statial representation of the Tagents. Use this constructor when you have a sparsely populated world with many agents (>1000). The agents must not be outside the boundaries given here. If in doubt, use an initial boundary that is way to big.

Todo

Get rid of that limitation. A dynamical outer boundary algorithm would be nice.

Date

2012-01-17

Parameters

left	is the left side of the boundary
up	is the upper side of the boundary
width	is the total width of the boundary. Basically from left to right.
height	is the total height of the boundary. Basically from up to down.

3.3.2.3 Ped::Tscene::∼Tscene ( ) `[virtual]`

Destructor

Date

2012-02-04

3.3.3 Member Function Documentation

3.3.3.1 void Ped::Tscene::addAgent (Ped::Tagent ∗ *a*) `[virtual]`

Used to add a Tagent to the Tscene.

Date

2012-01-17

Warning

addAgent() does call Tagent::assignScene() to assign itself to the agent.

Parameters

∗a	A pointer to the Tagent to add.

Note

The Tagents∗ given to addAgent() are not const (i.e. not const Tagent∗) because of moveAgents(double h). It obviously modifies the agents. Agents added to the Scene are not deleted if the Scene is destroyed. The reason for this is because they could be member of another Scene theoretically.

Examples:

example.cpp.

3.3.3.2 void Ped::Tscene::addObstacle (Ped::Tobstacle ∗ *o*) `[virtual]`

Used to add a Tobstacle to the Tscene.

Date

2012-01-17

Parameters

∗o	A pointer to the Tobstacle to add.

Note

Obstacles added to the Scene are not deleted if the Scene is destroyed. The reason for this is because they could be member of another Scene theoretically.

Examples:

example.cpp.

3.3.3.3 void Ped::Tscene::cleanup ( ) `[virtual]`

This triggers a cleanup of the tree structure. Unused leaf nodes are collected in order to save memory. Ideally cleanup() is called every second, or about every 20 timestep.

Date

2012-01-28

3.3.3.4 set< const Ped::Tagent ∗ > Ped::Tscene::getNeighbors (double *x*, double *y*, double *dist*) const

Returns the list of neighbors within dist of the point x/y. This can be the position of an agent, but it is not limited to this.

Date

2012-01-29

Returns

The list of neighbors

Parameters

x	the x coordinate
y	the y coordinate
dist	the distance around x/y that will be saerched for agents (search field is a square in the current implementation)

3.3.3.5 void Ped::Tscene::moveAgents (double *h*) `[virtual]`

This is a convenience method. It calls Ped::Tagent::move(double h) for all agents in the Tscene.

Date

2012-02-03

Parameters

h	This tells the simulation how far the agents should proceed.

See also

Ped::Tagent::move(double h)

Examples:

example.cpp.

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

- ped_scene.h
- ped_scene.cpp

3.4 Ped::Ttree Class Reference

Public Member Functions

- Ttree (Ped::Tscene ∗scene, int depth, double x, double y, double w, double h)
- virtual ∼Ttree ()
- virtual void addAgent (const Ped::Tagent ∗a)
- virtual void moveAgent (const Ped::Tagent ∗a)
- virtual set< const Ped::Tagent ∗ > getAgents () const

- virtual bool intersects (double px, double py, double pr) const
- double **getx** () const
- double **gety** () const
- double **getw** () const
- double **geth** () const
- double **getdepth** () const

Protected Member Functions

- virtual int cut ()
- virtual void addChildren ()

Protected Attributes

- Ttree * **tree1**
- Ttree * **tree2**
- Ttree * **tree3**
- Ttree * **tree4**

Friends

- class **Tscene**

3.4.1 Constructor & Destructor Documentation

3.4.1.1 Ped::Ttree::Ttree (Ped::Tscene * *pscene*, int *pdepth*, double *px*, double *py*, double *pw*, double *ph*)

Description: set intial values

Author

chgloor

Date

2012-01-28

Referenced by addChildren().

3.4.1.2 Ped::Ttree::∼Ttree ( ) `[virtual]`

Destructor. Deleted this node and all its children. If there are any agents left, they are removed first (not deleted).

Author

chgloor

Date

2012-01-28

3.4.2 Member Function Documentation

3.4.2.1 void Ped::Ttree::addAgent (const Ped::Tagent $*$ *a*) `[virtual]`

Adds an agent to the tree. Searches the right node and adds the agent there. If there are too many agents at that node allready, a new child is created.

