Traffic Simulator

Generated by Doxygen 1.9.1

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

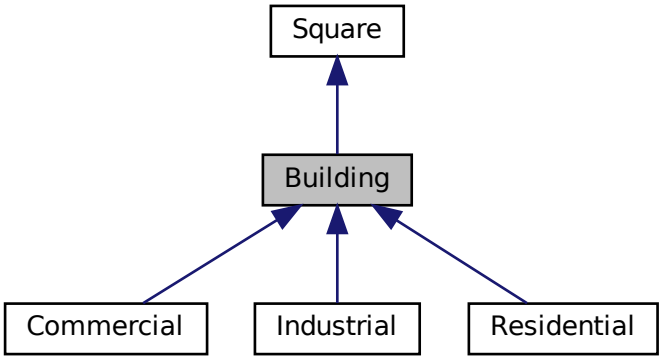
Class Documentation

3.1 Building Class Reference

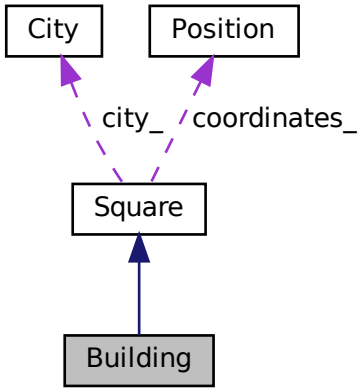
Building represents an building in the traffic simulation.

`#include <Building.hpp>`

Inheritance diagram for Building:



Collaboration diagram for Building:



Public Member Functions

- Building (Position pos, City &city, int capacity)

  Construct a new Building object.
- ~Building ()=default

  Destroy the Building object.
- virtual void initSquare ()

  Removes all residents and cars.
- int newResident (City &city, Position currentPos, int id)

  Create a new resident and add it to the building.
- int addResident (std::unique_ptr< Resident > &resident)

  Adds resident to the building.
- int removeResident (int residentId, const Position endPos, int carId)

  Creates a car and moves the resident from building to the car.
- virtual int addCar (std::unique_ptr< Car > &&car)

  Removes the car. This function is called when resident is moved from car to building.
- virtual int addPed (std::unique_ptr< Resident > &&ped)

  Adds a pedestrian to the building.
- virtual void simulate ()

  simulate moves a car inside the building to the road
- virtual int getCarCount () const

  Get how many cars are in the building.
- int getCapacity () const

  Get the capacity of the building.
- int getResidents () const

  Get the number of residents in the building.
- bool isBuilding () const

  Checks if the square is building or not.
- virtual BuildingType buildingType () const

  Checks the type of building.
- int joinCar (std::unique_ptr< Car > &&car, Direction dir)

  Calls the addCar function to remove the car.

Protected Attributes

- std::vector< std::unique_ptr< Resident > > residents_

  Vector containing the Resident pointers.
- std::vector< std::unique_ptr< Car > > cars_

  Vector containing the Car pointers.
- std::vector< std::unique_ptr< Resident > > pedestrians_

  Vector containing the Resident pointers that are going to walk.
- int capacity_

  capacity is the number of residents the building can take The user can affect only on the number of residential building capacity. It is set to 500 but can be changed.

3.1.1 Detailed Description

Building represents an building in the traffic simulation.

Building contains a vector for residents. When the resident is about to leave, it is moved to either pedestrians vector, or if going by car, it is moved to a car and cars have their own vector. There are also different type of buildings and different type of buildings have different capacities for residents.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 Building()

```
Building::Building (
            Position pos,
            City & city,
            int capacity )  [inline]
```

Construct a new Building object.

Parameters

pos	Position of the building
city	Reference to the city
capacity	The maximum number of people in the building

3.1.3 Member Function Documentation

3.1.3.1 addCar()

```
int Building::addCar (
            std::unique_ptr< Car > && car )  [virtual]
```

Removes the car. This function is called when resident is moved from car to building.

Parameters

car	Car object which is going to be removed

Returns

int 1

Reimplemented from Square.

3.1.3.2 addPed()

```
int Building::addPed (
            std::unique_ptr< Resident > && ped )  [virtual]
```

Adds a pedestrian to the building.

Parameters

ped	Resident to be added

Returns

int 1 if successful, 0 if not

Reimplemented from Square.

3.1.3.3 addResident()

```
int Building::addResident (
            std::unique_ptr< Resident > & resident )
```

Adds resident to the building.

Parameters

resident	Resident to be added

Returns

int 1 if successful, 0 if not

3.1.3.4 buildingType()

```
virtual BuildingType Building::buildingType ( ) const  [inline], [virtual]
```

Checks the type of building.

Returns

BuildingType None

Reimplemented from Square.

Reimplemented in Residential, Industrial, and Commercial.

3.1.3.5 getCapacity()

```
int Building::getCapacity ( ) const  [inline]
```

Get the capacity of the building.

Returns

int returns the capacity number

3.1.3.6 getCarCount()

```
int Building::getCarCount ( ) const  [virtual]
```

Get how many cars are in the building.

Returns

int number of cars

Reimplemented from Square.

3.1.3.7 getResidents()

```
int Building::getResidents ( ) const  [inline]
```

Get the number of residents in the building.

Returns

int number of residents

3.1.3.8 isBuilding()

```
bool Building::isBuilding ( ) const  [inline], [virtual]
```

Checks if the square is building or not.

Returns

bool true

Reimplemented from Square.

3.1.3.9 joinCar()

```
int Building::joinCar (
            std::unique_ptr< Car > && car,
            Direction dir )
```

Calls the addCar function to remove the car.

Parameters

car	The car to remove
dir	the direction of the car

Returns

int 1

3.1.3.10 newResident()

```
int Building::newResident (
            City & city,
            Position currentPos,
            int id )
```

Create a new resident and add it to the building.

Parameters

city	Reference to the city
currentPos	Position where the resident is created
id	Id for the new resident

Returns

int 1 if successful, 0 if not

3.1.3.11 removeResident()

```
int Building::removeResident (
            int residentId,
            const Position endPos,
            int carId )
```

Creates a car and moves the resident from building to the car.

Parameters

residentId	Id of the resident
endPos	The position where the car is supposed to go
carId	Id for the new car

Returns

int 1 if successful, 0 if not

3.1.3.12 simulate()

```
void Building::simulate ( )  [virtual]
```

simulate moves a car inside the building to the road

If there are cars inside the building this function moves one car to the road. If there doesn't exists a road the car can't move. If the first car can't join to the road the the function tries the next one.

