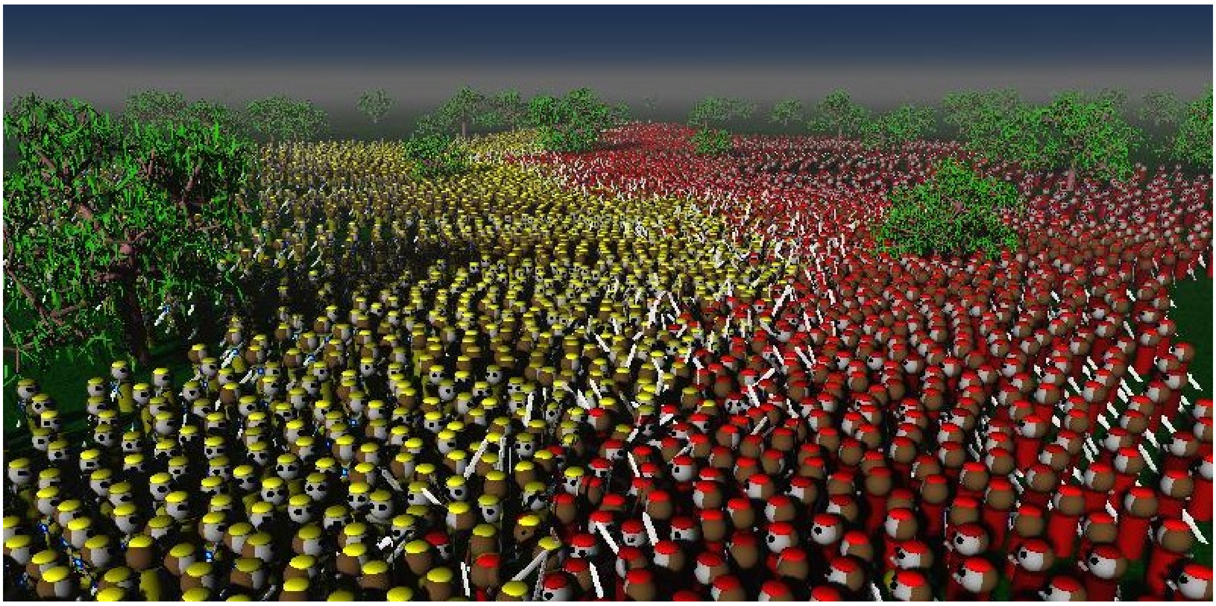
PEDSIM

A Pedestrian Crowd Simulation System



Motivation, Usage, Installation and Library Documentation

http://pedsim.silmaril.org/

BUILDING BETTER WORLDS

Contents

Chapter 1

libpedsim

PEDSIM is a microscopic pedestrian crowd simulation library. It is suitable for use in crowd simulations (e.g. indoor evacuation simulation, large scale outdoor simulations), where one is interested in output like pedestrian density or evacuation time. Also, the quality of the individual agent's trajectory is high enough for creating massive pedestrian crowd animations (e.g. for motion pictures or architectural visualization). Since libpedsim is easy to use and extend, it is a good starting point for science projects. See the `examples` page for example pictures, short movies, and for screenshots.

The PEDSIM library allows you to use pedestrian dynamics in your own software. Based on pure C++ without additional packages, it runs virtually on every operating system. PEDSIM has been developed and tested on Linux. Also supported, but slightly less tested, is Visual Studio on Windows. In the `ecosystem` directory you find additional parts that use or extend the PEDSIM library. They are meant to give a quick overview of the capabilities, and are starting points for your own experiments. Most of them are built using the Qt Framework, which you'll need to download separately.

The pedestrians are visible on the user interface in real-time. Using the file or network-based output, both batch and real-time processing is possible.

To create video sequences the output of PEDSIM is usually fed into a rendering engine, where realistically looking humans are created. These humans walk based on the trajectories generated by PEDSIM.

General Usage Notes

- Create a Ped::Tscene
- Create a Ped::Tobstacle
- Add the Ped::Tobstacle to the scene
- Create a Ped::Tagent
- Create a Ped::Twaypoint
- Add the Ped::Twaypoint to the Ped::Tagent
- Add the Ped::Tagent to the Ped::Tscene
- Call Ped::Tagent->move() for each timestep

See the Code Example further down, and have a look at the full source code, available on the `download` page.

Detailed Class Documentation

There is a complete documentation of the classes in the library. It is automatically generated out of the source code. You can access this documentation online here. This same documentation is delivered as PDF file with the library for offline use.

Code Example

This example shows the very basic usage of the library. No fancy graphics, of course, only text output to the console. This is example01.cpp in the examples folder.

Please note: Additional steps around the example code might be required in order to compile it. If you are using *Windows* and, for example, *Microsoft Visual Studio*, you can create a new console application using the wizard. Create a file called `example01.cpp` and copy-paste the code into it. In the project's *Properties*, under *Linker/Input*, add `libpedsim.lib` in front of the *Additional Dependencies* list. Click run. Also see Using PEDSIM on Windows.

On a typical linux system, if you are in the *libpedsim* folder, use this to compile, link and run:

```
g++ examples/example01.cpp -o example -lpedsim -L. -I. -std=c++11
export LD_LIBRARY_PATH=.
./example
```

It will create 100 agents, which are placed somewhat randomly distributed around -50/0. They should walk between -100/0 and 100/0. An obstacle (wall) is placed from 0/-50 to 0/50. The agents must walk around that obstacle. That's it, as simple as that: the agents will walk with that little code.

If you want to display some graphics, write a file, or send data over the network, you will get the agent's positions with a->getPosition(). Of course, you can inherit your own classes from Tagent, Tobctacle etc, if you want to have more control. See the Demo App Source for an example.

```
// pedsim - A microscopic pedestrian simulation system.
// Copyright (c) by Christian Gloor

#include <iostream>
#include <cstdlib>
#include <chrono>
#include <thread>

#include "ped_includes.h"

#include "ped_outputwriter.h"

using namespace std;

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

    // create an output writer which will send output to a file
    Ped::OutputWriter *ow = new Ped::FileOutputWriter();
    ow->setScenarioName("Example 01");

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

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

    pedscene->setOutputWriter(ow);

    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<10; 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 700 steps (and write their position through the outputwriter)
    for (int i=0; i<700; ++i) {
        pedscene->moveAgents(0.3);
        std::this_thread::sleep_for(std::chrono::milliseconds(3));
    }

    // Cleanup
    for (Ped::Tagent* agent : pedscene->getAllAgents()) delete agent;
    delete pedscene;
    delete w1;
    delete w2;
    delete o;
    delete ow;

    return EXIT_SUCCESS;
}
```

Chapter 2

PedSim Behavior and Background

Here follows a small introduction to the technique used in the code of PEDSIM.

The simulation core takes care of the physical aspects of the system, such as interaction of the agents with the environment or with each other. Typical simulation techniques for such problems are:

- In *microscopic simulations* each particle is represented individually.
- In macroscopic or *field-based simulations*, particles are aggregated into fields. The corresponding mathematical models are partial differential equations, which need to be discretized for computer implementations.
- It is possible to combine microscopic and field-based methods, which is sometimes called **smooth particle hydrodynamics**. In SPH, the individuality of each particle is maintained. During each time step, particles are aggregated to field quantities such as density, then velocities are computed from these densities, and then each individual particle is moved according to these macroscopic velocities.
- As a fourth method, somewhat on the side, exist the queuing simulations from operations research. Here, particles move in a networks of queues, where each queue has a service rate. Once a particle is served, it moves into the next queue.

For PEDSIM, we need to maintain individual particles, since they need to be able to make individual decisions, such as route choices, throughout the simulation. This immediately rules out field-based methods. We also need a realistic representation of inter-pedestrian interactions, which rules out both the queue models and the SPH models.

For microscopic simulations, there are essentially two techniques: methods based on *coupled differential equations*, and *cellular automata (CA)* models. In our situation, it is important that agents can move in arbitrary directions without artifacts caused by the modeling technique, which essentially rules out CA techniques. A generic coupled differential equation model for pedestrian movement is the *social force model* by Helbing et al., see e.g. this paper.

$$m_i \frac{d\mathbf{v}_i}{dt} = m_i \frac{\mathbf{v}_i^0 - \mathbf{v}_i}{\tau_i} + \sum_{j \neq i} \mathbf{f}_{ij} + \sum_W \mathbf{f}_{iW}$$

where *m* is the mass of the pedestrian and *v* its velocity. *v0* is its desired velocity; in consequence, the first term on the RHS models exponential approach to that desired velocity, with a time constant *tau*. The second term on the RHS models pedestrian interaction, and the third models interaction of the pedestrian with the environment. The *social force model* should be considered as an example on how to model the pedestrian interaction. It is easy to understand and simple to implement. However, a future implementation of PEDSIM might use a different model.

Pedestrians interact with each other, which includes avoiding collisions (short range interaction), and attraction to enemies (long range, which represents the "will" of the agents. This attraction to enemies is just an example and should be replaced by some more complicated and meaningful functions). Also avoidance of objects like trees is implemented.

This simulation also works close to obstacles, as are found e.g. close to buildings. Also the simulation of the inside of buildings is possible, which allows the usage of the same framework for e.g. evacuation simulation.

Any mobility simulation system does not just consist of the mobility simulation itself (which controls the physical constraints of the agents in a virtual world), but also of modules that compute higher level strategies of the agents. In fact, it makes sense to consider the physical and the mental world completely separately.

- The **Physical Layer** (the mobility simulation) takes care of the physical aspects of the system, such as movement of the agents, interaction of the agents with the environment, or interactions between the agents.
- The **Mental Layers** implement the humans intelligence (well, at least a part of it), which improves the agent's behavior. Actually, if the mental layer strategy are very sophisticated, there is no need for the *social force model* in the physical simulation - all the forces can be set to zero. The **Look Ahead** mental strategy tells each agent to look for other agents in front of him, an count the ones at the left side and the ones at the right side. It then will walk into the direction where less other agents are. Collisions with walls and other pedestrians are avoided by the pedestrian itself, and not by a constraint by the underlying physical model. Another example for a mental layer module is a **Route Generator**. It is not enough to have agents walk around randomly; for realistic applications it is necessary to generate plausible routes for each pedestrian. Being able to compute routes, as the route generator does, only makes sense if one knows the destinations for the agents. A technique in transportation research is to generate a (say) day-long chain of activities for each agent, and each activity's specific location. There exist very sophisticated mental layer modules. There is for example a **View Analyzer Module**, which describes to the system what individual agents "see" as they move through the landscape. The agents field-of-view is analyzed, and events are sent to the system describing what the agent sees.

Chapter 3

Using PEDSIM on Linux

These are some notes regardng setting up a Linux development system for PEDSIM. If you have written software before, you might have everything needed in place already.

Basically you need a C++ compiler to compile *libpedsim* (the core library of PEDSIM). Graphical bits of the ecosystem folder require *Qt5* - see here https://www.qt.io/developers/. Other compilers might work - make sure they support *C++11*.

C++ Compiler

I personally use *gcc version 5.4.0* on a *Linux Mint* system. *gcc 4.8* probably works just fine.

Preparing a system for software development

You should be root for these steps:

```
sudo -s
```

First install the C++ compiler:

```
aptitude install gcc g++
```

For the graphical bits install *Qt5*:

```
aptitude install qt5-default qt5-qmake qt5-style-plugins
```

If you want to update the PEDSIM documentation, you need *Doxygen* and, if you want pdf output, *LaTeX*:

```
aptitude install doxygen
aptitude install texlive-full
```

Good luck!

Chapter 4

Using PEDSIM on Windows

This document explains how to use `_Microsoft Visual Studio Community 2015_` and `Qt _5.7_` to compile PEDSIM and the examples.

libpedsim

Basically, in *Visual Studio*, create a new project of type *Win32 Console Application*, and click "DLL" in the wizard. Uncheck the "Precompiled Header" and "Security Development Lifecycle (SDL) checks" boxes. This will open a new project containing a few empty documents. Add all `.cpp` Files to the "Source Files" filter, and all `.h` files to the "Header Files" filter. Then remove the files that Windows generated for you, they are not used.

If you click "Build", Visual Studio will generate a `libpedsim.lib` and a `libpedsim.dll` file.

Such a solution file is located in the "msvc15" folder. If you have installed Visual Studio, double click `libpedsim.sln`. Then, once the IDE has opened, use "Build Solution F7" from the menu. This will generate a folder called "x64" (on a 64bit system), with a subfolder called "Debug". In there are the compiled library files. Copy `libpedsim.dll` and `libpedsim.lib` into the `/pedsim/libpedsim/` folder (where the `.cpp` and `.h` files are).

Examples (and your own programs):

Again, create a new *Win32 Console Application* project. This time, do not check he "DLL" box (still uncheck the others). I think the default is "Console Application" here; leave that as it is. Again, remove the auto-generated files, and add e.g. the file `example03.cpp`.

Now here comes the tricky part. You have to specify the include and library directories. Go to "Project Properties Alt+F7". You probably have to select the item below the top item in the menu at the left. The first is the "solution", the second is the "project" which is what we need. Once opened, there is a window with "Configuration Properties". Go to "VC++ Directories". Add the path to the libpedsim source files to the "Include Directories". This is where the `ped_*.cpp`/`.h` files are. Then, do the same with "Library Directories", where you add the path to the libpedsim.lib file generated while compiling libpedsim.

A bit further down should be a tab called "Linker", and "Input". There you have to add `libpedsim.lib` to the list of libraries to include ("Additional Dependencies).

Now you should be able to build the example. Again, one sample project file is in the examples folder. Theoretically, it should be possible to open that using *Visual Studio*, and just click on "Build Solution F7" to compile `example03.exe`.

Once you have generated the `example03.exe` file, you need to copy the `libpedsim.dll` file into that folder (next to the `.exe`). This is because `libpedsim.dll` is not installed in one of the system-wide folders.

2-dimensional visualizer 2dvis

2dvis is built on *Qt*. If you want to use all the features (especially charts), you need *Qt 5.7* or above. It should be possible to compile it using an oder version. However, you will not see the metrics charts in 2dvis then.

Download the latest version of Qt, and install it. I think you only need the msvc15 files, about 3.0 GB. I assume *Visual Studio* has been installed in the steps above. It is possible to use e.g. *Cygwin*, but that is beyond the scope of that short introduction.

Once *Qt* is installed, simply double click the `2dvis.pro` file. This will open *Qt Creator* with the project loaded, where you can build `2dvis.exe`. Before you can start the application, you need to copy various `Qt` libraries into the same directory. Easiest is to just double click the `2dvis.exe` file, and see what happens. These `qt5*d.dll` libraries are somewhere in your *Qt* installation folder, located (probably) under `C:\Qt\...\bin`. Once the required ones are there, start 2dvis using the command line:

Run 2dvis from the command line:

```
2dvis.exe -n 2222
```

This starts a 2dvis that is listening on network port 2222 for incoming data (used by the Ped::UDPOutputWriter class of *libpedsim* in the examples).

Chapter 5

PEDSIM Contributors

PEDSIM was written mostly by Christian Gloor. But there have been a lot of people helping with writing the code and providing feedback! Thanks!