Author

chgloor

Date

2012-01-28

Parameters

$*a$	The agent to add

3.4.2.2 void Ped::Ttree::addChildren ( ) `[protected, virtual]`

A little helper that adds child nodes to this node

Author

chgloor

Date

2012-01-28

Referenced by addAgent().

3.4.2.3 int Ped::Ttree::cut ( ) `[protected, virtual]`

Checks if this tree node has not enough agents in it to justify more child nodes. It does this by checking all child nodes, too, recursively. If there are not enough children, it moves all the agents into this node, and deletes the child nodes.

Author

chgloor

Date

2012-01-28

Returns

the number of agents in this and all child nodes.

3.4.2.4 set< const Ped::Tagent $*$ > Ped::Ttree::getAgents ( ) const `[virtual]`

Returns the set of agents that is stored within this tree node

Author

chgloor

Date

2012-01-28

Returns

The set of agents

Todo

This might be not very efficient, since all childs are checked, too. And then the set (set of pointer, but still) is being copied around.

Referenced by Ped::Tscene::getNeighbors().

3.4.2.5 bool Ped::Ttree::intersects (double *px*, double *py*, double *pr*) const `[virtual]`

Checks if a point x/y is within the space handled by the tree node, or within a given radius r

Author

chgloor

Date

2012-01-29

Returns

true if the point is within the space

Parameters

px	The x co-ordinate of the point
py	The y co-ordinate of the point
pr	The radius

Referenced by Ped::Tscene::getNeighbors().

3.4.2.6 void Ped::Ttree::moveAgent (const Ped::Tagent ∗ *a*) [virtual]

Updates the tree structure if an agent moves. Removes the agent and places it again, if outside boundary. If an this happens, this is O(log n), but O(1) otherwise.

Author

chgloor

Date

2012-01-28

Parameters

∗*a*	the agent to update

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

- ped_tree.h
- ped_tree.cpp

3.5 Ped::Tvector Class Reference

```
#include <ped_vector.h>
```

Public Member Functions

- Tvector ()
- **Tvector** (double px, double py, double pz)
- **Tvector** (const Tvector &source)
- Tvector & **operator=** (const Tvector &source)
- virtual ∼Tvector ()
- void cross (const Tvector &a, const Tvector &b)
- void normalize ()

Public Attributes

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

Friends

- double scalar (const Tvector &a, const Tvector &b)

3.5.1 Detailed Description

Vector helper class. This is basically a struct with some related functions attached. x, y, and z are public, so that they can be accessed easily.

Author

chgloor

Date

2010-02-12

3.5.2 Constructor & Destructor Documentation

3.5.2.1 Ped::Tvector::Tvector ( )

Default constructor, which makes sure that all the values are set to 0.

Date

2012-01-16

3.5.2.2 Ped::Tvector::∼Tvector ( ) `[virtual]`

Default destructor

Date

2012-05-05

3.5.3 Member Function Documentation

3.5.3.1 void Ped::Tvector::cross (const Tvector & *a*, const Tvector & *b*)

Vector cross product helper: calculates the cross product of two vectors.

Date

2010-02-12

Warning

The result is assigned to the vector calling the method.

Parameters

&a	The first vector
&b	The second vector

3.5.3.2 void Ped::Tvector::normalize ( )

Normalizes the vector to a length of 1.

Date

2010-02-12

3.5.4 Friends And Related Function Documentation

3.5.4.1 double scalar (const Tvector & *a*, const Tvector & *b*) `[friend]`

Vector scalar product helper: calculates the scalar product of two vectors.

Date

2012-01-14

Returns

The scalar product.

Parameters

&a	The first vector
&b	The second vector

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

- ped_vector.h
- ped_vector.cpp

3.6 Ped::Twaypoint Class Reference

`#include <ped_waypoint.h>`

Public Member Functions

- Twaypoint ()
- Twaypoint (double x, double y, double r)
- virtual ∼Twaypoint ()
- virtual Tvector getForce (double myx, double myy, double fromx, double fromy, bool ∗reached) const
- virtual Tvector normalpoint (double p1, double p2, double oc11, double oc12, double oc21, double oc22) const
- void **setType** (int t)
- void **setx** (double px)
- void **sety** (double py)

- void **setr** (double pr)
- void **settype** (int t)
- int **getid** () const
- int **gettype** () const
- double **getx** () const
- double **gety** () const
- double **getr** () const

Static Public Attributes

- static const int **TYPE_NORMAL** = 0
- static const int **TYPE_POINT** = 1

3.6.1 Detailed Description

The waypoint classs

Author

chgloor

Date

2012-01-07

Examples:

example.cpp.