If there are residents in the building who are about to walk to a new destination, they are moved from pedestrians vector the road

Reimplemented from Square.

3.1.4 Member Data Documentation

3.1.4.1 cars_

```
std::vector<std::unique_ptr<Car> > Building::cars_  [protected]
```

Vector containing the Car pointers.

Car is added to cars_ vector when the car is created for a resident and removed when the car leaves the building. Car never enters a building but is removed when the resident gets to its destination.

3.1.4.2 pedestrians_

```
std::vector<std::unique_ptr<Resident> > Building::pedestrians_  [protected]
```

Vector containing the Resident pointers that are going to walk.

When the resident is going to leave the building by walking, it is added to pedestrians_ vector. Pedestrian is moved to a road from the simulate function

3.1.4.3 residents_

```
std::vector<std::unique_ptr<Resident> > Building::residents_  [protected]
```

Vector containing the Resident pointers.

When a resident enters the building, it is moved to residents vector. When the resident leaves the building, it is removed from the vector

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

- src/Buildings/Building.hpp
- src/Buildings/Building.cpp

3.2 Car Class Reference

Car class represents a car in the city.

```
#include <Car.hpp>
```

Public Member Functions

- Car (City &city, const Position startPos, const Position endPos, int id)

  Constructor for the Car class.
- ~Car ()

  Destructor for the Car class.
- Car & operator= (Car &other)

  Assignment operator for the Car class.
- bool addPassenger (std::unique_ptr< Resident > &KimiRaikkonen)

  Adds a resident as a passenger to the car.
- std::unique_ptr< Resident > getPassenger ()

  Retrieves the passenger from the car.
- int findPath ()

  Finds the shortest path from the start position to the end position using A∗ algorithm.
- const Direction getDir () const

  Gets the current direction the car is facing.
- Direction peekDir () const

  Peeks at the next direction in the car's movement path.
- Direction updateDir ()

  Updates the direction the car is facing during movement.
- void createPath (std::vector< Direction > &path)

  Creates a custom path for the car.
- void simulate (Position cur)

  Simulates the daily routine and actions of residents inside the car.

3.2.1 Detailed Description

Car class represents a car in the city.

The car class contains the information about the car and the functions that simulate the car's actions in the simulation.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 Car()

```
Car::Car (
            City & city,
            const Position startPos,
            const Position endPos,
            int id )
```

Constructor for the Car class.

Parameters

city	Reference to the City object.
startPos	Initial position of the car.
endPos	Final destination of the car.
id	Unique identifier for the car.

3.2.2.2 ~Car()

```
Car::~Car ( )
```

Destructor for the Car class.

Moves residents in the car to their final destination building.

3.2.3 Member Function Documentation

3.2.3.1 addPassenger()

```
bool Car::addPassenger (
            std::unique_ptr< Resident > & KimiRaikkonen )
```

Adds a resident as a passenger to the car.

Parameters

KimiRaikkonen	Reference to the resident to be added as a passenger.

Returns

True if the addition is successful, false otherwise.

3.2.3.2 createPath()

```
void Car::createPath (
            std::vector< Direction > & path )
```

Creates a custom path for the car.

Parameters

path↩ _	The vector of directions representing the custom path.

3.2.3.3 findPath()

```
int Car::findPath ( )
```

Finds the shortest path from the start position to the end position using A∗ algorithm.

Returns

The length of the found path, or -1 if no path is found.

3.2.3.4 getDir()

```
const Direction Car::getDir ( ) const
```

Gets the current direction the car is facing.

Returns

The current direction of the car.

3.2.3.5 getPassenger()

```
std::unique_ptr< Resident > Car::getPassenger ( )
```

Retrieves the passenger from the car.

Returns

A unique pointer to the passenger resident, or nullptr if there is no passenger.

3.2.3.6 operator=()

```
Car & Car::operator= (
            Car & other )
```

Assignment operator for the Car class.

Parameters

other	Another Car object for assignment.

Returns

A reference to the assigned Car object.

3.2.3.7 peekDir()

```
Direction Car::peekDir ( ) const
```

Peeks at the next direction in the car's movement path.

Returns

The next direction in the movement path.

3.2.3.8 simulate()

```
void Car::simulate (
            Position cur )
```

Simulates the daily routine and actions of residents inside the car.

Parameters

cur	The current position of the car.

3.2.3.9 updateDir()

```
Direction Car::updateDir ( )
```

Updates the direction the car is facing during movement.

Returns

The updated direction the car is facing.

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

- src/Cars/Car.hpp
- src/Cars/Car.cpp

3.3 City Class Reference

CityStats class is used to store the city as a grid of squares.

`#include <City.hpp>`

Public Member Functions

- City (size_t size)
  Construct a new City object.
- ~City ()
  destroys the city object
- void add (Position pos, Square *square)
  add adds (or replaces) a square in the city
- Square * getSquare (Position pos)
  getSquare returns the square in the given position
- const std::vector< std::vector< Square * > > & getCity () const
  getCity returns the grid of squares
- size_t getCitySize () const
  getCitySize returns the size of the city
- int getResidentCount () const
  getResidentCount returns the amount of residents in the city
- std::vector< Square * > getIndustrials () const
  getIndustrials returns the vector of industrial squares
- std::vector< Square * > getCommercials () const
  getCommercials returns the vector of commercial squares
- std::vector< Square * > getResidentials () const
  getResidentials returns the vector of residential squares
- std::vector< Road * > getRoads () const
  getRoads returns the vector of roads
- void simulate ()
  simulates everything in the city
- void init ()
  initializes everything in the city for simulation
- double getTime () const
  getTime returns the time in the city in 24h format
- double currentRoadTrafficPercentage () const
  Retrieves the current average road traffic percentage from all roads.
- double averageRoadTrafficPercentage () const
  Calculates and returns the average road traffic percentage from all roads from the entire simulation.
- std::vector< Intersection * > getIntersections () const
  Retrieves a vector of pointers to intersections.
- int getResidentAmount () const
  Retrieves the amount of residents in the city.
- int carsOnRoads () const
  carsOnRoads returns the amount of cars on the roads
- std::vector< double > getStats () const
  Retrieves the hourly averages from the statistics and returns them.
- void clearAndResize (size_t size)
  Clears the existing data and resizes the container to the specified size, this is needed for importing different size cities.

- std::vector< int > getHourCounters () const

  Returns the "hour counters", meaning how many times has a specific hour been simulated.
- void exportStats (const std::string &filename)

  Exports the statistics data to a file with the specified filename.
- int getPedCount () const

  Returns the amount of pedestrians currently traveling on roads.
- int newResidentId ()

  Returns new ID for new resident.