Initial Package

The Multi-Agent Sim Team @ ETH Zürich and TU Berlin:

- Kai Nagel
- Duncan Cavens
- Nurhan Cetin
- Bryan Raney
- Michael Balmer
- David Charypar
- Fabrice Marchal

Notable Contributions

Sven Wehner provided an extremly comprehensive patch fixing many mistakes.

Max Küng has built the initial server-side git repository and the Nginx setup. He also wrote some patches.

Chapter 6

FAQ

PEDSIM Frequently Asked Questions

General

> Is there a PEDSIM application that allows me to design, run and analyze my own scenario?

No. At this point, PEDSIM is a library, plus some helper applications. This means you basically have to write your own application. You need a C++ compiler and some computer science knowledge.

Some helper applications are included, however. They allow you to try simple scenarios without programming anything. You can e.g. define a scenario in XML for the Demo Application (included), and simulate it there. It is also possible to extend one of the C++ examples, which is very straightforward, and visualize/analyze it using the 2-dimensional visualizer 2dvis (included).

> Is PEDSIM suited for evacuation simulations?

PedSim is an open source implementation typically used to add pedestrian dynamics to you own product. Is is therefore not *per se* suited for such evacuation simulations, because it lacks a convenient user interface. The provided sample user interface is more seen as a demonstration of the possibilities. The target user group is not planners, but system developers. However, the model of PedSim is perfectly able to simulate such an evacuation scenario.

> I see that PEDSIM can be downloaded as an open-source software or also under a commercial license from yourself, right? Is there any difference in the capabilities and features of these two, or in their functionality?

No, at the moment there is no difference. The availability of a commercial license is just mentioned in case somebody can not use the GPL for whatever reason. Please contact me if you are interested in this.

Model

> Is the model able to calculate an individual escape route for each of the persons within a subway station? (This includes the flow of people from/to the trains and the subway station)

Yes. However, at this point, the routing algorithm layer is not available for download yet. This means that the nearest escape exit (final point) has to be set manually per person or group of persons. Once the routing is available, this is done automatically.

> Is the model applicable in case of fire and/or toxic gas attack? (Because of the different characteristics of the different substances.)

> Is the model able to consider dispersal of pollutants?

No. However, libpedsim is extensible, you can implement this by yourself. Just define a model for the pollutants and add the reaction of the agents to you pedsim application. Characteristics of gases have not been implemented. You can easily extend the model for your own project.

> How long does the calculation take?

This depends on the number of persons simulated and the complexity of the scenario. In "realistic" examples, the speed of the simulation is more than 100 times faster than real-time.

> Is the model able to consider the architecture of a subway station e.g. platforms, corridors, stairways, etc.?

Yes, modeling architecture is possible. There are two categories of items in the model scenario: obstacles and paths. Obstacles can be invisible. This allows great flexibility for modeling any kind of architectural item.

> Is PEDSIM 3-dimensional?

While the model calculates almost everything in 3D internally, the ground is 2D only at the moment (so no level crossings).

> Is the model able to consider inaccessible ways or places (e.g. in case of a construction site)?

Yes. You can use invisible obstacles for this.

> If an agent moves in front of a convex building and the next waypoint is behind this building, the agent gets caught.

Yes, this is normal behavior. The agents try to proceed to the next waypoint directly. There is, at the moment, no routing algorithm included. Agents do not find their way around obstacles by themselves. The easiest solution to avoid this would be to add an additional waypoint on one side of the obstacle. Alternatively, you can add an 'invisible' wall to the obstacle to make it concave. Invisible since you add it to the model, but do not display it somewhere.

> If too many agents move through a very narrow path which is edged with obstacles, the force becomes so high that some of the agents are pushed trough the obstacles.

Since version 2.4.1, this should no longer happen. You can play with the obstacle forces as well. Usually, if this problem arises, it means that something is already wrong with the scenario. If pressure is that high, people would die in reality.

> Is it possible to consider further influencing factors?

Yes. As stated above, since the core library can be extended easily, you can add your own factors.

Installation, Compilation

Can I get a binary version of PEDSIM?

Yes and no. Every now and then I compile PEDSIM on various operating systems to check if everything is still fine everywhere. Theoretically I can send out these binary files. Please contact me. However, these binaries are usually a bit older than the repo release. And you still need a C++ compiler to do anything useful with PEDSIM. You can't really test your own scenarios using the pre-compiled binaries yet.

Do you have a Mac version? iPhone? Android?

I don't know. I do not own a Mac. I think is is possible to compile the core library (libpedsim) on any modern operating system, as long as a decent C++ compiler is available. Support is available for Linux and Windows.

All the graphical parts use the Qt framework, which is compatible with Mac, iOS and Android devices. See here. So theoretically, PEDSIM runs almost everywhere. But I can't test it.

Chapter 7

2-Dimensional Visualizer

The 2-dimensional visualizer is a separate application that can be used to visualize the output of a PEDSIM simulation run. It listens on a network socket for update information (agent positions, but also dynamic scene definitions.), which are rendered in real-time. This is perfectly suited to observe stand-allone simulations or optimizations which take a long time to run. Also nice for demonstrations, where the visualizer is installed on the machine connected to the beamer, and the simulation runs on a separate host. Ah and yes, it is able to output png files for each frame, which can be combined into a video easily!

2dvis is built on *Qt*. If you want to use all the features (especially charts), you need *Qt 5.7* or above. It should be possible to compile it using an oder version. However, you will not see the metrics charts in *2dvis* then. See documenation for compiling on Linux and Windows.

Usage

```
Usage: ./2dvis [options]
2-dimensional PEDSIM visualizer.

Options:
  -h, --help                      Displays this help.
  -q, --quiet                     Do not show graphical output
  -n, --network <port>            Read input from network on port <port>
  -f, --file <file>               Read input from <file>
  -c, --charts                    Display charts DockWidget
  -m, --metrics                   Display metrics DockWidget
  -o, --outputdirectory <directory>  Write frame-by-frame image output to <directory>
```

Usually, `2dvis` is started in network mode, where it listens to incoming data packets on the specified UDP port.

```
./2dvis -n 2222
```

Metrics and charts display

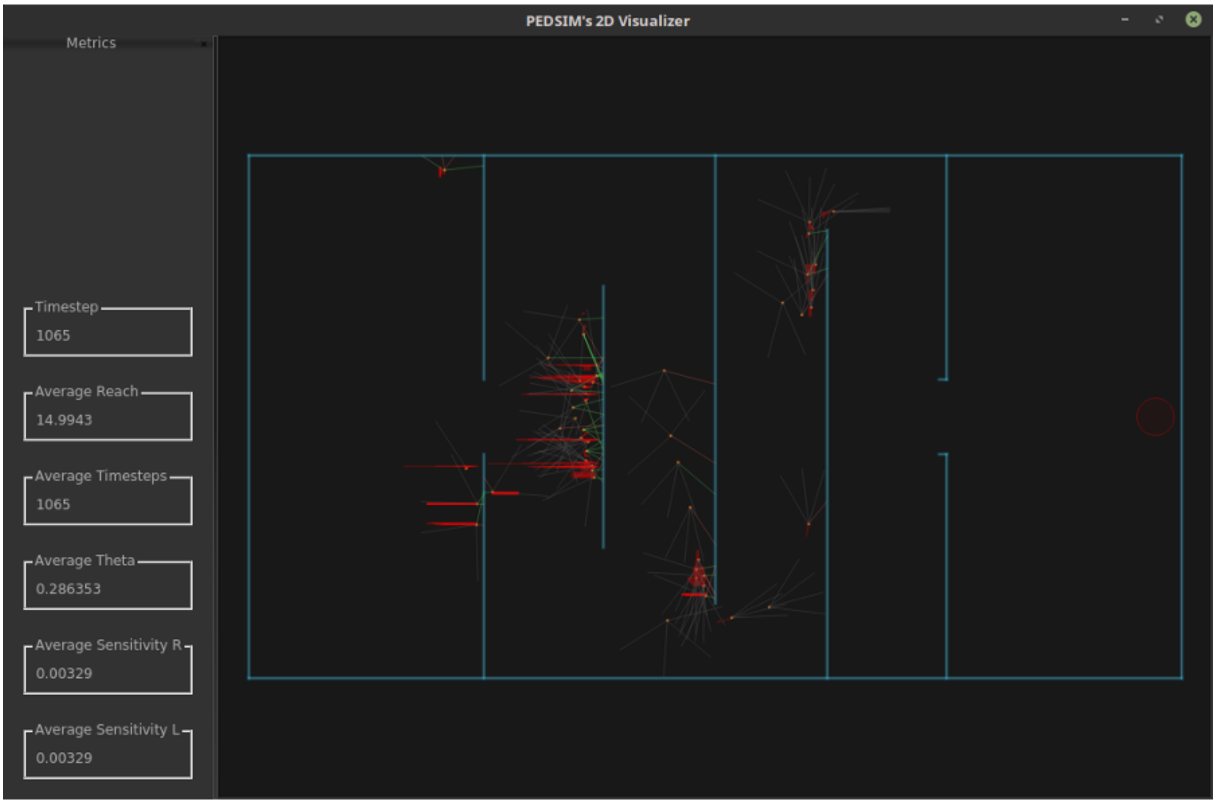
2dvis has the ability to display user-defined metrics coming from the simulation. It can display the latest metrics in numerical form, and also chart the values as line graphs. These two dockable windows are enabled by specifying `-m/--metrics` or `-c/--charts` respectively on the command line. Note that the charts window needs *Qt* version 5.7 or above. Otherwise the feature will not be compiled in. Numerical metrics work for all *Qt* versions.

These metrics are submitted from the simulation using Ped::XMLOutputWriter::writeMetrics(std::unordered_map<std::string, std::string> hash). For example like this:

```
ow->writeMetrics({{"name1", "value1"}, {"name2", "value2"}});
```

Here is an example with metrics transmitted:

```
ow->writeMetrics({
  {"Average Timesteps", std::to_string(sum_age/agents.size())},
  {"Average Theta", std::to_string(sum_theta/agents.size())},
  {"Average Sensitivity L", std::to_string(sum_sensitivity_l/agents.size())},
  {"Average Sensitivity R", std::to_string(sum_sensitivity_r/agents.size())},
  {"Average Reach", std::to_string(sum_reach/agents.size())}
});
```



Video generation

Instead of a network stream it is also possible to process a XML file containing the messages. This is meant for creating videos. At the moment, 2dvis will try to play all events in full speed, resulting in an overloaded graphics engine. Use it together with the `-o` output option only. This mode can be specified using

```
./2dvis -f filename.xml
```

In order to generate a video sequence out of a PEDSIM run, use these steps:

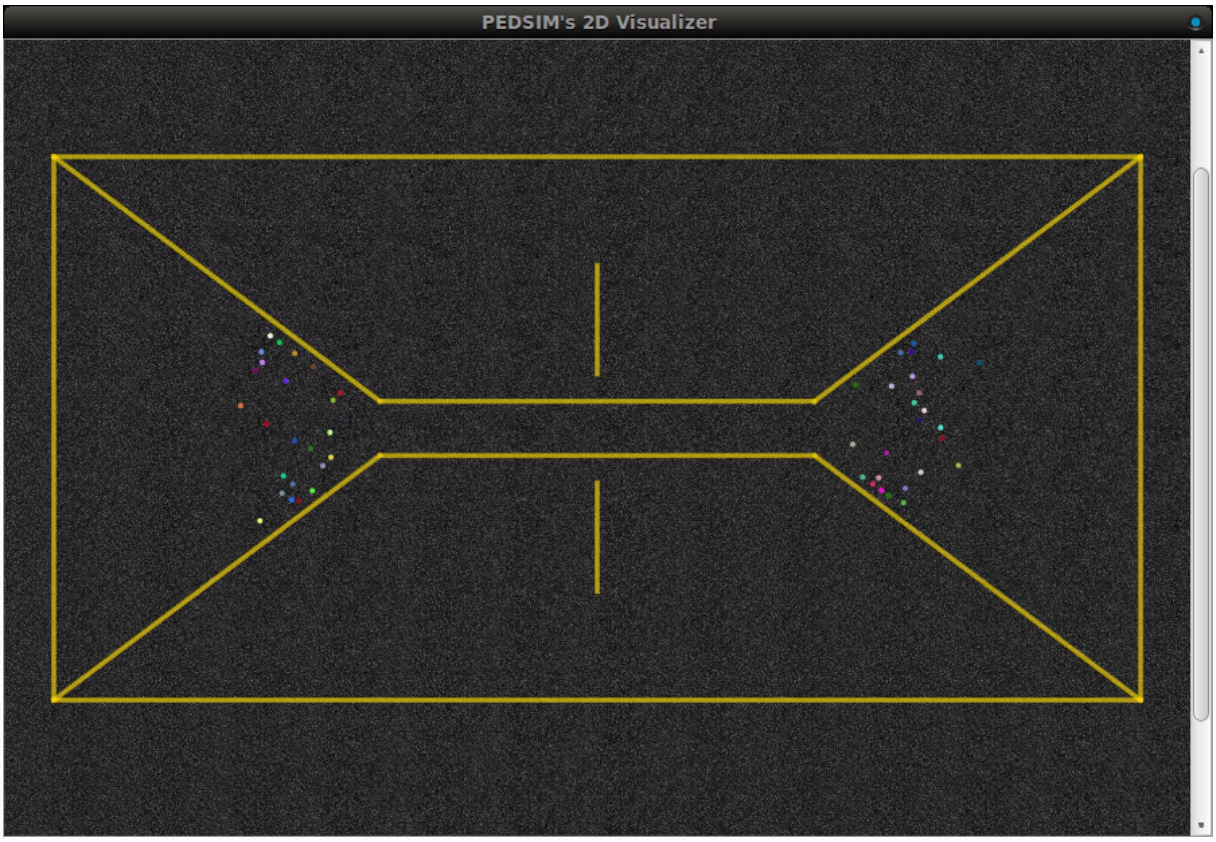
```
./2dvis -f ../../libpedsim/examples/pedsim_out.txt -o output
mencoder mf://output/*.png -mf w=1280:h=720:fps=25:type=png -ovc lavc -lavcopts
       vcodec=mpeg4:mbd=2:trell:vbitrate=6000 -oac copy -o example01.avi
```

Find more about *mencoder*, which is part of the *mplayer* suite, here.

For example videos, see PEDSIM's YouTube channel.

Supported XML tags

See here for a list of supported XML tags.



Chapter 8

3-Dimensional Visualizer

The 3-dimensional visualizer is a separate application that can be used to visualize the output of a PEDSIM simulation run. It listens on a network socket for update information (agent positions, but also dynamic scene definitions.), which are rendered in real-time. This is perfectly suited to observe stand-allone simulations or optimizations which take a long time to run. Also nice for demonstrations, where the visualizer is installed on the machine connected to the beamer, and the simulation runs on a separate host.

The difference between the two visualizers is that [2dvis](2-Dimensional Visualizer) is intended for displaying a technical view of the scenario, while 3dvis show a "real" view. 3dvis does not render the scenario realistically in any way - however, it only shows what would be visible in real live - i.e. no waypoints or forces.

3dvis is built on *Qt*. You need *Qt 5.7* or above with Qt3D (might be included, check when you install Qt). See further documenation for compiling on Linux and Windows.

Usage

`3dvis` is always started in network mode, where it listens to incoming data packets on the specified UDP port. In contrast to 2dvis, it can not render file based input.