3.6.2 Constructor & Destructor Documentation

3.6.2.1 Ped::Twaypoint::Twaypoint ( )

Constructor - sets the most basic parameters.

Date

2012-01-07

3.6.2.2 Ped::Twaypoint::Twaypoint (double *px*, double *py*, double *pr*)

Constructor: Sets some intial values. The agent has to pass within the given radius.

Date

2012-01-07

Parameters

px	The x coordinate of the waypoint
py	The y coordinate of the waypoint
pr	The radius of the waypoint

3.6.2.3 Ped::Twaypoint::∼Twaypoint ( ) `[virtual]`

Default Destructor

Author

chgloor

Date

2012-02-04

3.6.3 Member Function Documentation

3.6.3.1 Ped::Tvector Ped::Twaypoint::getForce (double *myx*, double *myy*, double *fromx*, double *fromy*, bool ∗ *reached*) const `[virtual]`

Returns the force into the direction of the waypoint

Date

2012-01-10

Parameters

myx	The x coordinate of the current position of the agent
myy	The y coordinate of the current position of the agent
fromx	The x coordinate of the last assigned waypoint, i.e. where the agent is coming from
fromy	The y coordinate of the last assigned waypoint, i.e. where the agent is coming from
∗*reached*	Set to true if the agent has reached the waypoint in this call.

Returns

Tvector The calculated force

3.6.3.2 Ped::Tvector Ped::Twaypoint::normalpoint (double *p1*, double *p2*, double *oc11*, double *oc12*, double *oc21*, double *oc22*) const `[virtual]`

Calculates the point that is on the given line and normal to the given position. If it is not inside the line, the start or end point of the line is returned.

Date

2012-01-10

Parameters

p1	The x coordinate of the point outside the obstacle
p2	The y coordinate of the point outside the obstacle
oc11	The x coordinate of the first corner of the obstacle
oc12	The y coordinate of the first corner of the obstacle
oc21	The x coordinate of the second corner of the obstacle
oc22	The y coordinate of the second corner of the obstacle

Returns

Tvector The calculated point

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

- ped_waypoint.h
- ped_waypoint.cpp

4 Example Documentation

4.1 example.cpp

```
//
// pedsim - A microscopic pedestrian simulation system.
// Copyright (c) 2003 - 2012 by Christian Gloor
//
// Use somethin like this to compile:
// g++ examples/example.cpp -o example -I. -lpedsim -L. -g
//
// Check for memory leaks e.g. like this:
// valgrind --leak-check=yes ./example

#include "ped_includes.h"

#include <iostream>
#include <cstdlib> // rand

using namespace std;

int main(int argc, char *argv[]) {

        cout << "PedSim Example using libpedsim version " << Ped::LIBPEDSIM_VERSI
      ON << endl;

        // setup
        Ped::Tscene *pedscene = new Ped::Tscene(-200, -200, 400, 400);

        Ped::Twaypoint *w1 = new Ped::Twaypoint(-100, 0, 24);
        Ped::Twaypoint *w2 = new Ped::Twaypoint(+100, 0, 12);
        Ped::Tobstacle *o = new Ped::Tobstacle(0, -50,  0, +50);
        pedscene->addObstacle(o);

        for (int i = 0; i<100; i++) {
          Ped::Tagent *a = new Ped::Tagent();

          a->addWaypoint(w1);
          a->addWaypoint(w2);

          a->setPosition(-50 + rand()/(RAND_MAX/80)-40, 0 + rand()/(RAND_MAX/20)
      -10, 0);

          pedscene->addAgent(a);
        }

        // move all agents for 10 steps (and print their position)
   for (int i=0; i<10; ++i) {
                pedscene->moveAgents(0.2);

                const vector<Ped::Tagent*>& myagents = pedscene->getAllAgents();
                for (vector<Ped::Tagent*>::const_iterator iter = myagents.begin()
      ; iter != myagents.end(); ++iter) {
                        (*iter)->print();
                }
        }

        // cleanup
        const vector<Ped::Tagent*>& myagents = pedscene->getAllAgents();
        for (vector<Ped::Tagent*>::const_iterator iter = myagents.begin(); iter !
      = myagents.end(); ++iter) {
                delete *iter;
        }
        delete pedscene;
        delete w1;
        delete w2;
        delete o;

}
```

Index