3.3.1 Detailed Description

CityStats class is used to store the city as a grid of squares.

City is a grid of squares. City also tracs many statistics, e.g. time, and the ammount of total traffic in the city.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 City()

```
City::City (
            size_t size )
```

Construct a new City object.

Parameters

size	The city is going to be size x size grid

3.3.3 Member Function Documentation

3.3.3.1 add()

```
void City::add (
            Position pos,
            Square * square )
```

add adds (or replaces) a square in the city

Parameters

pos	Position where to add the square
square	Pointer to the square to be added

3.3.3.2 averageRoadTrafficPercentage()

```
double City::averageRoadTrafficPercentage ( ) const
```

Calculates and returns the average road traffic percentage from all roads from the entire simulation.

Returns

The average road traffic percentage as a double.

3.3.3.3 carsOnRoads()

```
int City::carsOnRoads ( ) const
```

carsOnRoads returns the amount of cars on the roads

Returns

int amount of cars on the roads

3.3.3.4 clearAndResize()

```
void City::clearAndResize (
            size_t size )
```

Clears the existing data and resizes the container to the specified size, this is needed for importing different size cities.

Parameters

size	The new size of the container.

3.3.3.5 currentRoadTrafficPercentage()

```
double City::currentRoadTrafficPercentage ( ) const
```

Retrieves the current average road traffic percentage from all roads.

Returns

The current road traffic percentage as a double.

3.3.3.6 exportStats()

```
void City::exportStats (
            const std::string & filename )
```

Exports the statistics data to a file with the specified filename.

Parameters

filename	The name of the file to which the statistics will be exported. This is "TrafficData.csv" in the program.

3.3.3.7 getCity()

```
const std::vector<std::vector<Square *> >& City::getCity ( ) const  [inline]
```

getCity returns the grid of squares

Returns

std::vector<std::vector<Square*>> The grid of squares

3.3.3.8 getCitySize()

```
size_t City::getCitySize ( ) const  [inline]
```

getCitySize returns the size of the city

Returns

size_t size of the city

3.3.3.9 getCommercials()

```
std::vector< Square * > City::getCommercials ( ) const
```

getCommercials returns the vector of commercial squares

Returns

std::vector<Square*> vector of commercial squares

3.3.3.10 getHourCounters()

```
std::vector<int> City::getHourCounters ( ) const  [inline]
```

Returns the "hour counters", meaning how many times has a specific hour been simulated.

Returns

A vector of int representing the hour counters.

3.3.3.11 getIndustrials()

```
std::vector< Square * > City::getIndustrials ( ) const
```

getIndustrials returns the vector of industrial squares

Returns

std::vector<Square*> vector of industrial squares

3.3.3.12 getIntersections()

```
std::vector< Intersection * > City::getIntersections ( ) const
```

Retrieves a vector of pointers to intersections.

Returns

A vector of Intersection pointers representing the intersections.

3.3.3.13 getResidentAmount()

```
int City::getResidentAmount ( ) const
```

Retrieves the amount of residents in the city.

Returns

The number of residents as an integer.

3.3.3.14 getResidentCount()

```
int City::getResidentCount ( ) const  [inline]
```

getResidentCount returns the amount of residents in the city

Returns

int amount of residents in the city

3.3.3.15 getResidentials()

```
std::vector< Square * > City::getResidentials ( ) const
```

getResidentials returns the vector of residential squares

Returns

std::vector<Square*> vector of residential squares

3.3.3.16 getRoads()

```
std::vector< Road * > City::getRoads ( ) const
```

getRoads returns the vector of roads

Returns

std::vector<Road*> vector of roads

3.3.3.17 getSquare()

```
Square * City::getSquare (
            Position pos )
```

getSquare returns the square in the given position

Parameters

pos	Position of the square

Returns

Square* Pointer to the square

3.3.3.18 getStats()

```
std::vector<double> City::getStats ( ) const  [inline]
```

Retrieves the hourly averages from the statistics and returns them.

Returns

A vector of doubles representing the hourly averages.

3.3.3.19 getTime()

```
double City::getTime ( ) const  [inline]
```

getTime returns the time in the city in 24h format

Returns

double time in the city

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

- src/City.hpp
- src/City.cpp

3.4 CityStats Struct Reference

Struct representing statistics for the entire city.

```
#include <Utilities.hpp>
```

Public Member Functions

- void incrementCounter (int index, double percentage)

  Increments the counter and sum for a specific hour.
- std::vector< double > getHourlyAverages () const

  Calculates and retrieves the hourly averages for percentages.

Public Attributes

- std::vector< int > **counters**
- std::vector< double > **sums**

3.4.1 Detailed Description

Struct representing statistics for the entire city.

3.4.2 Member Function Documentation

3.4.2.1 getHourlyAverages()

```
std::vector<double> CityStats::getHourlyAverages ( ) const  [inline]
```

Calculates and retrieves the hourly averages for percentages.

Returns

A vector of double representing the hourly averages for percentages.

3.4.2.2 incrementCounter()

```
void CityStats::incrementCounter (
            int index,
            double percentage )  [inline]
```

Increments the counter and sum for a specific hour.

Parameters

index	Hour index.
percentage	Percentage value to be added to the sum.

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

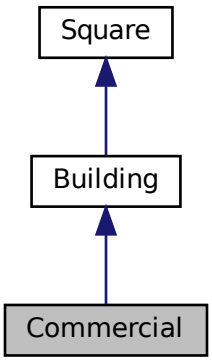
- src/Utilities.hpp

3.5 Commercial Class Reference

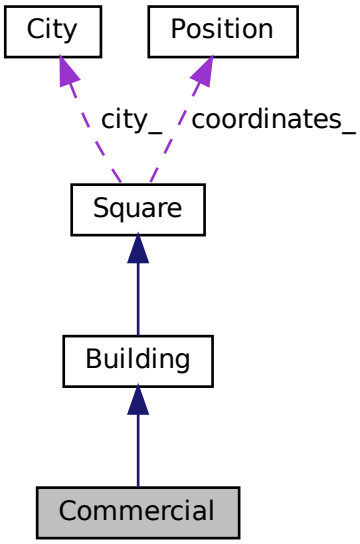
Commercial building represents an commercial building in the traffic simulation.