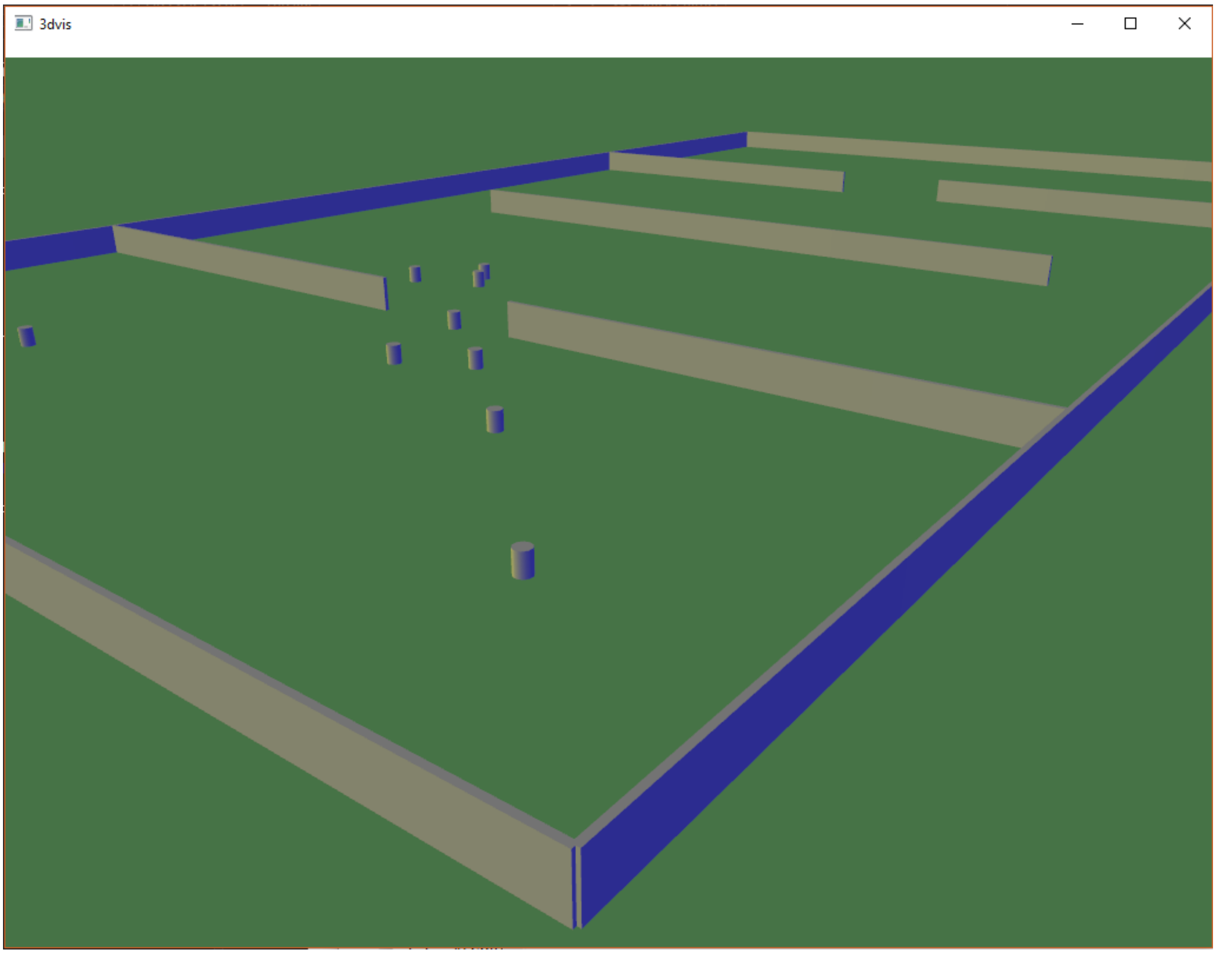
```
./3dvis
```

Video generation

There is no built in video generation mode for 3dvis. In order to caputre a video from a 3dvis animation, use a 3rd party capture tool, ideally one that supports your 3D graphics card natively.

Supported XML tags

See here for a list of valid XML tags. However, 3dvis does not render all tags. It only renders objects that have a physical representation, e.g. agents or obstacles.



Chapter 9

PEDSIM Demo Application

Introduction and purpose of the Demo Application

Originally, PEDSIM was a monolithic software package that was capable to read simple scenario definitions from a file and run the crowd simulation. This was PEDSIM version 1.x. For version 2.x, PEDSIM has been separated into a library and the Demo Application. These two pieces of software do more or less the same as PEDSIM Version 1.x, but with less import/export and scenario analysis features.

The main focus of development is on the PEDSIM library, since it is assumed that interested users are quickly able to develop their own software using the library. The Demo Application is still being maintained, but no longer extended. Its purpose is now to show how `libpedsim` can be used in a flexible way. It uses OOP inheritance to achive a tighter integration. These offers possibilities beyond what is presented in the code examples.

At the same time the Demo App is used as a *manual* integration test case. The library contains unit and user acceptance tests, based on the Google test framework. These tests are run automatically and are supposed to cover all possible aspects of failure. However, I believe that in the end a human has to look at the output of the system and judge if everything still looks sane. This is what the Demo Application is used for internally.

The scenario input features mentioned above are still a good starting point for your own small experiments. It is possible to define a scenario by writing a simple XML file, and use the user interface to quickly play with the various forces of the underlying model.

More Documentation

- GUI Documentation
- Scenario Definition

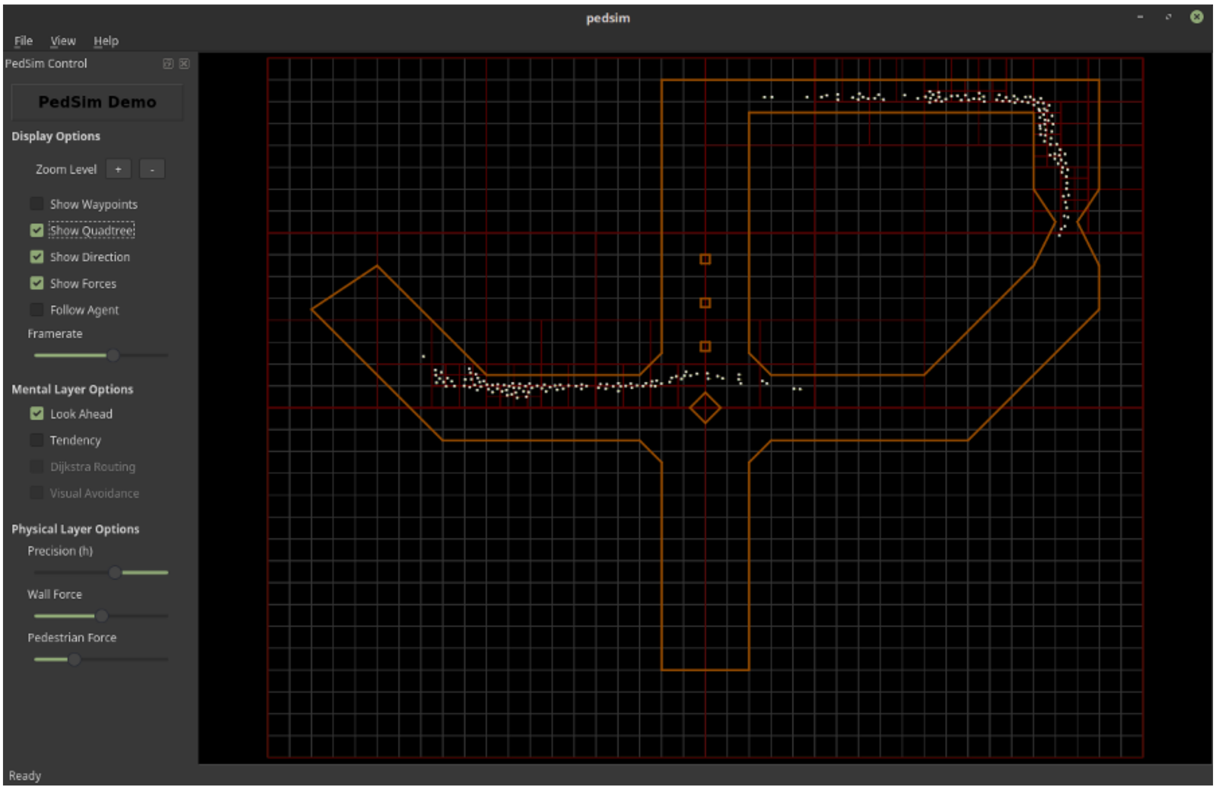
Compilation

Qt is needed to compile and run the demo application! See the installation documentation on Linux and Windows for more information regarding compillation of the source code.

Please note that (at least for the time being) the DemoApp does not link `libpedsim` dynamically using the .dll on Windows or the .so library on Linux respectively. This is due to bug related to incompatible compiler versions or settings in Qt and msvc15, which is often used to compile the library on Windows.

The source of `libpedsim` is directly included in the DemoApp Qt project file. This means that there is no need to compile the library separately at the moment. Compiling the DemoApp will also compile and statically link the library into the code. No need to link the library, or specify its location.

This method of including `libpedsim` can also be used by your own project. Make sure you do not violate the terms of the GPL doing this, e.g. by including the library source into your commercial projects. (Linking the library is OK under the terms of the LGPL.)



Chapter 10

GUI Documentation

PedSim Demo App Documentation

GUI Functionality

Using the GUI you can basically switch on and off certain predefined functionality, and set some parameters.

Display Options

- **Show Waypoints** - If this is checked, the waypoints are displayed. Only currently to an agent assigned waypoints can change their visibility, so you might have to wait a bit before all waypoints appear/disappear. Since several agents share one waypoint, it is not possible to display the details of the waypoint for each agent (they can adapt to the agent's direction). So only the information of the last agent assigned will be displayed, resulting in the waypoints changing their look during the simulation.
- **Show Quadtree** - Displays the quadtree used to store the agents internally. In order to find neighbors of agents quickly, they are grouped together in cells. These cells are generated and arranged dynamically.
- **Show Direction** - Displays the agent's direction. The yellow line represents where they actually do walk at the moment. The length of this line represents the velocity of the agent.
- **Show Forces** - Displays the forces that affect the individual agents. The forces are shown towards the direction the agents is accelerated. Red: direction they would like to walk to (desired direction). Blue: force that pulls them away from walls. Green: force that pulls them away from each other. Magenta: "Look Ahead" force.
- **Framerate** - Specify how many updates per second shoud be made. If the requested value is higher than what your computer can deliver, it will have no effect.

Mental Layer Options

- **Look Ahead** - This mental layer strategy is a bit more sophisticated. Each agent looks ahead a certain distance and counts the other agents to his left, and to his right, respectively. It then walks slightly into the direction where less agents were counted. Only agents in front of the agent, and only those with a walking direction in the opposite direction are considdered. So walking in lines behind each other is not affected.
- **Dijkstra Routing** - Not implemented in this demo (yet?), because the scenario is too small for this feature.
- **Visual Avoidance** -Not implemented in this demo (yet?).

Physical Layer Options

- **Precision (h)** - In each timestep (frame) of the simulation, the agent is allowed to walf forward a tiny step. All the accelerations and velocoties are scaled to match this step length. The precision defines how long such a step is (High precision = small steps). Setting precision too low will allow the angents to walk through walls and each other, since they only detect an obstacle when they are already through it. If the precision is high, the simulation will run very slowly, but the results are not supposed to change dramatically (except for rounding errors in the calculations, which can cause a slightly different overall result). The value h is also known as tau in literature and the documentation of the social force model.
- **Wall Force** - This slider defines how strong the force pushing away from the walls and other obstacles is (fiW). The higher it is, the bigger the distance an agent will keep from the wall will be. If the wall force is too low, agents will be able to walk through the walls. This will be more likely the larger the force between pedestrians becomes (see next slider), especially if the pedestrian density is very high (bottlenecks, or during an unevenly initialization).
- **Pedestrian Force** - Defines the distance each agent tries to keep from the other agents (fij). Set this to a high value if no mental layer strategies are activated (i.e. the simulation is run as a pure social force model simulation). Some mental layer strategies try to steer agents around other agents in advance, to this force is only used as a last change to avoid a collision. (or to actually simulate a collision, if set to a very low value.)

Chapter 11

Scenario Definition

PedSim Demo App Documentation

Scenario Definition

The scenario of the simulation is defined in a simple XML file. The default is `scene.xml`, placed in the same directory as the executable. However, it can also be specified on the command line:

```
./pedsim myscenario.xml
```

There must be a top-level tag in each XML document. In the predefined scenario file `scene.xml` and in the examples, it is `<scenario>`. At this moment, it does not matter what you write in there - as long as you have exacly one top-level tag.

Waypoints

The first item you should define are the waypoints, because they are used later in the agent definitions.

A waypoint has coordinates `x` and `y`, and a raduis `r`. A waypoint definition also contains the waypoint `id`, which is used to reference it later:

```
<waypoint id="w1" x="-160" y="-51" r=" 17" />
```

Agents

Agent definitions are a bit more complex. An agent has a start position with coordinates `x` and `y`.

Since it would not be comfortable to define each of potentially many agents individually, there is a way to specify groups of agents. In the agent definition, a agent multiplier `n` can be added. *N* copies of that agent will be spawned into the simulation. It would not be wise to place them all at the same location, so there is a way to spread them out a bit by specifying the `dx` and `dy` modifiers. The agents will be placed evenly distributed between *x-dx* and *x+dx* (resp y and dy).

Each agent has waypoints assigned. It will walk from the initial position to the first waypoint, then to the second and so on. The waypoints are added inside the `<agent></agent>` tag. Each added waypoint is a reference to a waypoint defined earlier. Please make sure that only waypoints really specified earlier are referenced.

Example

```
<agent x="60" y="0" n="100" dx="70" dy="10">
     <addwaypoint id="w1"/>
     <addwaypoint id="w2"/>
</agent>
```

Obstacles

Defining obstacles is very simple. Each obstacle is a line from coordinates `x1/y1` to coordinates `x2/y2`. It is not relevant where they are added, or how they are grouped.

