Traffic Simulator

Generated by Doxygen 1.9.1

Chapter 1

Main Page

1.1 Introduction

The purpose of this program is to simulate and analyze the traffic of a city model inputted as a JSON file.

This page contains instructions on how to use the program. The same information can also be found in the README file of the project.

1.2 Instructions on how to use the program

Dependencies: The simulator might require some additional libraries that are required to be installed through terminal, so run these commands:

- Linux: `sudo apt-get install libxrandr-dev libxcursor-dev libsfml-dev`
- MacOS: `brew install sfml`

Building and running the program:

- Clone `https://version.aalto.fi/gitlab/karhuj5/traffic-simulator-henrik-toikka-4.git`
- Navigate to the folder `cd build`
- Generate makefile with CMake: `cmake ..`
- Build the file in the same directory `make`
- Navigate to the root folder `cd ..`
- Run the file `build/./main`

How to use the simulation: In the beginning, the program asks for a JSON file name. If the file is not found in the root folder, or if it is invalid, the program will not proceed and it will ask for another file (see the JSON file template and requirements below). After the JSON file has been successfully loaded, the program asks for a road index to analyze during the simulation. Index i chooses the i:th road listed in the JSON file to be analyzed. After choosing the road to be analyzed, the program asks for simulation speed (1, 2, 4, 8, or 16). After this, the program asks if the gui should be enabled or not, and then it will start the simulation.

During the simulation you can use the commands:

- `status` Prints the current simulation time to the terminal
- `exit` Exits the program
- `analyze` Prints the usage data of the i:th road to the terminal
- `export` Exports the road analysis data to a CSV file

1.3 JSON file template for city loading

- The city is loaded from a JSON file which should be located in the root folder of the project. A template for the JSON files is provided below. If the JSON file is invalid, the program will continue asking for a new file until the file is valid. The root folder of this project contains two example JSON files: city.json and smallcity.json.
- Requirements:
- The order of the elements must be the same as in the template.
- All the locations of the buildings, intersections, roads, and traffic lights must be inside city boundaries.
- The building type must be on of the following types: "Residential", "Industrial", "Gym", "Shop", or "Restaurant".
- The starting and ending points of roads must be either intersections or buildings.
- The PersonType parameter of person objects must be one of the following types: "Lazy", "Active", "Neutral", "Gentleman", or "Angry".
- The workplace of a person must be one of the Industrial buildings added to the city (i.e. the name of the workplace must be found in the added Industrial buildings' names)
- The home of a person must be one of the Residential buildings added to the city (i.e. the name of the home building must be found in the added Residential buildings' names)
- The traffic lights must have the same coordinates as one of the intersections.
- The objects must be placed in locations where there are no other objects (excluding traffic lights, which must be placed on top of intersections).

The template:

```
{
  "x": vertical_city_size(int),
  "y": horizontal_city_size(int),
  "buildings": [
    {"name": ["building type", [x-coordinate(int), y-coordinate(int)]]}
  ],
  "intersections": [
    [x-coordinate(int), y-coordinate(int)]
  ],
  "roads": [
    [starting_x-coordinate(int), starting_y-coordinate(int), ending_x-coordinate(int),
       ending-y-coordinate(int)]
  ],
  "persons": [
    ["name", "PersonType", "name of workplace", "name of home building"]
  ],
  "trafficLights": [
    [[x-coordinate(int), y-coordinate(int)], red_and_green_light_duration(int), yellow_light_duration(int)]
  ]
}
```

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

Class Documentation

4.1 Analysis Class Reference

Class used for analyzing the traffic data provided by the simulator.

`#include <analysis.hpp>`

Public Member Functions

- Analysis (City *city, SimulationClock *clock)
  Construct a new Analysis object.
- void Analyze ()
  The main function of this class, which is used to obtain the analysis data.
- void SpecifyRoad (int roadIndex)
  Specifies the road that is going to be analyzed.
- void GenerateHourlyHistogram (std::vector< std::vector< int >> data)
  Prints out a histogram of the current day with hourly statistics of road usage.
- void ExportToCSV (const std::string &filename)
  Exports the analysis data into a CSV file in the root folder of the project.
- std::vector< std::vector< int > > GetData ()
  Returns the hourly road data.
- void Init ()
  Initializes the previousCars_ private variable.

4.1.1 Detailed Description

Class used for analyzing the traffic data provided by the simulator.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Analysis()

```
Analysis::Analysis (
            City * city,
            SimulationClock * clock )
```

Construct a new Analysis object.

Parameters

city	Pointer to the main city object
clock	Pointer to the main clock object

4.1.3 Member Function Documentation

4.1.3.1 ExportToCSV()

```
void Analysis::ExportToCSV (
            const std::string & filename )
```

Exports the analysis data into a CSV file in the root folder of the project.