`#include <Commercial.hpp>`

Inheritance diagram for Commercial:



Collaboration diagram for Commercial:



Public Member Functions

- Commercial (Position pos, City &city, int capacity)

  Construct a new Commercial building object.
- BuildingType buildingType () const

  Checks the type of building.

Additional Inherited Members

3.5.1 Detailed Description

Commercial building represents an commercial building in the traffic simulation.

Commercial building is one type of building for the residents who are going to stores, gyms, restaurants etc. The capacity is randomized by the program so the user can't affect it.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 Commercial()

```
Commercial::Commercial (
            Position pos,
            City & city,
            int capacity )  [inline]
```

Construct a new Commercial building object.

Parameters

pos	Position of the building
city	Reference to the city
capacity	The maximum number of people in the building

3.5.3 Member Function Documentation

3.5.3.1 buildingType()

```
BuildingType Commercial::buildingType ( ) const  [inline], [virtual]
```

Checks the type of building.

Returns

BuildingType Commercial

Reimplemented from Building.

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

- src/Buildings/Commercial/Commercial.hpp

3.6 GUI Class Reference

Represents the GUI for interacting with the City.

`#include <GUI.hpp>`

Public Member Functions

- GUI (City &city, int size)
  Constructor for the GUI class.
- void init ()
  Initializes the window and all the elements needed for the GUI.
- void handleEvent ()
  Handles all the possible events in the GUI.
- void drawAndDisplay ()
  Draws and displays the GUI.
- bool isOpen ()
  Checks if the GUI window is open.
- bool isPaused ()
  Checks if the simulation is paused.
- bool hasCityChanged ()
  Checks if the city has changed since the last time the simulation paused.
- void setChanged (bool val)
  Sets the changed status of the city.
- Tool currentToolType ()
  Gets the current tool type selected in the GUI.
- void drawHistogram (std::vector< double > data, std::string yLabel, int type)
  Opens another window and draws a histogram on that window based on the data.
- void readCSV (const std::string &filename)
  Reads a City from a CSV file.
- void writeCSV (City &city, const std::string &filename)
  Writes the City to a CSV file.

3.6.1 Detailed Description

Represents the GUI for interacting with the City.

3.6.2 Constructor & Destructor Documentation

3.6.2.1 GUI()

```
GUI::GUI (
            City & city,
            int size )
```

Constructor for the GUI class.

Parameters

city	Reference to the City object.
size	Size parameter for the GUI representing the size of the City object.

3.6.3 Member Function Documentation

3.6.3.1 currentToolType()

```
Tool GUI::currentToolType ( )  [inline]
```

Gets the current tool type selected in the GUI.

Returns

The current tool type.

3.6.3.2 drawHistogram()

```
void GUI::drawHistogram (
            std::vector< double > data,
            std::string yLabel,
            int type )
```

Opens another window and draws a histogram on that window based on the data.

Parameters

data	Vector of data for the histogram.
yLabel	Label for the y-axis.
type	Type parameter for the histogram. 1 is for Road histogram, 2 is for City and 3 is for Intersection.

3.6.3.3 hasCityChanged()

```
bool GUI::hasCityChanged ( )  [inline]
```

Checks if the city has changed since the last time the simulation paused.

Returns

True if the city has changed, false otherwise.

3.6.3.4 isOpen()

```
bool GUI::isOpen ( )  [inline]
```

Checks if the GUI window is open.

Returns

True if the window is open, false otherwise.

3.6.3.5 isPaused()

```
bool GUI::isPaused ( )  [inline]
```

Checks if the simulation is paused.

Returns

True if paused, false otherwise.

3.6.3.6 readCSV()

```
void GUI::readCSV (
            const std::string & filename )
```

Reads a City from a CSV file.

Parameters

filename	Name of the CSV file to read. This is "City.csv" in the program.

3.6.3.7 setChanged()

```
void GUI::setChanged (
            bool val )  [inline]
```

Sets the changed status of the city.

Parameters

val	New value for the cityChanged flag.

3.6.3.8 writeCSV()

```
void GUI::writeCSV (
            City & city,
            const std::string & filename )
```

Writes the City to a CSV file.

Parameters

city	Reference to the City object.
filename	Name of the CSV file to write. This is "City.csv" in the program.

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

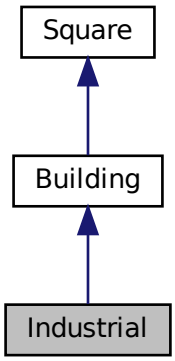
- src/GUI.hpp
- src/GUI.cpp

3.7 Industrial Class Reference

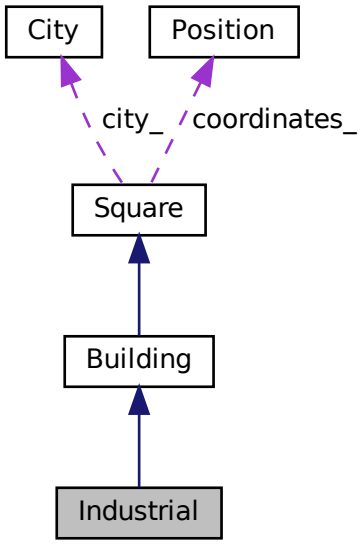
Industrial building represents an industrial building in the traffic simulation.

```
#include <Industrial.hpp>
```

Inheritance diagram for Industrial:



Collaboration diagram for Industrial:



Public Member Functions

- Industrial (Position pos, City &city, int capacity)
  Construct a new Industrial building object.
- BuildingType buildingType () const
  Checks the type of building.

Additional Inherited Members

3.7.1 Detailed Description

Industrial building represents an industrial building in the traffic simulation.

Industrial building is one type of building for the residents who are going to work. The capacity is randomized by the program so the user can't affect it.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 Industrial()

```
Industrial::Industrial (
            Position pos,
            City & city,
            int capacity )  [inline]
```

Construct a new Industrial building object.

Parameters

pos	Position of the building
city	Reference to the city
capacity	The maximum number of people in the building

3.7.3 Member Function Documentation

3.7.3.1 buildingType()

```
BuildingType Industrial::buildingType ( ) const  [inline], [virtual]
```

Checks the type of building.

Returns

BuildingType Industrial

Reimplemented from Building.