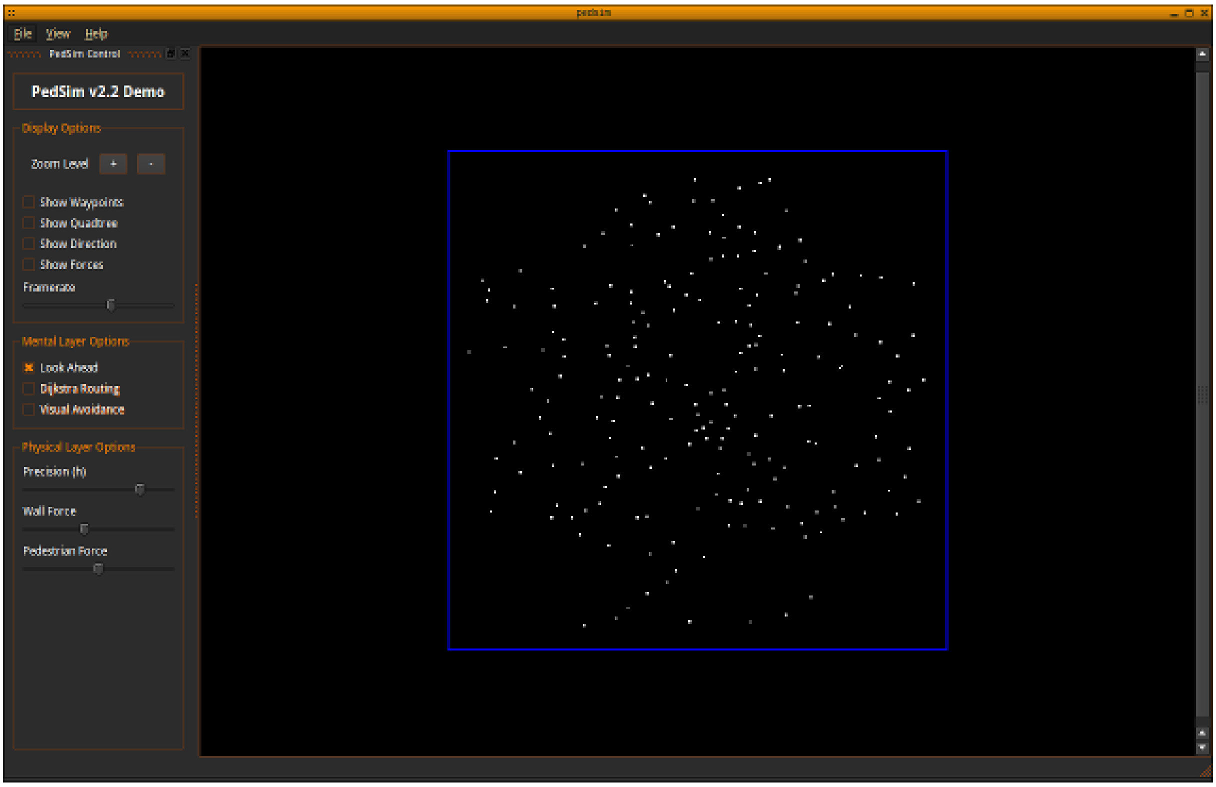
```
<obstacle x1="-2" y1="-50" x2="2" y2="-50" />
```

Combined Example

A box of four walls is defined, a waypoint on each side. 200 agents walk from one side of the box to the other.

```
<scenario>

  <waypoint id="wu" x="0" y="-100" r="50" />
  <waypoint id="wd" x="0" y=" 100" r="50" />
  <waypoint id="wl" x="-100" y="0" r="50" />
  <waypoint id="wr" x=" 100" y="0" r="50" />

  <agent x="0" y="0" n="100" dx="90" dy="90">
    <addwaypoint id="wu" />
    <addwaypoint id="wd" />
  </agent>

  <agent x="0" y="0" n="100" dx="90" dy="90">
    <addwaypoint id="wl" />
    <addwaypoint id="wr" />
  </agent>

  <obstacle x1="-110" y1="-110" x2=" 110" y2="-110" />
  <obstacle x1=" 110" y1="-110" x2=" 110" y2=" 110" />
  <obstacle x1=" 110" y1=" 110" x2="-110" y2=" 110" />
  <obstacle x1="-110" y1=" 110" x2="-110" y2="-110" />

</scenario>
```

This is a screenshot of the DemoApp displaying this example scenario:


pedsim
File View Help
PedSim Control
PedSim v2.2 Demo
Display Options
Zoom Level + -
Show Waypoints
Show Quadtree
Show Direction
Show Forces
Framerate
Mental Layer Options
Look Ahead
Dijkstra Routing
Visual Avoidance
Physical Layer Options
Precision (h)
Wall Force
Pedestrian Force

Chapter 12

Licensing

In short, the library `libpedsim` itself is licensed under terms of the LGPL, while all the rest under the terms of the GPL. This allows the library to be used even in commercial or proprietary software, if linked against the library dynamically.

12.1 The library libpedsim

GNU LESSER GENERAL PUBLIC LICENSE

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

0. Additional Definitions.

As used herein, "this License" refers to version 3 of the GNU Lesser General Public License, and the "GNU GPL" refers to version 3 of the GNU General Public License.

"The Library" refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

An "Application" is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A "Combined Work" is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the "Linked Version".

The "Minimal Corresponding Source" for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

The "Corresponding Application Code" for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work.

1. Exception to Section 3 of the GNU GPL.

You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL.

2. Conveying Modified Versions.

If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

- a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or
- b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

- a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License.
- b) Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

- a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License.
- b) Accompany the Combined Work with a copy of the GNU GPL and this license document.
- c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document.
- d) Do one of the following:
  - 0) Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.
  - 1) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user's computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version.
- e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.

You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

- a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License.
- b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy's public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.

12.2 Examples, DemoApp; i.e. everything except the libpedsim library

GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. http://fsf.org/

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

- a) The work must carry prominent notices stating that you modified it, and giving a relevant date.
- b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
- c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
- d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

- a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
- b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
- c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

- d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
- e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

- a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
- b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
- c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
- d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
- e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
- f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

Chapter 13

XML Messaging Format Specification

This is the message tags supported by *libpedsim*'s outputwriter class.

13.1 Supported XML Tags

<reset>

A client receiving this tag should reset its internal state, so that output can begin (again).

Argument	Description
n/a	-

Example: `<reset />`

<timestep>

This tag indicates the start of a new timestep. The client should advance one frame.

Argument	Description
value	This is the number of the timestep. It does not have to be an integer necessarily, as long as it is sortable. E.g. 'A00001' is a possible value. However, in *libpedsim* it is defined as type *long int*.

Example: `<timestep value="000001"/>`

<position>

This tag is used to transmit the position of an object.

Argument	Description

type	'agent', 'obstacle' and so on
id	The id of the object
x	The x co-ordinate of the object
y	The y co-ordinate of the object

Example: `<position type="agent" id="000001" x="25" y="-10" />`

<remove>

This tag is used to remove an object.

Argument	Description
type	'agent', 'obstacle' and so on
id	The id of the object

Example: `<remove type="agent" id="000001" />`

<scenario>

This tag is used to transmit the start of a new scenario

Argument	Description
name	The name of the new scenarion. It may be printed on the putput device.

Example: `<scenario name="Example 01" />`

<draw>

This tag is used to render a graphic item on the output device.

Argument	Description
type	The type of the graphical item to render. E.g. "line"
sx	The x co-ordinate of the start point (in case of a line)
sy	The y co-ordinate of the start point (in case of a line)
ex	(optional) The x co-ordinate of the end point (in case of a line)
ey	(optional) The y co-ordinate of the end point (in case of a line)
duration	How many timesteps the item will be displayed on the output device
red	The red value of the item's color (0.0 .. 1.0)
green	The green value of the item's color (0.0 .. 1.0)
blue	The blue value of the item's color (0.0 .. 1.0)

Example: `<draw type="line" sx="100" sy="100" ex="200" ey="200" duration="10" red="0.1" green="0.2" blue="1.0" />`

<metrics>

This tag is used to transmit measured metrics

Argument	Description
hash	A keyword-value hash, string string

Example: `<metrics> <metric key="name1" value="value1" /> <metric key="name2" value="value2" /> </metrics/>`

Chapter 14

Tests

PEDSIM uses *Google*'s gtest test framework for writing C++ tests. As a user of *libpedsim* running the library tests is not strictly necessary unless you play with the source of *libpedsim* and want to make sure you've not broken anything.

```
aptitude install libgtest-dev cmake valgrind
cd /usr/src/gtest/
cmake .
make
mv libgtest* /usr/lib/
```

To run the tests, run this in the *libpedsim* source folder:

```
make clean ; make
export LD_LIBRARY_PATH=.
make test
```

This export is needed since the library is not installed in a system wide known directory.

14.1 Memory Leak Test

Invoking `make test` will run a memory leak test. It uses Valgrind for this purpose, make sure you have it installed on your system. In order to perform this test the familiar code example `example01.cpp` will be compiled and executed, dynamically linked against `libpedsim`. This tests the bulk of the PEDSIM functionality for memory leaks. An example output is shown here:

```
==18274== Memcheck, a memory error detector
==18274== Copyright (C) 2002-2015, and GNU GPLd, by Julian Seward et al.
==18274== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==18274== Command: ./example01
==18274==
PedSim Example using libpedsim version 2.4
==18274==
==18274== HEAP SUMMARY:
==18274==     in use at exit: 72,704 bytes in 1 blocks
==18274==   total heap usage: 183,490 allocs, 183,489 frees, 17,025,573 bytes allocated
==18274==
==18274== LEAK SUMMARY:
==18274==    definitely lost: 0 bytes in 0 blocks
==18274==    indirectly lost: 0 bytes in 0 blocks
==18274==      possibly lost: 0 bytes in 0 blocks
==18274==    still reachable: 72,704 bytes in 1 blocks
==18274==         suppressed: 0 bytes in 0 blocks
==18274== Reachable blocks (those to which a pointer was found) are not shown.
==18274== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==18274==
==18274== For counts of detected and suppressed errors, rerun with: -v
==18274== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

14.2 Unit Tests

The folder `libpesim/tests/unit/` contains unit tests for most of the functions of libpedsim. That content is *not* included in this documentation. Please consult the source code directly if you are interested in the unit tests.

14.3 User Acceptance Tests

The folder `libpesim/tests/acceptance/` contains user acceptance tests for libpedsim. These are small test scenarios that test the behavior of the complete libpedsim functionality. First a scenario is defined, then the full simulation is run for a few timesteps. The output of that simulation is then compared against a high level description of the behavior dynamics.

This is a very important aspect for developers of the library - if these test pass, the expected behavior has probably not changed. It is not expected to change even if the underlying mechanisms are modified - it is quite stable and can be seen as a kind of guarantee to the library user.

Further, since these tests are self-contained small programs, they can serve naturally as **code examples** quite well. This is why their source code is included into the documentation verbatimly.

Move not if not Affected

This tests if an agent stays where it was placed as long as there are no forces affecting him. This is only the case if it is basically alone in a world without other agents or obstacles.

```
TEST_F(DynamicsTest, moveNotIfNotAffected) {
  Ped::Tagent *a = new Ped::Tagent();
  a->setPosition(50, 20, 0);
  pedscene->addAgent(a);

  for (int i=0; i<10; ++i) {
    pedscene->moveAgents(0.2);
  }

  vector<Ped::Tagent*> all = pedscene->getAllAgents();

  ASSERT_EQ(50, all[0]->getPosition().x);
  ASSERT_EQ(20, all[0]->getPosition().y);
  ASSERT_EQ( 0, all[0]->getPosition().z);

  // Cleanup
  for (Ped::Tagent* agent : pedscene->getAllAgents()) delete agent;
}
```

Move Towards End Point

This tests if the agent moves towards a waypoint that has been assigned to it.

```
TEST_F(DynamicsTest, moveTowardsWaypoint) {
  Ped::Tagent *a = new Ped::Tagent();
  a->setPosition(50, 0, 0);
  a->addWaypoint(w1);
  pedscene->addAgent(a);

  for (int i=0; i<10; ++i) {
    pedscene->moveAgents(0.2);
  }

  vector<Ped::Tagent*> all = pedscene->getAllAgents();

  ASSERT_GT(50, all.front()->getPosition().x);

  // Cleanup
  for (Ped::Tagent* agent : pedscene->getAllAgents()) delete agent;
}
```

Move Stops at Last Waypoint

If waypoint mode is set to BEHAVIOR_ONCE, the agent should stop once reached the last waypoint. This is only the case if the dynamics work in such a way that the agent's velocity reduced to 0 after a while without any forces affecting it. Like drag, or decaying momentum.

```
TEST_F(DynamicsTest, moveStopsAtLastWaypoint) {
  Ped::Tagent *a = new Ped::Tagent();
  a->setPosition(-50, 0, 0);
  a->addWaypoint(w1);
  a->setWaypointBehavior(Ped::Tagent::WaypointBehavior::BEHAVIOR_ONCE);
  pedscene->addAgent(a);

  // Move all agents for some time steps
  for (int i=0; i<200; ++i) { // It takes about 100 steps to reach the waypoint
    pedscene->moveAgents(0.5); // Only low precision required for this
  }

  vector<Ped::Tagent*> all = pedscene->getAllAgents();

  // The agent should be near the inner corner of the waypoint's
  // radius. w1 is from -1 to +1 (radius 2)
  EXPECT_NEAR(0.0, all.front()->getPosition().x, 2.0);
  EXPECT_NEAR(0.0, all.front()->getPosition().y, 2.0);

  // Cleanup
  for (Ped::Tagent* agent : pedscene->getAllAgents()) delete agent;
}
```

Move Axis Stability

This is a numerical stability test. 10 agents are placed very close to each other spread out on the x axis. Y and Z axis positions are identical for all agents. The force pushing them appart should only affect the x axis value of their positions.

```
TEST_F(DynamicsTest, moveAxisStability) {
  for (int i = 0; i<10; i++) {
    Ped::Tagent *a = new Ped::Tagent();
    a->setfactorlookaheadforce(0.0); // disable that
    a->setPosition(0.1 * i, 1.0, 0.0);
    pedscene->addAgent(a);
  }

  // Move all agents for some steps, with very high precision (0.03).
  for (int i=0; i<1000; ++i) {
    pedscene->moveAgents(0.03);
  }

  vector<Ped::Tagent*> all = pedscene->getAllAgents();

  // All agents should stay on y = 1.0 and move only horizontally on the x-axis.
  for (Ped::Tagent* agent : pedscene->getAllAgents()) {
    EXPECT_NEAR(1.0, agent->getPosition().y, 0.1);
  }

  // Cleanup
  for (Ped::Tagent* agent : pedscene->getAllAgents()) delete agent;
}
```

Chapter 15

Hierarchical Index

15.1 Class Hierarchy

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

Chapter 16

Class Index

16.1 Class List

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

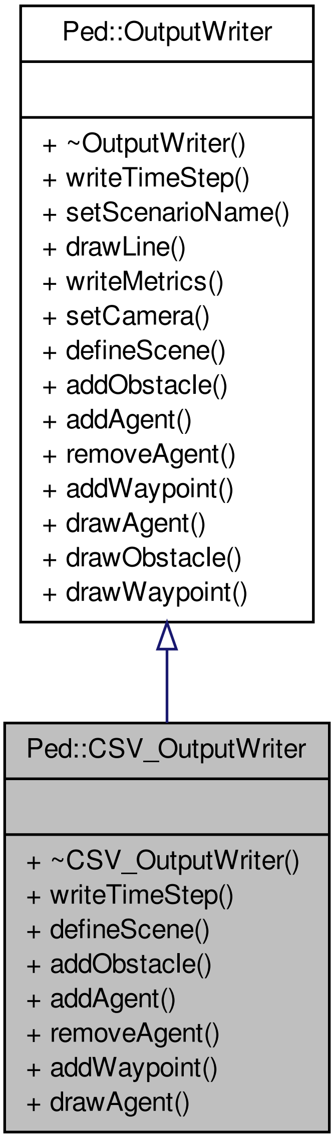
Chapter 17

Class Documentation

17.1 Ped::CSV_OutputWriter Class Reference

```
#include <ped_outputwriter.h>
```

Inheritance diagram for Ped::CSV_OutputWriter:



Public Member Functions

- virtual void **writeTimeStep** (long int timestep)
- virtual void **defineScene** (Tscene &s)
- virtual void **addObstacle** (Tobstacle &o)
- virtual void **addAgent** (Tagent &a)
- virtual void **removeAgent** (Tagent &a)
- virtual void **addWaypoint** (Twaypoint &w)
- virtual void **drawAgent** (Tagent &a)

17.1.1 Detailed Description

Class that defines a simple CSV OutputWriter

Author

chgloor

Date

2014-12-19

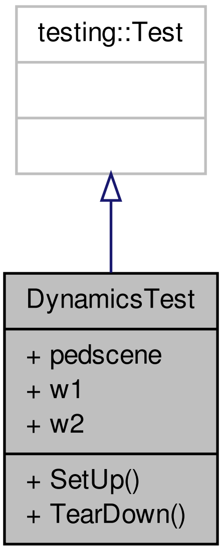
Definition at line 85 of file ped_outputwriter.h.

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

- ped_outputwriter.h

17.2 DynamicsTest Class Reference

Inheritance diagram for DynamicsTest:



Public Member Functions

- virtual void **SetUp** ()
- virtual void **TearDown** ()

Public Attributes

- Ped::Tscene * **pedscene**
- Ped::Twaypoint * **w1**
- Ped::Twaypoint * **w2**

17.2.1 Detailed Description

Definition at line 105 of file test_dynamics.cpp.

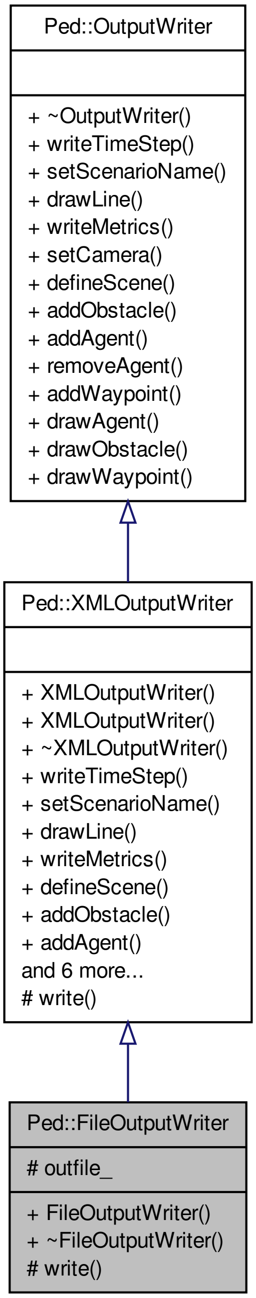
The documentation for this class was generated from the following file:

- tests/acceptance/test_dynamics.cpp

17.3 Ped::FileOutputWriter Class Reference