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

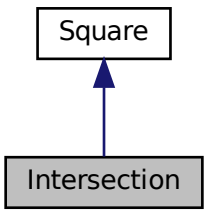
- src/Buildings/Industrial.hpp

3.8 Intersection Class Reference

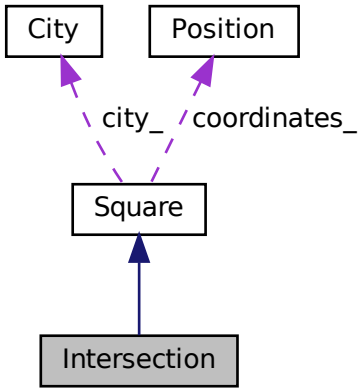
Intersection represents an intersection in the traffic simulation.

```
#include <Intersection.hpp>
```

Inheritance diagram for Intersection:



Collaboration diagram for Intersection:



Public Member Functions

- Intersection (Position pos, City &city, std::vector< Direction > dirs, int type=0)

  Construct a new Intersection object.
- ~Intersection ()=default

  Destroy the Intersection object.
- virtual void initSquare ()

  initSquare initializes the intersection
- int addCar (std::unique_ptr< Car > &&car)

  addCar adds a car to the intersection
- int joinCar (std::unique_ptr< Car > &&car)

  joinCar adds a car from a building to the road or intersection
- int addPed (std::unique_ptr< Resident > &&ped)

  addPed adds a pedestrian to the intersection
- virtual void simulate ()

  simulate simulates the intersection
- virtual int getCarCount () const

  getCarCount returns the number of cars in the intersection
- virtual int getPedCount () const

  getPedCount returns the ammount of pedestrians in the square
- bool isRoad () const

  isRoad returns true for intersections
- bool isIntersection () const

  isIntersection returns true for intersections
- int getType () const

  getType returns the type of the intersection
- std::vector< double > getStats () const

  getStats returns the statistics of the intersection
- int **maxCarAmount** () const

Additional Inherited Members

3.8.1 Detailed Description

Intersection represents an intersection in the traffic simulation.

Intersection contains output and input spots for cars and pedestrians and simulates them according to the traffic laws. They move the cars and pedestrians where they need to go.

3.8.2 Constructor & Destructor Documentation

3.8.2.1 Intersection()

```
Intersection::Intersection (
            Position pos,
            City & city,
            std::vector< Direction > dirs,
            int type = 0 )
```

Construct a new Intersection object.

Parameters

pos	Position of the intersection
city	Reference to the city
dirs	Directions of the roads
type	Type of the intersection

3.8.3 Member Function Documentation

3.8.3.1 addCar()

```
int Intersection::addCar (
            std::unique_ptr< Car > && car )  [virtual]
```

addCar adds a car to the intersection

The function adds a car to the intersection. The car is added to the input spot of the intersection depending on the direction of the car.

Parameters

car	the car to add

Returns

int 1 if succesfull, 0 if not

Reimplemented from Square.

3.8.3.2 addPed()

```
int Intersection::addPed (
            std::unique_ptr< Resident > && ped )  [virtual]
```

addPed adds a pedestrian to the intersection

The function adds a pedestrian to the intersection. The pedestrian is added to the input vector of the intersection depending on the direction of the pedestrian.

Parameters

ped	the pedestrian to add

Returns

int 1 if succesfull, 0 if not

Reimplemented from Square.

3.8.3.3 getCarCount()

```
int Intersection::getCarCount ( ) const  [virtual]
```

getCarCount returns the number of cars in the intersection

Returns

int number of cars in the intersection

Reimplemented from Square.

3.8.3.4 getPedCount()

```
int Intersection::getPedCount ( ) const  [virtual]
```

getPedCount returns the ammount of pedestrians in the square

Returns

the ammount of pedestrians in the square. Default implementation returns 0.

Reimplemented from Square.

3.8.3.5 getStats()

```
std::vector< double > Intersection::getStats ( ) const
```

getStats returns the statistics of the intersection

Returns

std::vector<double> statistics of the intersection. These are the average hourly number of cars passing through the intersection.

3.8.3.6 getType()

```
int Intersection::getType ( ) const  [inline]
```

getType returns the type of the intersection

Returns

int type of the intersection, 0 = equal, 1 = traffic light

3.8.3.7 isIntersection()

```
bool Intersection::isIntersection ( ) const  [inline], [virtual]
```

isIntersection returns true for intersections

Returns

true

Reimplemented from Square.

3.8.3.8 isRoad()

```
bool Intersection::isRoad ( ) const  [inline], [virtual]
```

isRoad returns true for intersections

Returns

true

Reimplemented from Square.

3.8.3.9 joinCar()

```
int Intersection::joinCar (
            std::unique_ptr< Car > && car )  [inline], [virtual]
```

joinCar adds a car from a building to the road or intersection

The function adds a car to the road. The car is added to the middle of the road, to simulate the car driving out of a building to the middle of the road.

Parameters

car	the car to add

Returns

1 if successfull, 0 if not (default is 0)

Reimplemented from Square.

3.8.3.10 simulate()

```
void Intersection::simulate ( )  [virtual]
```

simulate simulates the intersection

The function simulates the intersection by moving the cars in the intersection according to the traffic laws. The function also takes into account the reaction time of the drivers.

Reimplemented from Square.

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

- src/Roads/Intersection.hpp
- src/Roads/Intersection.cpp

3.9 IntersectionStats Struct Reference

Struct representing statistics for an intersection.

```
#include <Utilities.hpp>
```

Public Member Functions

- void incrementCounter (int index)
  Increments the counter for a specific hour.
- std::vector< double > getTotalCars () const
  Retrieves the total cars for each hour.

Public Attributes

- std::vector< double > **carCounters**
- int **overallCounter**

3.9.1 Detailed Description

Struct representing statistics for an intersection.

3.9.2 Member Function Documentation

3.9.2.1 getTotalCars()

```
std::vector<double> IntersectionStats::getTotalCars ( ) const  [inline]
```

Retrieves the total cars for each hour.

Returns

A vector of double representing the total cars for each hour.

3.9.2.2 incrementCounter()

```
void IntersectionStats::incrementCounter (
            int index )  [inline]
```

Increments the counter for a specific hour.

Parameters

index	Hour index.

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

- src/Utilities.hpp

3.10 Position Struct Reference

Struct representing a position with x and y coordinates.

`#include <Utilities.hpp>`

Public Member Functions

- **Position** (size_t xa, size_t ya)
- Position & **operator=** (const Position &other)
- bool **operator==** (const Position &a) const
- bool **operator!=** (const Position &a) const

Public Attributes

- size_t **x**
- size_t **y**

3.10.1 Detailed Description

Struct representing a position with x and y coordinates.

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

- src/Utilities.hpp

3.11 Resident Class Reference

Resident class represents a resident in the city.