```
#include <ped_outputwriter.h>
```

Inheritance diagram for Ped::FileOutputWriter:



Public Member Functions

- FileOutputWriter ()
- virtual ∼FileOutputWriter ()

Protected Member Functions

- virtual void **write** (string message)

Protected Attributes

- ofstream **outfile_**

17.3.1 Detailed Description

Class that defines a frame-by-frame proprietary XMLOutputWriter. For supported tags, see XML Messaging Format Specification.

Author

chgloor

Date

2016-08-09

Examples:

examples/example01.cpp.

Definition at line 147 of file ped_outputwriter.h.

17.3.2 Constructor & Destructor Documentation

17.3.2.1 Ped::FileOutputWriter::FileOutputWriter ( )

Constructor used to open the output file

Date

2016-07-02

Definition at line 50 of file ped_outputwriter.cpp.

17.3.2.2 Ped::FileOutputWriter::∼FileOutputWriter ( ) [virtual]

Destructor used to close the output file

Date

2016-10-09

Definition at line 57 of file ped_outputwriter.cpp.

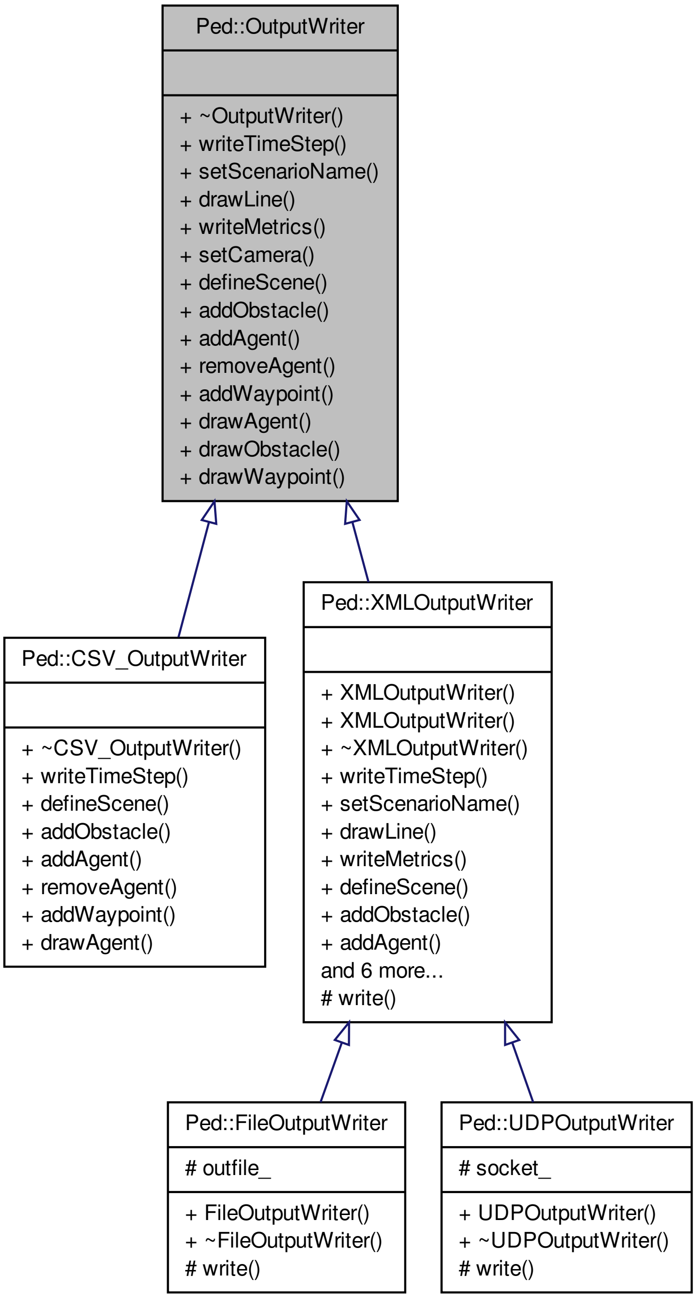
The documentation for this class was generated from the following files:

- ped_outputwriter.h
- ped_outputwriter.cpp

17.4 Ped::OutputWriter Class Reference

`#include <ped_outputwriter.h>`

Inheritance diagram for Ped::OutputWriter:



Public Member Functions

- virtual void **writeTimeStep** (long int timestep)=0

- virtual void **setScenarioName** (string name)=0
- virtual void **drawLine** (Tvector &s, Tvector &e, int duration=1, double red=1.0, double green=1.0, double blue=1.0)=0
- virtual void **writeMetrics** (std::unordered_map< std::string, std::string > hash)=0
- virtual void **setCamera** (Ped::Tvector pos, Ped::Tvector direction, string id="")=0
- virtual void **defineScene** (Tscene &s)=0
- virtual void **addObstacle** (Tobstacle &o)=0
- virtual void **addAgent** (Tagent &a)=0
- virtual void **removeAgent** (Tagent &a)=0
- virtual void **addWaypoint** (Twaypoint &w)=0
- virtual void **drawAgent** (Tagent &a)=0
- virtual void **drawObstacle** (Tobstacle &o)=0
- virtual void **drawWaypoint** (Twaypoint &w)=0

17.4.1 Detailed Description

Abstract Base Class that defines a Toutputwriter interface/default implementation.

Author

chgloor

Date

2014-12-18

Examples:

examples/example01.cpp, examples/example03.cpp, examples/example04.cpp, and examples/example05.-cpp.

Definition at line 51 of file ped_outputwriter.h.

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

- ped_outputwriter.h

17.5 Ped::Tagent Class Reference

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

Public Types

- enum **WaypointBehavior** { **BEHAVIOR_CIRCULAR** = 0, **BEHAVIOR_ONCE** = 1 }

Public Member Functions

- Tagent ()
- virtual ∼Tagent ()
- virtual void computeForces ()
- virtual void move (double stepSizeIn)
- virtual Tvector desiredForce ()
- virtual Tvector socialForce (const set< const Ped::Tagent ∗ > &neighbors)
- virtual Tvector obstacleForce (const set< const Ped::Tagent ∗ > &neighbors)
- virtual Tvector lookaheadForce (Tvector desired, const set< const Ped::Tagent ∗ > &neighbors)
- virtual Tvector myForce (Tvector desired, const set< const Ped::Tagent ∗ > &neighbors)
- void **setType** (int t)
- int **getType** () const
- void setVmax (double vmax)
- double getVmax ()
- void setFollow (int id)
- int getFollow () const
- int **getid** () const
- void setPosition (double px, double py, double pz)
- void **setPosition** (const Tvector &pos)
- Tvector **getPosition** () const
- Tvector **getVelocity** () const
- Tvector **getAcceleration** () const
- void setfactorsocialforce (double f)
- void setfactorobstacleforce (double f)
- void setfactordesiredforce (double f)
- void setfactorlookaheadforce (double f)
- void setscene (Tscene ∗s)
- Tscene ∗ getscene ()
- void addWaypoint (Twaypoint ∗wp)
- bool **removeWaypoint** (const Twaypoint ∗wp)
- void **clearWaypoints** ()
- deque< Twaypoint ∗ > **getWaypoints** ()
- bool **reachedDestination** ()
- void **setWaypointBehavior** (int mode)

Protected Attributes

- int id

  agent number
- Tvector p

  current position of the agent
- Tvector v

  velocity of the agent
- Tvector a

  current acceleration of the agent
- int **type**
- double vmax

  individual max velocity per agent
- int **follow**
- Ped::Tvector **desiredDirection**
- Ped::Tscene ∗ **scene**
- deque< Twaypoint ∗ > waypoints

coordinates of the next destinations
- Twaypoint ∗ destination

  coordinates of the next destination
- Twaypoint ∗ lastdestination

  coordinates of the last destination
- int waypointbehavior

  waypoints are round queues or not.
- bool **mlLookAhead**
- double **factordesiredforce**
- double **factorsocialforce**
- double **factorobstacleforce**
- double **factorlookaheadforce**
- double **obstacleForceSigma**
- Ped::Tvector **desiredforce**
- Ped::Tvector **socialforce**
- Ped::Tvector **obstacleforce**
- Ped::Tvector **lookaheadforce**
- Ped::Tvector **myforce**
- double **relaxationTime**
- double **agentRadius**
- long **timestep**

17.5.1 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:

examples/example01.cpp, examples/example02.cpp, examples/example03.cpp, examples/example04.cpp, and examples/example05.cpp.

Definition at line 51 of file ped_agent.h.

17.5.2 Constructor & Destructor Documentation

17.5.2.1 Ped::Tagent::Tagent ( )

Default Constructor

Date

2003-12-29

Definition at line 24 of file ped_agent.cpp.

17.5.2.2 Ped::Tagent::∼Tagent ( ) [virtual]

Destructor

Date

2012-02-04

Definition at line 63 of file ped_agent.cpp.

17.5.3 Member Function Documentation

17.5.3.1 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. Adding a waypoint will also selecting the first waypoint in the internal list as the active one, i.e. the first waypoint added will be the first point to headt to, no matter what is added later.

Author

chgloor

Date

2012-01-19

Examples:

examples/example01.cpp, examples/example02.cpp, and examples/example04.cpp.

Definition at line 94 of file ped_agent.cpp.

17.5.3.2 void Ped::Tagent::computeForces ( ) [virtual]

This is the first step of the 2-step update process used here. First, all forces are computed, using the t-1 agent positions as input. Once the forces are computed, all agent positions for timestep t are updated at the same time.

Definition at line 498 of file ped_agent.cpp.

17.5.3.3 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. This behavior can be controlled by a flag using setWaypointBehavior().

Date

2012-01-17

Returns

Tvector: the calculated force

Definition at line 236 of file ped_agent.cpp.

17.5.3.4 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

Definition at line 158 of file ped_agent.cpp.

17.5.3.5 Ped::Tscene * Ped::Tagent::getscene ( )

Returns the Tscene assigned to the agent.

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.

Returns

*s A Tscene initialized earlier, if one is assigned to the agent.

Definition at line 81 of file ped_agent.cpp.

17.5.3.6 double Ped::Tagent::getVmax ( )

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

Date

2016-08-10

Returns

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

Definition at line 175 of file ped_agent.cpp.

17.5.3.7 Ped::Tvector Ped::Tagent::lookaheadForce (Ped::Tvector *e*, const set< const Ped::Tagent * > & *neighbors*) [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.

Definition at line 439 of file ped_agent.cpp.

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

Does the agent dynamics stuff. In the current implementation a simple Euler integration is used. As the first step, the new position is calculated using t-1 velocity. Then, the new contributing individual forces are calculated. This will then be added to the existing velocity, which again is used during the next time step. See e.g. `https://en.wikipedia.org/wiki/Euler_method`

Date

2003-12-29

Parameters

h	Integration time step delta t

Definition at line 519 of file ped_agent.cpp.

17.5.3.9 Ped::Tvector Ped::Tagent::myForce (Ped::Tvector *e*, const set< const Ped::Tagent * > & *neighbors*) [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.

Definition at line 489 of file ped_agent.cpp.

17.5.3.10 Ped::Tvector Ped::Tagent::obstacleForce (const set< const Ped::Tagent * > & *neighbors*) [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

Definition at line 411 of file ped_agent.cpp.

17.5.3.11 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

Definition at line 215 of file ped_agent.cpp.

17.5.3.12 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

Definition at line 224 of file ped_agent.cpp.

17.5.3.13 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

Examples:

examples/example05.cpp.

Definition at line 206 of file ped_agent.cpp.

17.5.3.14 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

Examples:

examples/example04.cpp, and examples/example05.cpp.

Definition at line 197 of file ped_agent.cpp.

17.5.3.15 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)

Definition at line 149 of file ped_agent.cpp.

17.5.3.16 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:

examples/example01.cpp, examples/example02.cpp, examples/example03.cpp, and examples/example04.-cpp.

Definition at line 186 of file ped_agent.cpp.

17.5.3.17 void Ped::Tagent::setscene (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.

Definition at line 73 of file ped_agent.cpp.

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

17.5.3.18 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.

Examples:

examples/example05.cpp.

Definition at line 167 of file ped_agent.cpp.

17.5.3.19 Ped::Tvector Ped::Tagent::socialForce (const set< const Ped::Tagent * > & *neighbors*) [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 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

Definition at line 316 of file ped_agent.cpp.

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

- ped_agent.h
- ped_agent.cpp

17.6 Ped::Tobstacle Class Reference

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

Public Member Functions

- Tobstacle ()
- Tobstacle (double ax, double ay, double bx, double by)
- Tobstacle (const Tvector &startIn, const Tvector &endIn)
- virtual ~Tobstacle ()

  Destructor.
- int **getid** () const
- int **gettype** () const
- double **getax** () const
- double **getay** () const
- double **getbx** () const
- double **getby** () const
- Tvector **getStartPoint** () const
- Tvector **getEndPoint** () const
- virtual void setPosition (double ax, double ay, double bx, double by)
- virtual void **setPosition** (const Tvector &startIn, const Tvector &endIn)
- virtual void **setStartPoint** (const Tvector &startIn)
- virtual void **setEndPoint** (const Tvector &endIn)
- virtual void **setType** (int t)
- virtual Tvector closestPoint (double p1, double p2) const
- virtual Tvector **closestPoint** (const Tvector &pointIn) const
- virtual void rotate (double x, double y, double phi)

Protected Attributes

- int id

  Obstacle number.
- double ax

  Position of the obstacle.
- double ay

  Position of the obstacle.
- double bx

  Position of the obstacle.
- double by

  Position of the obstacle.
- int **type**

17.6.1 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:

examples/example01.cpp, examples/example02.cpp, examples/example03.cpp, examples/example04.cpp, and examples/example05.cpp.

Definition at line 34 of file ped_obstacle.h.

17.6.2 Constructor & Destructor Documentation

17.6.2.1 Ped::Tobstacle::Tobstacle ( )

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

Date

2012-01-07

Definition at line 15 of file ped_obstacle.cpp.

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

Constructor used to set initial 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.

Definition at line 27 of file ped_obstacle.cpp.

17.6.2.3 Ped::Tobstacle::Tobstacle (const Tvector & *startIn*, const Tvector & *endIn*)

Constructor used to set initial values.

Date

2013-08-02

Parameters

startIn	The first corner of the obstacle.
endIn	The second corner of the obstacle.

Definition at line 39 of file ped_obstacle.cpp.

17.6.3 Member Function Documentation

17.6.3.1 Ped::Tvector Ped::Tobstacle::closestPoint (double *p1*, double *p2*) const [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

Definition at line 115 of file ped_obstacle.cpp.

Referenced by Ped::Tagent::obstacleForce().

17.6.3.2 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.

Definition at line 127 of file ped_obstacle.cpp.

17.6.3.3 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.

Examples:

examples/example05.cpp.

Definition at line 69 of file ped_obstacle.cpp.

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

- ped_obstacle.h
- ped_obstacle.cpp

17.7 Ped::Tscene Class Reference

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

Public Member Functions

- Tscene ()
- Tscene (double left, double top, double width, double height)
- virtual ~Tscene ()
- virtual void **clear** ()
- virtual void addAgent (Tagent *a)
- virtual void addObstacle (Tobstacle *o)
- virtual void **addWaypoint** (Twaypoint *w)
- virtual bool removeAgent (Tagent *a)
- virtual bool removeObstacle (Tobstacle *o)
- virtual bool removeWaypoint (Twaypoint *w)
- 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
- const vector< Tobstacle * > & **getAllObstacles** () const
- const vector< Twaypoint * > & **getAllWaypoints** () const
- void **setOutputWriter** (OutputWriter *ow)

Protected Member Functions

- void placeAgent (const Ped::Tagent *a)
- void moveAgent (const Ped::Tagent *a)
- void **getNeighbors** (list< const Ped::Tagent * > &neighborList, double x, double y, double dist) const

Protected Attributes

- vector< Tagent * > **agents**
- vector< Tobstacle * > **obstacles**
- vector< Twaypoint * > **waypoints**
- Ttree * **tree**
- long int **timestep**

Friends

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

17.7.1 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 might 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 allowed 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:

examples/example01.cpp, examples/example02.cpp, examples/example03.cpp, examples/example04.cpp, and examples/example05.cpp.

Definition at line 68 of file ped_scene.h.

17.7.2 Constructor & Destructor Documentation

17.7.2.1 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

Definition at line 23 of file ped_scene.cpp.

17.7.2.2 Ped::Tscene::Tscene (double *left,* double *top,* 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.

Definition at line 36 of file ped_scene.cpp.

17.7.2.3 Ped::Tscene::∼Tscene ( ) [virtual]

Destructor

Date

2012-02-04

Definition at line 42 of file ped_scene.cpp.

17.7.3 Member Function Documentation

17.7.3.1 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::setscene() to assign itself to the agent.

Parameters

∗a	A pointer to the Tagent to add.

Examples:

examples/example01.cpp, examples/example02.cpp, examples/example03.cpp, examples/example04.cpp, and examples/example05.cpp.

Definition at line 71 of file ped_scene.cpp.

17.7.3.2 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:

examples/example01.cpp, examples/example02.cpp, examples/example03.cpp, examples/example04.cpp, and examples/example05.cpp.

Definition at line 84 of file ped_scene.cpp.

17.7.3.3 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

Definition at line 205 of file ped_scene.cpp.

17.7.3.4 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 searched for agents (search field is a square in the current implementation)

Definition at line 217 of file ped_scene.cpp.

17.7.3.5 void Ped::Tscene::moveAgent (const Ped::Tagent * *a*) [protected]

Moves a Tagent within the tree structure. The new position is taken from the agent. So it basically updates the tree structure for that given agent. Ped::Tagent::move(double h) calls this method automatically.

Date

2012-01-28

Parameters

∗a	the agent to move.

Definition at line 197 of file ped_scene.cpp.

17.7.3.6 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:

examples/example01.cpp, examples/example02.cpp, examples/example03.cpp, examples/example04.cpp, and examples/example05.cpp.

Definition at line 167 of file ped_scene.cpp.

17.7.3.7 void Ped::Tscene::placeAgent (const Ped::Tagent ∗ *a*) [protected]

Internally used to update the quadtree.

Date

2012-01-28

Parameters

Definition at line 187 of file ped_scene.cpp.

17.7.3.8 bool Ped::Tscene::removeAgent (Ped::Tagent ∗ *a*) [virtual]

Remove an agent from the scene.

Warning

Used to delete the agent. I don't think Tscene has ownership of the assigned objects. Will not delete from now on.

Definition at line 109 of file ped_scene.cpp.

17.7.3.9 bool Ped::Tscene::removeObstacle (Ped::Tobstacle * *o*) [virtual]

Remove an obstacle from the scene.

Warning

Used to delete the obstacle. I don't think Tscene has ownership of the assigned objects. Will not delete from now on.

Definition at line 132 of file ped_scene.cpp.

17.7.3.10 bool Ped::Tscene::removeWaypoint (Ped::Twaypoint * *w*) [virtual]

Remove a waypoint from the scene.

Warning

Used to delete the waypoint. I don't think Tscene has ownership of the assigned objects. Will not delete from now on.

Definition at line 144 of file ped_scene.cpp.

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

- ped_scene.h
- ped_scene.cpp

17.8 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 **clear** ()
- virtual void addAgent (const Ped::Tagent *a)
- virtual void moveAgent (const Ped::Tagent *a)
- virtual bool **removeAgent** (const Ped::Tagent *a)
- virtual set< const Ped::Tagent * > getAgents () const
- virtual void **getAgents** (list< const Ped::Tagent * > &outputList) 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 ()
- Ttree * **getChildByPosition** (double x, double y)

Protected Attributes

- bool **isleaf**
- double **x**
- double **y**
- double **w**
- double **h**
- int **depth**
- Ttree * **tree1**
- Ttree * **tree2**
- Ttree * **tree3**
- Ttree * **tree4**
- Ped::Tscene * **scene**

Friends

- class **Tscene**

17.8.1 Detailed Description

Definition at line 39 of file ped_tree.h.

17.8.2 Constructor & Destructor Documentation

17.8.2.1 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

Definition at line 19 of file ped_tree.cpp.

17.8.2.2 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

Definition at line 37 of file ped_tree.cpp.

17.8.3 Member Function Documentation

17.8.3.1 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

Definition at line 65 of file ped_tree.cpp.

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

A little helper that adds child nodes to this node

Author

chgloor

Date

2012-01-28

Definition at line 97 of file ped_tree.cpp.

17.8.3.3 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.

Definition at line 151 of file ped_tree.cpp.

17.8.3.4 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

Definition at line 188 of file ped_tree.cpp.

17.8.3.5 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

Definition at line 226 of file ped_tree.cpp.

17.8.3.6 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

Definition at line 125 of file ped_tree.cpp.

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

- ped_tree.h
- ped_tree.cpp

17.9 Ped::Tvector Class Reference

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

Public Member Functions

- Tvector ()
- **Tvector** (double px, double py, double pz=0)
- double length () const
- double lengthSquared () const
- void normalize ()
- Tvector normalized () const
- void scale (double factor)
- Tvector scaled (double factor) const
- Tvector **leftNormalVector** () const
- Tvector **rightNormalVector** () const
- double **polarRadius** () const
- double **polarAngle** () const

- double **angleTo** (const Tvector &other) const
- void rotate (double theta)
- Ped::Tvector rotated (double theta) const
- std::string **to_string** () const
- Tvector **operator+** (const Tvector &other) const
- Tvector **operator-** (const Tvector &other) const
- Tvector **operator*** (double factor) const
- Tvector **operator/** (double divisor) const
- Tvector & **operator+=** (const Tvector &vectorIn)
- Tvector & **operator-=** (const Tvector &vectorIn)
- Tvector & **operator*=** (double factor)
- Tvector & **operator*=** (const Tvector &vectorIn)
- Tvector & **operator/=** (double divisor)

Static Public Member Functions

- static double scalar (const Tvector &a, const Tvector &b)
- static double dotProduct (const Tvector &a, const Tvector &b)
- static Tvector crossProduct (const Tvector &a, const Tvector &b)
- static bool lineIntersection (const Ped::Tvector &p0, const Ped::Tvector &p1, const Ped::Tvector &p2, const Ped::Tvector &p3, Ped::Tvector *intersection)

Public Attributes

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

17.9.1 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.

Examples:

examples/example02.cpp, and examples/example03.cpp.

Definition at line 32 of file ped_vector.h.

17.9.2 Constructor & Destructor Documentation

17.9.2.1 Ped::Tvector::Tvector ( )

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

Date

2012-01-16

Definition at line 18 of file ped_vector.cpp.

17.9.3 Member Function Documentation

17.9.3.1 Ped::Tvector Ped::Tvector::crossProduct (const Tvector & *a*, const Tvector & *b*) [static]

Calculates the cross product of two vectors.

Date

2010-02-12

Parameters

&a	The first vector
&b	The second vector

Definition at line 91 of file ped_vector.cpp.

17.9.3.2 double Ped::Tvector::dotProduct (const Tvector & *a*, const Tvector & *b*) [static]

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

Date

2012-01-14

Returns

The dot product.

Parameters

&a	The first vector
&b	The second vector

Definition at line 82 of file ped_vector.cpp.

Referenced by Ped::Twaypoint::normalpoint(), and scalar().

17.9.3.3 double Ped::Tvector::length ( ) const

Returns the length of the vector.

Returns

the length

Examples:

examples/example03.cpp.

Definition at line 28 of file ped_vector.cpp.

Referenced by Ped::Twaypoint::getForce(), scalar(), and Ped::Tagent::socialForce().

17.9.3.4 double Ped::Tvector::lengthSquared ( ) const

Returns the length of the vector squared. This is faster than the real length.

Returns

the length squared

Definition at line 36 of file ped_vector.cpp.

Referenced by Ped::Twaypoint::normalpoint(), Ped::Tagent::obstacleForce(), and Ped::Tagent::socialForce().

17.9.3.5 bool Ped::Tvector::lineIntersection (const Ped::Tvector & *p0,* const Ped::Tvector & *p1,* const Ped::Tvector & *p2,* const Ped::Tvector & *p3,* Ped::Tvector ∗ *intersection*) [static]

Calculates the itnersection point of two lines, defined by Ped::Tvectors p0, p1, and p2, p3 respectively. Based on an algorithm in Andre LeMothe's "Tricks of the Windows Game Programming Gurus"

Returns

bool True if there is an intersection, false otherwise
∗intersection If the supplied pointer to a Ped::Tvector is not NULL, it will contain the intersection point, if there is an intersection.

Examples:

examples/example03.cpp.

Definition at line 264 of file ped_vector.cpp.

Referenced by Ped::Tagent::move().

17.9.3.6 void Ped::Tvector::normalize ( )

Normalizes the vector to a length of 1.

Date

2010-02-12

Definition at line 43 of file ped_vector.cpp.

17.9.3.7 Ped::Tvector Ped::Tvector::normalized ( ) const

Normalizes the vector to a length of 1.

Date

2013-08-02

Examples:

examples/example03.cpp.

Definition at line 57 of file ped_vector.cpp.

Referenced by Ped::Tagent::desiredForce(), Ped::Twaypoint::getForce(), Ped::Tagent::move(), Ped::Tagent::obstacleForce(), and Ped::Tagent::socialForce().

17.9.3.8 void Ped::Tvector::rotate (double *theta*)

Rotates a vector. Rotates around 0,0 in 2 dimensions only (z unchanged)v

Parameters

theta	in rad

Definition at line 285 of file ped_vector.cpp.

17.9.3.9 Ped::Tvector Ped::Tvector::rotated (double *theta*) const

Rotates a vector. Rotates around 0,0 in 2 dimensions only (z set to 0.0)

Parameters

theta	in rad

Examples:

examples/example03.cpp.

Definition at line 294 of file ped_vector.cpp.

17.9.3.10 double Ped::Tvector::scalar (const Tvector & *a*, const Tvector & *b*) [static]

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

Definition at line 72 of file ped_vector.cpp.

17.9.3.11 void Ped::Tvector::scale (double *factor*)

Scales this vector by a given factor in each dimension.

Date

2013-08-02

Parameters

factor	The scalar value to multiply with.

Definition at line 102 of file ped_vector.cpp.

17.9.3.12 Ped::Tvector Ped::Tvector::scaled (double *factor*) const

Returns a copy of this vector which is multiplied in each dimension by a given factor.

Date

2013-07-16

Returns

The scaled vector.

Parameters

factor	The scalar value to multiply with.

Examples:

examples/example03.cpp.

Definition at line 113 of file ped_vector.cpp.

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

- ped_vector.h
- ped_vector.cpp

17.10 Ped::Twaypoint Class Reference

```
#include <ped_waypoint.h>
```

Public Types

- enum **WaypointType** { **TYPE_NORMAL** = 0, **TYPE_POINT** = 1 }

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=NULL) const
- virtual Tvector normalpoint (const Tvector &p, const Tvector &obstacleStart, const Tvector &obstacleEnd) const
- virtual Tvector normalpoint (double p1, double p2, double oc11, double oc12, double oc21, double oc22) const
- void **setx** (double px)
- void **sety** (double py)
- void **setr** (double pr)
- void **settype** (WaypointType t)
- int **getid** () const
- int **gettype** () const
- double **getx** () const
- double **gety** () const
- double **getr** () const

Protected Attributes

- int id

    waypoint number
- double x

    position of the waypoint
- double y

    position of the waypoint
- double r

    position of the waypoint
- WaypointType type

    type of the waypoint

17.10.1 Detailed Description

The waypoint classs

Author

chgloor

Date

2012-01-07

Examples:

examples/example01.cpp, examples/example02.cpp, examples/example04.cpp, and examples/example05.-cpp.

Definition at line 37 of file ped_waypoint.h.

17.10.2 Constructor & Destructor Documentation

17.10.2.1 Ped::Twaypoint::Twaypoint ( )

Constructor - sets the most basic parameters.

Date

2012-01-07

Definition at line 25 of file ped_waypoint.cpp.

17.10.2.2 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

Definition at line 16 of file ped_waypoint.cpp.

17.10.2.3 Ped::Twaypoint::∼Twaypoint ( ) [virtual]

Default Destructor

Author

chgloor

Date

2012-02-04

Definition at line 34 of file ped_waypoint.cpp.

17.10.3 Member Function Documentation

17.10.3.1 Ped::Tvector Ped::Twaypoint::getForce (double *agentX*, double *agentY*, double *fromx*, double *fromy*, bool ∗ *reached* = NULL) const [virtual]

Returns the force into the direction of the waypoint

Date

2012-01-10

Parameters

agentX	The x coordinate of the current position of the agent
agentY	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

Definition at line 81 of file ped_waypoint.cpp.

17.10.3.2 Ped::Tvector Ped::Twaypoint::normalpoint (const Tvector & *p*, const Tvector & *obstacleStart*, const Tvector & *obstacleEnd*) 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

p	The point outside the obstacle
normalLineStart	The first corner of the normal line
normalLineEnd	The second corner of the normal line

Returns

Tvector The calculated point

Definition at line 44 of file ped_waypoint.cpp.

17.10.3.3 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

Definition at line 68 of file ped_waypoint.cpp.

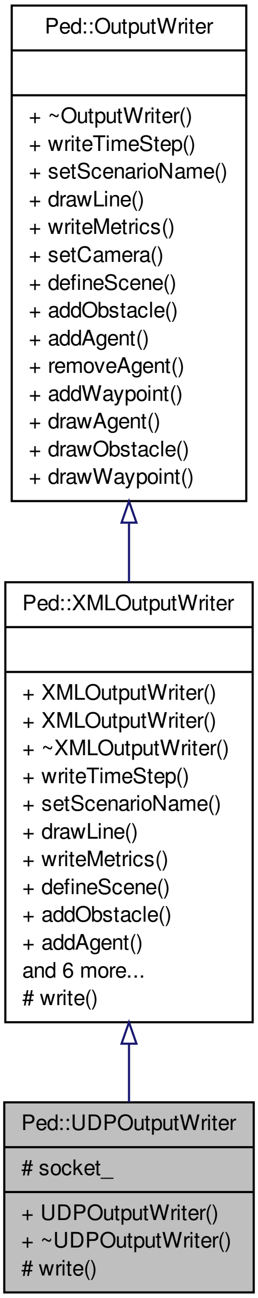
The documentation for this class was generated from the following files:

- ped_waypoint.h
- ped_waypoint.cpp

17.11 Ped::UDPOutputWriter Class Reference

`#include <ped_outputwriter.h>`

Inheritance diagram for Ped::UDPOutputWriter:



Public Member Functions

- UDPOutputWriter ()
- virtual ∼UDPOutputWriter ()

Protected Member Functions

- virtual void **write** (string message)

Protected Attributes

- SOCKET **socket_**

17.11.1 Detailed Description

Class that defines a frame-by-frame proprietary XMLOutputWriter that sends output over the network. For supported tags, see XML Messaging Format Specification.

Author

chgloor

Date

2016-10-09

Examples:

examples/example03.cpp, examples/example04.cpp, and examples/example05.cpp.

Definition at line 161 of file ped_outputwriter.h.

17.11.2 Constructor & Destructor Documentation

17.11.2.1 Ped::UDPOutputWriter::UDPOutputWriter ( )

Constructor used to open the network socket

Date

2016-10-09

Definition at line 69 of file ped_outputwriter.cpp.

17.11.2.2 Ped::UDPOutputWriter::∼UDPOutputWriter ( ) [virtual]

Destructor used to close the network socket

Date

2016-10-09

Definition at line 63 of file ped_outputwriter.cpp.

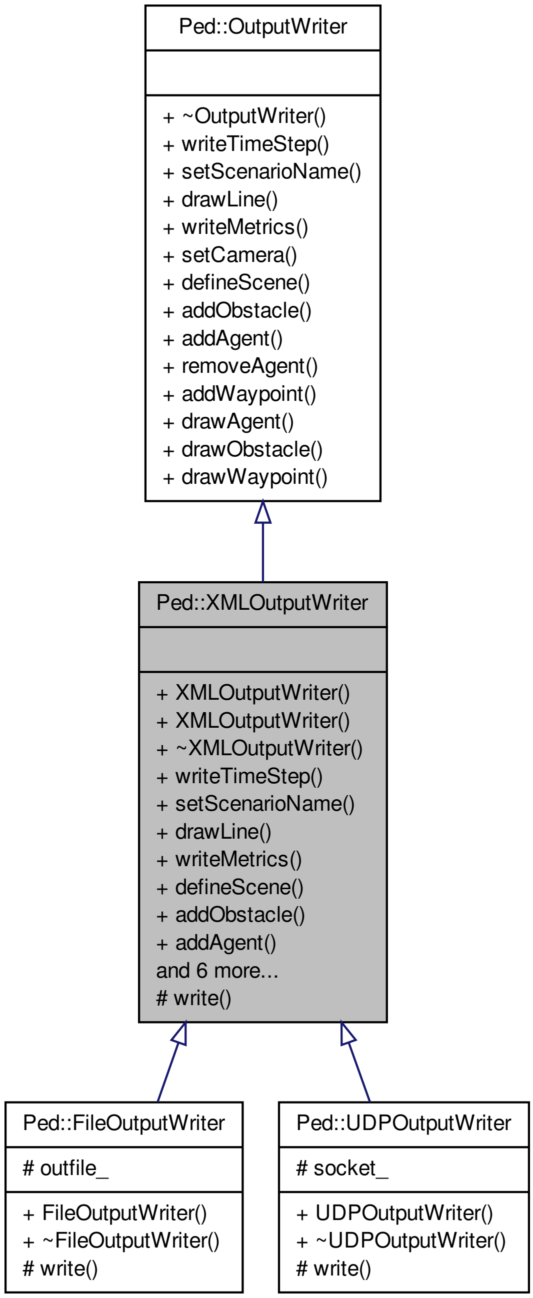
The documentation for this class was generated from the following files:

- ped_outputwriter.h
- ped_outputwriter.cpp

17.12 Ped::XMLOutputWriter Class Reference

```
#include <ped_outputwriter.h>
```

Inheritance diagram for Ped::XMLOutputWriter:



Public Member Functions

- XMLOutputWriter ()

- XMLOutputWriter (string scenarioname)
- virtual ∼XMLOutputWriter ()
- virtual void writeTimeStep (long int timestep)
- virtual void setScenarioName (string name)
- virtual void drawLine (Tvector &s, Tvector &e, int duration=1, double red=1.0, double green=0.0, double blue=0.0)
- virtual void writeMetrics (std::unordered_map< std::string, std::string > hash)
- virtual void **defineScene** (Tscene &s)
- virtual void **addObstacle** (Tobstacle &o)
- virtual void **addAgent** (Tagent &a)
- virtual void removeAgent (Tagent &a)
- virtual void **addWaypoint** (Twaypoint &w)
- virtual void setCamera (Ped::Tvector pos, Ped::Tvector direction, string id="")
- virtual void drawAgent (Tagent &a)
- virtual void drawObstacle (Tobstacle &o)
- virtual void drawWaypoint (Twaypoint &w)

Protected Member Functions

- virtual void **write** (string message)

17.12.1 Detailed Description

Class that defines a frame-by-frame proprietary XMLOutputWriter. For supported tags, see XML Messaging Format Specification.

Author

chgloor

Date

2016-07-02

Definition at line 110 of file ped_outputwriter.h.

17.12.2 Constructor & Destructor Documentation

17.12.2.1 Ped::XMLOutputWriter::XMLOutputWriter ( )

Constructor used to open the output mechanism.

Date

2016-07-02

Definition at line 42 of file ped_outputwriter.cpp.

17.12.2.2 Ped::XMLOutputWriter::XMLOutputWriter (string *name*)

Constructor used to open the output file

Date

2016-07-02

Parameters

scenarioname	Used to generate file filename

Definition at line 113 of file ped_outputwriter.cpp.

17.12.2.3 Ped::XMLOutputWriter::∼XMLOutputWriter ( ) [virtual]

Constructor used to close the output file

Date

2016-07-02

Parameters

scenarioname	Used to generate file filename

Definition at line 139 of file ped_outputwriter.cpp.

17.12.3 Member Function Documentation

17.12.3.1 void Ped::XMLOutputWriter::drawAgent (Tagent & *a*) [virtual]

Writes an agent's position

Date

2016-07-02

Parameters

a	The agent to be rendered.

Implements Ped::OutputWriter.

Definition at line 182 of file ped_outputwriter.cpp.

17.12.3.2 void Ped::XMLOutputWriter::drawLine (Tvector & *start*, Tvector & *end*, int *duration* = 1, double *red* = 1.0, double *green* = 0.0, double *blue* = 0.0) [virtual]

Draws a user defined line. This can be used to draw any line primitive on the output device, e.g., but not limited to, forces, boundaries, directions.

Date

2016-10-11

Parameters

s	Start point of the line
e	End point of the line
duration	The item will be visible for that many timesteps. Default is 1 timestep if omitted. 1 means it will disappear emmidiately when a new timestep starts. This can be used for animations of dynamic values.
red	The amount of red in the line color (between 0.0 and 1.0). Default is white.
green	The amount of green in the line color (between 0.0 and 1.0)
blue	The amount of blue in the line color (between 0.0 and 1.0)

Implements Ped::OutputWriter.

Definition at line 318 of file ped_outputwriter.cpp.

17.12.3.3 void Ped::XMLOutputWriter::drawObstacle (**Tobstacle & *o***) `[virtual]`

Writes an obstacle's position

Date

2016-10-10

Parameters

o	The obstacle to be rendered.

Implements Ped::OutputWriter.

Definition at line 220 of file ped_outputwriter.cpp.

17.12.3.4 void Ped::XMLOutputWriter::drawWaypoint (**Twaypoint & *w***) `[virtual]`

Writes a waypoint's position

Date

2016-10-16

Parameters

w	The waypoint to be rendered.

Implements Ped::OutputWriter.

Definition at line 254 of file ped_outputwriter.cpp.

17.12.3.5 void Ped::XMLOutputWriter::removeAgent (**Tagent & *a***) `[virtual]`

removes an agent from the scene

Date

2016-10-16

Parameters

a	The agent to be rendered.

Implements Ped::OutputWriter.

Definition at line 208 of file ped_outputwriter.cpp.

17.12.3.6 void Ped::XMLOutputWriter::setCamera (Ped::Tvector *pos*, Ped::Tvector *direction*, string *id* = "") [virtual]

Writes the camera position, used for 3D output renderes. They might ignore the camera position and use their own.

Date

2016-11-05

Parameters

pos	The position of the camera.
direction	The direction the camera lens faces.
id	The ID of the camera, if there are more than one.

Implements Ped::OutputWriter.

Definition at line 237 of file ped_outputwriter.cpp.

17.12.3.7 void Ped::XMLOutputWriter::setScenarioName (string *name*) [virtual]

Writes an scenario name

Date

2016-10-10

Parameters

name	The name of the scenarion. It will be printed on the output device. E.g. rendered on screen on 2dvis' file output.

Implements Ped::OutputWriter.

Definition at line 280 of file ped_outputwriter.cpp.

17.12.3.8 void Ped::XMLOutputWriter::writeMetrics (std::unordered_map< std::string, std::string > *hash*) [virtual]

Writes an list of metrics

Date

2016-10-17

Parameters

name	hash A unordered_map of metrics to send. E.g. called like ow->writeMetrics({{"name1", "value1"}, {"name2", "value2"}});

Implements Ped::OutputWriter.

Definition at line 352 of file ped_outputwriter.cpp.

17.12.3.9 void Ped::XMLOutputWriter::writeTimeStep (long int *timestep*) [virtual]

Writes the value of a timestep, indicating start of a new frame

Date

2016-07-02

Implements Ped::OutputWriter.

Definition at line 159 of file ped_outputwriter.cpp.

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

- ped_outputwriter.h
- ped_outputwriter.cpp

Chapter 18

Example Documentation

18.1 examples/example01.cpp

This is the very basic first example that shows how to use libpedsim. The output is sent to a file using Ped::File-OutputWriter.

Use something like this to compile:

```
g++ examples/example01.cpp -o example01 -lpedsim -L. -I. -std=c++11
export LD_LIBRARY_PATH=.
./example01
```

```
// pedsim - A microscopic pedestrian simulation system.
// Copyright (c) by Christian Gloor

#include <iostream>
#include <cstdlib>
#include <chrono>
#include <thread>

#include "ped_includes.h"

#include "ped_outputwriter.h"

using namespace std;

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

    // create an output writer which will send output to a file
    Ped::OutputWriter *ow = new Ped::FileOutputWriter();
    ow->setScenarioName("Example 01");

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

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

    pedscene->setOutputWriter(ow);

    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<10; 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 700 steps (and write their position through the outputwriter)
    for (int i=0; i<700; ++i) {
        pedscene->moveAgents(0.3);
        std::this_thread::sleep_for(std::chrono::milliseconds(3));
    }

    // Cleanup
    for (Ped::Tagent* agent : pedscene->getAllAgents()) delete agent;
    delete pedscene;
    delete w1;
    delete w2;
    delete o;
    delete ow;

    return EXIT_SUCCESS;
}
```

18.2 examples/example02.cpp

In this example we can see how to inherit a new agent class with different behaviour. The Ped::Tagent::myForce() function is extended for this. All other methods are left untouched.

Use something like this to compile:

```
g++ examples/example02.cpp -o example02 -lpedsim -L. -I. -std=c++11
export LD_LIBRARY_PATH=.
./example02
```

```
// pedsim - A microscopic pedestrian simulation system.
// Copyright (c) by Christian Gloor

#include "ped_includes.h"

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

using namespace std;

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

    class Tagent2: public Ped::Tagent {
    public:
        Ped::Tvector myForce(Ped::Tvector e, const set<const Ped::Tagent*> &
      neighbors) {
            Ped::Tvector lf;
            lf = -100.0*e;
            return lf;
        }
    };


    cout << "PedSim Example using libpedsim version " << Ped::LIBPEDSIM_VERSION << 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);

    int nagents = 1;

    for (int i = 0; i<nagents; i++) {
        Ped::Tagent *a = new Tagent2();

        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)
      {
          cout << (*iter)->getPosition().x << "/" << (*iter)->getPosition().y << endl;
        }
    }

    // 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;

    return EXIT_SUCCESS;
}
```

18.3 examples/example03.cpp

This example uses a new class that inherits from the library agent class. It shows how the Ped::Tagent::myForce() method can be used to add an additional force component to an agent to change its behaviour. This here basically turns the force-based pedestrian model into a Braitenberg vehicle (type 2a) like agent.

Use something like this to compile:

```
g++ examples/example03.cpp -o example03 -lpedsim -L. -I. -std=c++11
export LD_LIBRARY_PATH=.
./example03
```

```
// pedsim - A microscopic pedestrian simulation system.
// Copyright (c) by Christian Gloor