`#include <Resident.hpp>`

Public Member Functions

- Resident (City &city, Position currentPos, int id)
  Constructor for the Resident class.
- ~Resident ()=default
  Destroy the Resident object.
- bool operator== (Resident &r)
  Overloaded equality operator for comparing residents based on their IDs.
- bool operator!= (Resident &r)
  Overloaded inequality operator for comparing residents based on their IDs.
- bool leave (const Position nextPos)
  Initiates the process of a resident leaving their current position for a new destination.
- int findPath ()
  Finds the shortest path from the current position to the destination using A∗ algorithm.
- void enter ()
  Resets the resident's movement state and clears the movement path.
- int getId () const
  Gets the unique identifier of the resident.
- const Position getPos () const
  Gets the current position of the resident.
- void simulate (Position cur)
  Simulates the daily routine and actions of the resident.
- std::string info () const
  Provides information about the resident.
- const Direction getDir () const
  Gets the current direction the resident is facing.
- Direction peekDir () const
  Peeks at the next direction in the resident's movement path.
- Direction updateDir ()
  Updates the direction the resident is facing during movement.

3.11.1 Detailed Description

Resident class represents a resident in the city.

The resident class contains the information about the resident and the functions that simulate the resident's actions in the simulation.

3.11.2 Constructor & Destructor Documentation

3.11.2.1 Resident()

```
Resident::Resident (
            City & city,
            Position currentPos,
            int id )
```

Constructor for the Resident class.

Parameters

city	Reference to the City object.
currentPos	Initial position of the resident.
id	Unique identifier for the resident.

3.11.3 Member Function Documentation

3.11.3.1 findPath()

```
int Resident::findPath ( )
```

Finds the shortest path from the current position to the destination using A∗ algorithm.

Returns

The length of the found path, or -1 if no path is found.

3.11.3.2 getDir()

```
const Direction Resident::getDir ( ) const
```

Gets the current direction the resident is facing.

Returns

The current direction of the resident.

3.11.3.3 getId()

```
int Resident::getId ( ) const  [inline]
```

Gets the unique identifier of the resident.

Returns

The ID of the resident.

3.11.3.4 getPos()

```
const Position Resident::getPos ( ) const  [inline]
```

Gets the current position of the resident.

Returns

The current position of the resident.

3.11.3.5 info()

```
std::string Resident::info ( ) const
```

Provides information about the resident.

Returns

A string containing information about the resident.

3.11.3.6 leave()

```
bool Resident::leave (
            const Position nextPos )
```

Initiates the process of a resident leaving their current position for a new destination.

Parameters

nextPos	The position to which the resident intends to move.

Returns

True if the resident successfully initiates the move, false otherwise.

3.11.3.7 operator"!=()

```
bool Resident::operator!= (
            Resident & r )
```

Overloaded inequality operator for comparing residents based on their IDs.

Parameters

r	Another Resident object for comparison.

Returns

True if the residents have different IDs, false otherwise.

3.11.3.8 operator==()

```
bool Resident::operator== (
            Resident & r )
```

Overloaded equality operator for comparing residents based on their IDs.

Parameters

r	Another Resident object for comparison.

Returns

True if the residents have the same ID, false otherwise.

3.11.3.9 peekDir()

```
Direction Resident::peekDir ( ) const
```

Peeks at the next direction in the resident's movement path.

Returns

The next direction in the movement path.

3.11.3.10 simulate()

```
void Resident::simulate (
            Position cur )
```

Simulates the daily routine and actions of the resident.

Parameters

cur	The current position of the resident.

3.11.3.11 updateDir()

```
Direction Resident::updateDir ( )
```

Updates the direction the resident is facing during movement.

Returns

The updated direction the resident is facing.

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

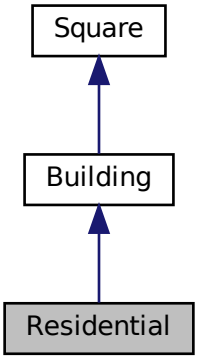
- src/Cars/Resident.hpp
- src/Cars/Resident.cpp

3.12 Residential Class Reference

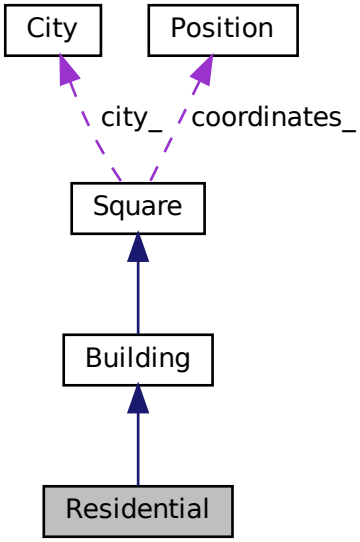
Residential building represents an residential building in the traffic simulation.

```
#include <Residential.hpp>
```

Inheritance diagram for Residential:



Collaboration diagram for Residential:



Public Member Functions

- Residential (Position pos, City &city, int capacity)

  Construct a new Residential building object.
- void initSquare ()

  Adds residents to the building.
- BuildingType buildingType () const

  Checks the type of building.

Additional Inherited Members

3.12.1 Detailed Description

Residential building represents an residential building in the traffic simulation.

Residential building is one type of building where the residents are implemented. The capacity is set to 100 by the program but user can also change it.

3.12.2 Constructor & Destructor Documentation

3.12.2.1 Residential()

```
Residential::Residential (
            Position pos,
            City & city,
            int capacity )  [inline]
```

Construct a new Residential building object.

Parameters

pos	Position of the building
city	Reference to the city
capacity	The maximum number of people in the building

3.12.3 Member Function Documentation

3.12.3.1 buildingType()

```
BuildingType Residential::buildingType ( ) const  [inline], [virtual]
```

Checks the type of building.

Returns

BuildingType Residential

Reimplemented from Building.

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

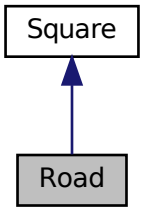
- src/Buildings/Residential.hpp
- src/Buildings/Residential.cpp

3.13 Road Class Reference

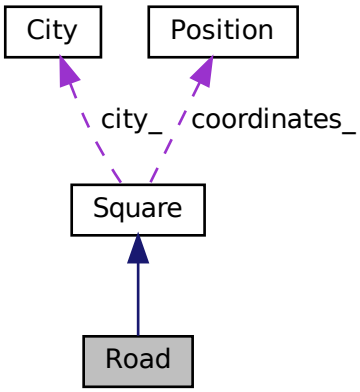
Road class represents a road in the traffic simulation.