#include "ped_includes.h"
#include <iostream>
#include <cstdlib>
#include <chrono>
#include <thread>
#include <cmath>
#include <algorithm>

using namespace std;


class Tagent2: public Ped::Tagent {

public:
  Tagent2(Ped::OutputWriter *ow) : Ped::Tagent(), ow(ow), sensor_sensitivity(0.
      1) {
    factorobstacleforce = 10.0;
    factordesiredforce = 0.0;
    factorlookaheadforce = 0.0;
    v.x = 1; // needs some initial direction
  }

private:
  Ped::OutputWriter *ow;
  double sensor_sensitivity;

  double distance_sensor(Ped::Tvector direction) {
    double distance = std::numeric_limits<double>::infinity();
    Ped::Tvector intersection;
    bool has_intersection = false;
    for (auto obstacle : scene->getAllObstacles()) {
      Ped::Tvector ray = direction.normalized().scaled(1000.0); // max sensor
       view distance
      Ped::Tvector possibleintersection;
      if (Ped::Tvector::lineIntersection(p, p + ray, obstacle->getStartPoint(
      ), obstacle->getEndPoint(), &possibleintersection) == 1) {
        Ped::Tvector distvector = possibleintersection - p;
```

```
        double d = distvector.length();
        if (d < distance) {
          distance = d;
          intersection = possibleintersection;
        }
        has_intersection = true;
      }

    }

    if (has_intersection) {
      ow->drawLine(p, intersection, 1, 0.5, 0.5, 0.5);
    }
    return distance;
  }

  Ped::Tvector myForce(Ped::Tvector e, const set<const Ped::Tagent*> &neighbors) {
    Ped::Tvector lf;
    obstacleforce = obstacleForce(neighbors);
    if (obstacleforce.length() > 0.001) {
      sensor_sensitivity += 0.01;
      auto p1 = p + obstacleforce.scaled(100.0);
      ow->drawLine(p, p1, 10, 1.0, 0.0, 0.0);
    }
    lf = v.normalized();

    Ped::Tvector r1 = lf.rotated(0.5);
    Ped::Tvector r2 = lf.rotated(-0.5);

    double distance1 = distance_sensor(r1);
    double distance2 = distance_sensor(r2);

    double x = sensor_sensitivity * (1.0 / min(distance1, distance2)) * ((distance1>distance2) ? 1.0 : -1.0
      );
    sensor_sensitivity -= 0.0001;
    return lf.rotated(x);
  }

};


int main(int argc, char *argv[]) {
    cout << "PedSim Example using libpedsim version " << Ped::LIBPEDSIM_VERSION << endl;

    Ped::OutputWriter *ow = new Ped::UDPOutputWriter();
    ow->setScenarioName("Example 03");

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

    pedscene->addObstacle(new Ped::Tobstacle(0, -50,  0, +50));
    pedscene->addObstacle(new Ped::Tobstacle(-62,  -70,  -62, -10));
    pedscene->addObstacle(new Ped::Tobstacle(-62,   10,  -62,  70));
    pedscene->addObstacle(new Ped::Tobstacle( 62,  -70,   62, -10));
    pedscene->addObstacle(new Ped::Tobstacle( 62,   10,   62,  70));
    pedscene->addObstacle(new Ped::Tobstacle(-125, 70,  125, 70));
    pedscene->addObstacle(new Ped::Tobstacle(-125, -70,  125, -70));
    pedscene->addObstacle(new Ped::Tobstacle(-125, 70,  -125, -70));
    pedscene->addObstacle(new Ped::Tobstacle( 125, 70,   125, -70));

    int nagents = 10;

    for (int i = 0; i<nagents; i++) {
        Ped::Tagent *a = new Tagent2(ow);
        a->setPosition(0 + rand()/(RAND_MAX/100)-100, 0 + rand()/(RAND_MAX/30)-15, 0);
        pedscene->addAgent(a);
    }

    // move all agents for a few steps
    long timestep = 0;
    for (int i=0; i<10000; ++i) {
        pedscene->moveAgents(0.4);
        std::this_thread::sleep_for(std::chrono::milliseconds(1000/66));
    }

    // cleanup
    for (auto a : pedscene->getAllAgents()) { delete a; };
    for (auto o : pedscene->getAllObstacles()) { delete o; };
    delete pedscene;

    return EXIT_SUCCESS;
}
```

18.4 examples/example04.cpp

This is an example that shows how to build a bottleneck. Agents can not go through as fast as they want, and the queue up in front of the bottleneck. This is a typical crowd simulation scenario.

Use something like this to compile:

```
g++ examples/example04.cpp -o example04 -lpedsim -L. -I. -std=c++11
export LD_LIBRARY_PATH=.
./example04
```

```
#include "ped_includes.h"
#include <iostream>
#include <cstdlib>
#include <chrono>
#include <thread>
#include <cmath>
#include <algorithm>

using namespace std;


int main(int argc, char *argv[]) {
    cout << "PedSim Example using libpedsim version " << Ped::LIBPEDSIM_VERSION << endl;

    Ped::OutputWriter *ow = new Ped::UDPOutputWriter();
    ow->setScenarioName("Example 04");

    // setup
    Ped::Tscene *pedscene = new Ped::Tscene();
    pedscene->setOutputWriter(ow);

    // outer boudaries
    pedscene->addObstacle(new Ped::Tobstacle(-125, 70,  125, 70));
    pedscene->addObstacle(new Ped::Tobstacle(-125, -70,  125, -70));
    pedscene->addObstacle(new Ped::Tobstacle(-125, 70,  -125, -70));
    pedscene->addObstacle(new Ped::Tobstacle( 125, 70,   125, -70));

    // bottleneck
    double w = 2;
    //    pedscene->addObstacle(new Ped::Tobstacle(  30,   -w,   30,  w));

    pedscene->addObstacle(new Ped::Tobstacle( -30,   w,   30,  w));
    pedscene->addObstacle(new Ped::Tobstacle( -30,  -w,   30, -w));

    pedscene->addObstacle(new Ped::Tobstacle( -30,  -w,   -100, -70));
    pedscene->addObstacle(new Ped::Tobstacle( -30,   w,   -100,  70));
    pedscene->addObstacle(new Ped::Tobstacle(  30,  -w,    100, -70));
    pedscene->addObstacle(new Ped::Tobstacle(  30,   w,    100,  70));


    int nagents = 250;

    Ped::Twaypoint *w1 = new Ped::Twaypoint(100, 0, 24);
    pedscene->addWaypoint(w1);

    for (int i = 0; i<nagents; i++) {
      Ped::Tagent *a = new Ped::Tagent();
        a->setPosition(0.0 + rand()/(RAND_MAX/40.0)-100.0, 0.0 + rand()/(RAND_MAX/30.0)-15.0, 0.
      0);
        a->addWaypoint(w1);
        a->setWaypointBehavior(Ped::Tagent::BEHAVIOR_ONCE);
        a->setfactorsocialforce(10.0);
        pedscene->addAgent(a);
    }

    // move all agents for a few steps
    long timestep = 0;
    for (int i=0; i<10000; ++i) {
        pedscene->moveAgents(0.4);
        std::this_thread::sleep_for(std::chrono::milliseconds(1000/50));
    }

    // cleanup
    for (auto a : pedscene->getAllAgents()) { delete a; };
    for (auto o : pedscene->getAllObstacles()) { delete o; };
    delete pedscene;

    return EXIT_SUCCESS;
}
```

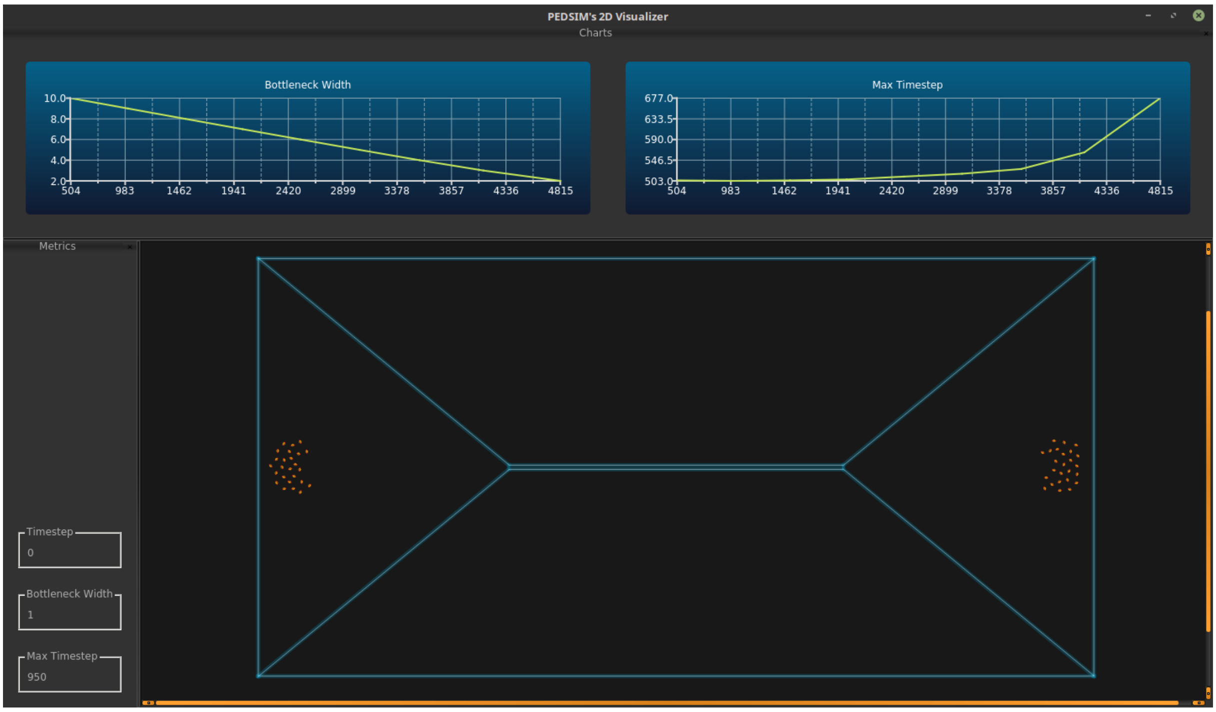
18.5 examples/example05.cpp

Example 05 is a small program used to demonstrate how PEDSIM can be used to find the ideal configuration of a scenario. In this simple demo case, a path width is modified on several automated simulation runs. The measured metric is the time until all agents have passed the opening. The output is sent to the 2-dimensional visualizer 2dvis, if launched.

You can find a video of this example on PEDSIM's YouTube channel.

Use something like this to compile:

```
g++ examples/example05.cpp -o example05 -lpedsim -L. -I. -std=c++11
export LD_LIBRARY_PATH=.
./example05
```




```
//
// pedsim - A microscopic pedestrian simulation system.
// Copyright (c) by Christian Gloor
//
// To collect the output in a file::
// ./example05 > out.dat
//
// Process output in gnuplot:
// gnuplot> plot "out.dat"

#include "ped_includes.h"

#include <iostream>
#include <sstream>

using namespace std;


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

  cout << "# PedSim Example using libpedsim version " << Ped::LIBPEDSIM_VERSION << endl;

  // setup
  Ped::Tscene *pedscene = new Ped::Tscene(); // no quadtree

  // create an output writer which will send output to a visualizer
  Ped::OutputWriter *ow = new Ped::UDPOutputWriter();
```

```
ow->setScenarioName("Example 05 / Dynamic Obstacles");
pedscene->setOutputWriter(ow);

// add one waypoint (=destination) with a small radius of 10 at the right end.
Ped::Twaypoint *w1 = new Ped::Twaypoint( 100, 0, 10);
Ped::Twaypoint *w2 = new Ped::Twaypoint(-100, 0, 10);

// create and add obstacles
pedscene->addObstacle(new Ped::Tobstacle(-100, -50,  100, -50));
pedscene->addObstacle(new Ped::Tobstacle(-100,  50,  100,  50));
pedscene->addObstacle(new Ped::Tobstacle(-100, -50, -100,  50));
pedscene->addObstacle(new Ped::Tobstacle( 100, -50,  100,  50));

// dynamic obstacles
Ped::Tobstacle *do1 = new Ped::Tobstacle(-40, -5,  40, -5);
pedscene->addObstacle(do1);
Ped::Tobstacle *do2 = new Ped::Tobstacle(-40, -5, -100, -50);
pedscene->addObstacle(do2);
Ped::Tobstacle *do3 = new Ped::Tobstacle( 40, -5,  100, -50);
pedscene->addObstacle(do3);
Ped::Tobstacle *do4 = new Ped::Tobstacle(-40,  5, 40,  5);
pedscene->addObstacle(do4);
Ped::Tobstacle *do5 = new Ped::Tobstacle(-40,  5, -100,  50);
pedscene->addObstacle(do5);
Ped::Tobstacle *do6 = new Ped::Tobstacle( 40,  5,  100,  50);
pedscene->addObstacle(do6);

// create agents
for (int i = 0; i<50; i++) {
  Ped::Tagent *a = new Ped::Tagent();
  a->setWaypointBehavior(Ped::Tagent::BEHAVIOR_ONCE);  // only once
  a->setVmax(1.2); // same speed for all agents
  a->setfactorsocialforce(10.0);
  a->setfactorobstacleforce(1.0);
  pedscene->addAgent(a);
}

// convenience
const vector<Ped::Tagent*>& myagents = pedscene->getAllAgents();
const vector<Ped::Tobstacle*>& myobstacles = pedscene->getAllObstacles();

ow->writeMetrics({
    {"Max Timestep", std::to_string(0)},
    {"Bottleneck Width", std::to_string((5-0)*2)}
  });

for (double h = 0; h < 5; h += 0.5) {

  // move obstacle
  do1->setPosition(-40, -5+h,  40, -5+h);
  ow->drawObstacle(*do1);
  do2->setPosition(-40, -5+h, -100, -50);
  ow->drawObstacle(*do2);
  do3->setPosition( 40, -5+h,  100, -50);
  ow->drawObstacle(*do3);
  do4->setPosition(-40,  5-h, 40,  5-h);
  ow->drawObstacle(*do4);
  do5->setPosition(-40,  5-h, -100,  50);
  ow->drawObstacle(*do5);
  do6->setPosition( 40,  5-h,  100,  50);
  ow->drawObstacle(*do6);

  long int timestep = 0;

  // reset agents
  for (vector<Ped::Tagent*>::const_iterator it = myagents.begin(); it != myagents.end(); ++it) {
    if ((*it)->getid() % 2 == 0) {
      (*it)->setPosition(-80, -25 + (*it)->getid(), 0);
      (*it)->addWaypoint(w1);
    } else {
      (*it)->setPosition( 80, -25 + (*it)->getid(), 0);
      (*it)->addWaypoint(w2);
    }
  }

  int notreached = myagents.size();
  while (notreached > 0) {
    timestep++;
    notreached = myagents.size();
    pedscene->moveAgents(0.4);

    for (auto a : myagents) {
      if (a->reachedDestination()) notreached--;
    }
    if (timestep >= 20000) notreached = 0; // seems to run forever.
  }
```

```
    cout << "# " << h << " " << timestep << endl;

    ow->writeMetrics({
        {"Max Timestep", std::to_string(timestep)},
        {"Bottleneck Width", std::to_string(2*(5-h))} // int for now
      });

  }

  // cleanup
  for (auto a : pedscene->getAllAgents()) { delete a; };
  for (auto w : pedscene->getAllWaypoints()) { delete w; };
  for (auto o : pedscene->getAllObstacles()) { delete o; };
  delete pedscene;

  return EXIT_SUCCESS;
}
```

Index