`#include <Road.hpp>`

Inheritance diagram for Road:



Collaboration diagram for Road:



Public Member Functions

- Road (Position pos, City &city, int speed=0)

  Construct a new Road object.
- ~Road ()=default

  Destroy the Road object.
- virtual int addPed (std::unique_ptr< Resident > &&ped)

  addPed adds a pedestrian to the square
- virtual int addCar (std::unique_ptr< Car > &&car)

addCar adds a car to the square.

- int joinCar (std::unique_ptr< Car > &&car)

  joinCar adds a car from a building to the road or intersection
- virtual int getCarCount () const

  getCarCount returns the number of cars on the road
- virtual int getPedCount () const

  getPedCount returns the ammount of pedestrians in the square
- virtual int getNECarCount () const

  getNECarCount returns the number of cars on the road in the directions North and East
- virtual int getSWCarCount () const

  getSWCarCount returns the number of cars on the road in the directions South and West
- virtual void simulate ()

  simulate simulates the movement of the cars on the road
- virtual void initSquare ()

  initSquare initializes the vectors and maps of the road
- bool isRoad () const

  Checks if the Square object is a road.
- std::vector< double > getStats () const

  Retrieves the statistics from the object.
- bool isNS ()

  Checks if the road has a north-south orientation. This is needed for the heatmap.
- bool isEW ()

  Checks if the road has an east-west orientation.
- double averageTrafficPercent () const

  Calculates and returns the average traffic percentage on the road from all hours of the entire simulation period.
- int getSpeed ()

  Get the Speed object.

Additional Inherited Members

3.13.1 Detailed Description

Road class represents a road in the traffic simulation.

The road class contains the vectors that stores cars and pedestrians that are on the move. Roads move the cars and pedestrians in the simulation

3.13.2 Constructor & Destructor Documentation

3.13.2.1 Road()

```
Road::Road (
            Position pos,
            City & city,
            int speed = 0 )
```

Construct a new Road object.

Parameters

pos	Position of thD the city
speed	Speed of the cars on the road

3.13.3 Member Function Documentation

3.13.3.1 addCar()

```
int Road::addCar (
            std::unique_ptr< Car > && car )  [virtual]
```

addCar adds a car to the square.

Every subclass has its own implementation of the function. Default implementation does nothing and returns 0 for failure.

Parameters

car	the car to add

Returns

int 1 if succesfull, 0 if not. (Default 0)

Reimplemented from Square.

3.13.3.2 addPed()

```
int Road::addPed (
            std::unique_ptr< Resident > && ped )  [virtual]
```

addPed adds a pedestrian to the square

The function adds a pedestrian to the road. The pedestrian is added to the beginning of the road vector, which is defined by the direction of the pedestrian.

Parameters

ped	the pedestrian to add

Returns

int 1 if succesfull, 0 if not

Reimplemented from Square.

3.13.3.3 averageTrafficPercent()

```
double Road::averageTrafficPercent ( ) const
```

Calculates and returns the average traffic percentage on the road from all hours of the entire simulation period.

Returns

The average traffic percentage as a double.

3.13.3.4 getCarCount()

```
int Road::getCarCount ( ) const  [virtual]
```

getCarCount returns the number of cars on the road

Returns

int number of cars on the road

Reimplemented from Square.

3.13.3.5 getNECarCount()

```
int Road::getNECarCount ( ) const  [virtual]
```

getNECarCount returns the number of cars on the road in the directions North and East

Returns

int number of cars on the road in the directions North and East

3.13.3.6 getPedCount()

```
virtual int Road::getPedCount ( ) const  [inline], [virtual]
```

getPedCount returns the ammount of pedestrians in the square

Returns

the ammount of pedestrians in the square. Default implementation returns 0.

Reimplemented from Square.

3.13.3.7 getSpeed()

```
int Road::getSpeed ( )  [inline]
```

Get the Speed object.

Returns

int speed of the cars on the road

3.13.3.8 getStats()

```
std::vector< double > Road::getStats ( ) const
```

Retrieves the statistics from the object.

Returns

A vector of double representing the average hourly car counts on the road.

3.13.3.9 getSWCarCount()

```
int Road::getSWCarCount ( ) const  [virtual]
```

getSWCarCount returns the number of cars on the road in the directions South and West

Returns

int number of cars on the road in the directions South and West

3.13.3.10 initSquare()

```
void Road::initSquare ( )  [virtual]
```

initSquare initializes the vectors and maps of the road

The function initializes the vectors and maps of the road. The function is called when the simulation is reset.

Reimplemented from Square.

3.13.3.11 isEW()

```
bool Road::isEW ( )
```

Checks if the road has an east-west orientation.

Returns

True if the road is oriented east-west, false otherwise.

3.13.3.12 isNS()

```
bool Road::isNS ( )
```

Checks if the road has a north-south orientation. This is needed for the heatmap.

Returns

True if the road is oriented north-south, north-east, north-west, south-west or south-east. The heatmap shows all these variations as two up to down lanes.

3.13.3.13 isRoad()

```
bool Road::isRoad ( ) const  [inline], [virtual]
```

Checks if the Square object is a road.

Returns

True.

Reimplemented from Square.

3.13.3.14 joinCar()

```
int Road::joinCar (
            std::unique_ptr< Car > && car )  [virtual]
```

joinCar adds a car from a building to the road or intersection

The function adds a car to the road. The car is added to the middle of the road, to simulate the car driving out of a building to the middle of the road.

Parameters

car	the car to add

Returns

1 if successfull, 0 if not (default is 0)

Reimplemented from Square.

3.13.3.15 simulate()

```
void Road::simulate ( )  [virtual]
```

simulate simulates the movement of the cars on the road

The function simulates the movement of the cars on the road. The function moves the cars if there is space in front of them, and they have waited for the time that is defined by the speed of the road.

The function also calls the simuate functions for the Residents inside the cars.

Reimplemented from Square.

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

- src/Roads/Road.hpp
- src/Roads/Road.cpp

3.14 RoadStats Struct Reference

Struct representing stats for roads.

```
#include <Utilities.hpp>
```

Public Member Functions

- void incrementCounter (int index)

  Increments the counter for a specific hour and updates the overall counter.
- double getAverage () const

  Calculates and retrieves the average hour for cars.

Public Attributes

- std::vector< int > **carCounters**
- int **overallCounter**

3.14.1 Detailed Description

Struct representing stats for roads.

3.14.2 Member Function Documentation

3.14.2.1 getAverage()

```
double RoadStats::getAverage ( ) const  [inline]
```

Calculates and retrieves the average hour for cars.

Returns

The average hour for cars as a double.

3.14.2.2 incrementCounter()

```
void RoadStats::incrementCounter (
            int index )  [inline]
```

Increments the counter for a specific hour and updates the overall counter.

Parameters

index	Hour index.

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

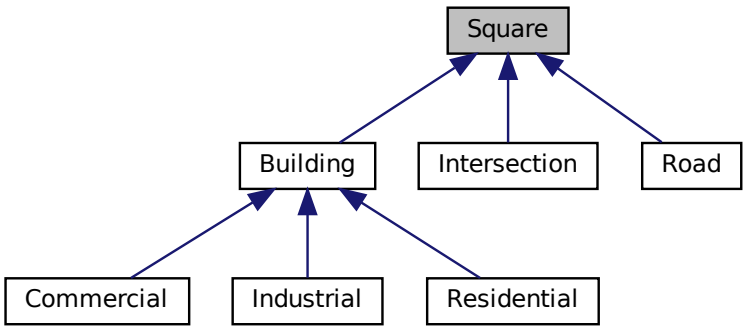
- src/Utilities.hpp

3.15 Square Class Reference

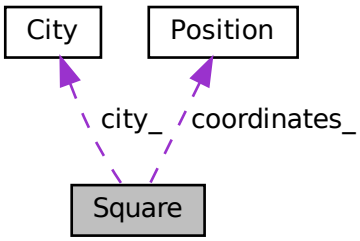
Square class is a building block of city, and might contain buildings or roads.

```
#include <Square.hpp>
```

Inheritance diagram for Square:



Collaboration diagram for Square:



Public Member Functions

- Square (Position pos, City &city)

  Construct a new Square object.
- ~Square ()=default

  Destroy the Square object.
- Position getCoordinates () const

  getCoordinates returns the coordinates of the square
- Square * getNeighbour (Direction dir)

  getNeighbour returns the neighbour of the square in the given direction
- virtual void simulate ()

  simulate simulates the functionalities of the square
- virtual int getCarCount () const

  getCarCount returns the ammount of cars in the square
- virtual int getPedCount () const

  getPedCount returns the ammount of pedestrians in the square

- virtual int addPed (std::unique_ptr< Resident > &&ped)

  addPed adds a pedestrian to the square
- virtual int addCar (std::unique_ptr< Car > &&car)

  addCar adds a car to the square.
- virtual int joinCar (std::unique_ptr< Car > &&car)

  joinCar adds a car from a building to the road or intersection
- Square & operator= (const Square &other)

  operator= copy assignment operator for square
- virtual bool isRoad () const

  isRoad returns is the square type road
- virtual bool isBuilding () const

  isBuilding returns is the square type building
- virtual BuildingType buildingType () const

  buildingType returns the building type in the square, Returns None if it is not building
- virtual bool isEmpty () const

  isEmpty returns is the square type empty
- virtual bool isIntersection () const

  isIntersection returns is the square type intersection
- virtual void initSquare ()

  initSquare initializes the square for simulation

Protected Attributes

- Position coordinates_

  coordinates_ The coordinates of the square
- City & city_

  city_ Reference to the city where the square resides

3.15.1 Detailed Description

Square class is a building block of city, and might contain buildings or roads.

Square is a building block of the city. It can contain buildings, roads or be empty. Empty squares simulate empty spaces in the city, and don't interract with other squares.

3.15.2 Constructor & Destructor Documentation

3.15.2.1 Square()

```
Square::Square (
            Position pos,
            City & city )  [inline]
```

Construct a new Square object.

Parameters

pos	Position of the square
city	Reference to the city

3.15.3 Member Function Documentation

3.15.3.1 addCar()

```
virtual int Square::addCar (
            std::unique_ptr< Car > && car )  [inline], [virtual]
```

addCar adds a car to the square.

Every subclass has its own implementation of the function. Default implementation does nothing and returns 0 for failure.

Parameters

car	the car to add

Returns

int 1 if succesfull, 0 if not. (Default 0)

Reimplemented in Road, Intersection, and Building.

3.15.3.2 addPed()

```
virtual int Square::addPed (
            std::unique_ptr< Resident > && ped )  [inline], [virtual]
```

addPed adds a pedestrian to the square

The function adds a pedestrian to the road. The pedestrian is added to the beginning of the road vector, which is defined by the direction of the pedestrian.

Parameters

ped	the pedestrian to add

Returns

int 1 if succesfull, 0 if not

Reimplemented in Road, Intersection, and Building.

3.15.3.3 getCarCount()

```
virtual int Square::getCarCount ( ) const  [inline], [virtual]
```

getCarCount returns the ammount of cars in the square

Returns

the ammount of cars in the square. Default implementation returns 0.

Reimplemented in Road, Intersection, and Building.

3.15.3.4 getCoordinates()

```
Position Square::getCoordinates ( ) const  [inline]
```

getCoordinates returns the coordinates of the square

Returns

Position coordinates of the square

3.15.3.5 getNeighbour()

```
Square * Square::getNeighbour (
            Direction dir )
```

getNeighbour returns the neighbour of the square in the given direction

Parameters

dir	Direction of the neighbour to get

Returns

Square* Pointer to the neighbour

3.15.3.6 getPedCount()

```
virtual int Square::getPedCount ( ) const  [inline], [virtual]
```

getPedCount returns the ammount of pedestrians in the square

Returns

the ammount of pedestrians in the square. Default implementation returns 0.

Reimplemented in Road, and Intersection.

3.15.3.7 joinCar()

```
virtual int Square::joinCar (
            std::unique_ptr< Car > && car )  [inline], [virtual]
```

joinCar adds a car from a building to the road or intersection

The function adds a car to the road. The car is added to the middle of the road, to simulate the car driving out of a building to the middle of the road.

Parameters

car	the car to add

Returns

1 if successfull, 0 if not (default is 0)

Reimplemented in Road, and Intersection.

3.15.3.8 operator=()

```
Square & Square::operator= (
            const Square & other )
```

operator= copy assignment operator for square

Parameters

other	the square to copy

Returns

Square& reference to the square

3.15.3.9 simulate()

```
virtual void Square::simulate ( )  [inline], [virtual]
```

simulate simulates the functionalities of the square

Each different square type has its own implementation of the function. Empty square does nothing.

Reimplemented in Road, Intersection, and Building.

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

- src/Square.hpp
- src/Square.cpp

Index