Parameters

data	Road usage data that is provided by the GetData function in this class

4.1.3.2 GenerateHourlyHistogram()

```
void Analysis::GenerateHourlyHistogram (
            std::vector< std::vector< int >> data )
```

Prints out a histogram of the current day with hourly statistics of road usage.

Parameters

data	Road usage data that is provided by the GetData function in this class

4.1.3.3 GetData()

```
std::vector< std::vector< int > > Analysis::GetData ( )
```

Returns the hourly road data.

Returns

The hourly road data

4.1.3.4 SpecifyRoad()

```
void Analysis::SpecifyRoad (
            int roadIndex )
```

Specifies the road that is going to be analyzed.

Parameters

roadIndex	Index of the road that is going to be analyzed

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

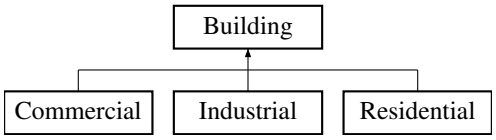
- src/analysis.hpp
- src/analysis.cpp

4.2 Building Class Reference

An abstract class for representing the buildings in the simulated city. Each building is either a commercial, a residential, or an industrial building.

```
#include <building.hpp>
```

Inheritance diagram for Building:



Public Member Functions

- Building (const std::string &buildingName, const std::pair< int, int > &location, const std::string &type)

  Construct a new Building object.
- virtual ~Building ()

  Destroy the Building object.
- const std::string & GetName () const

  Get the name of the building.
- virtual void Draw (sf::RenderWindow &window, int cellSize)=0

  Draws the building to the SFML window given as a parameter. Each subclass defines the draw function themselves.
- const std::pair< int, int > & GetLocation () const

  Get the location of the building.
- virtual std::string GetType () const =0

  Get type of building.

Protected Attributes

- std::string **buildingName_**
- std::pair< int, int > **location_**
- int **personAmount_**
- int **carAmount_**
- int **maxPersonCapacity_**
- int **maxCarCapacity_**
- std::string **type_**

4.2.1 Detailed Description

An abstract class for representing the buildings in the simulated city. Each building is either a commercial, a residential, or an industrial building.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Building()

```
Building::Building (
            const std::string & buildingName,
            const std::pair< int, int > & location,
            const std::string & type )
```

Construct a new Building object.

Parameters

buildingName	The name of the building
location	The coordinates of the building
type	The type of the building ("industrial", "residential", "shop", "gym", or "restaurant")

4.2.3 Member Function Documentation

4.2.3.1 Draw()

```
virtual void Building::Draw (
            sf::RenderWindow & window,
            int cellSize )  [pure virtual]
```

Draws the building to the SFML window given as a parameter. Each subclass defines the draw function themselves.

Parameters

window	A reference to the SFML window where the building should be drawn
cellSize	The size of the cell to be drawn

Implemented in Residential, Industrial, and Commercial.

4.2.3.2 GetLocation()

```
const std::pair< int, int > & Building::GetLocation ( ) const
```

Get the location of the building.

Returns

The coordinates of the building as std::pair<int, int>&

4.2.3.3 GetName()

```
const std::string & Building::GetName ( ) const
```

Get the name of the building.

Returns

The name of the building as std::string&

4.2.3.4 GetType()

```
virtual std::string Building::GetType ( ) const  [pure virtual]
```

Get type of building.

Returns

The type as a string.

Implemented in Residential, Industrial, and Commercial.

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

- src/building.hpp
- src/building.cpp

4.3 Car Class Reference

A class used for representing a car in the simulated city. The cars move around the city to varius destinations during the day.

`#include <car.hpp>`

Public Member Functions

- Car (Node *startingNode)

  Constructor. Constructs a new car object.
- void Update (float deltaTime, float currentTime, std::vector< Node * > allNodes, std::vector< Intersection * > intersections, std::vector< Car * > cars, std::vector< Road * > roads)

  Updates the cars location and state. Checks the cars schedule to find the next destination.
- Intersection * GetIntersection (std::vector< Intersection * > intersections)

  Used to return the intersection at the vicinity of the car.
- void SetDestination (Node *destination)

  Set a new destination for the car.
- void SetDirection (std::pair< int, int > current, std::pair< int, int > destination)

  Set the direction for the car.
- void SetSpeedLimit (std::vector< Road * > roads)

  Set the speedlimit for the car according to the current road.
- bool AtDestination (float destinationX, float destinationY)

  Check if the car is at destination.
- bool CheckIntersection (Intersection *intersection, std::vector< Car * > cars)

  Check that the intersection is ok and car can proceed.
- bool LaneIsFree (Intersection *intersection, std::vector< Car * > cars, std::string nextDirection)

  Check that the lane that the car will go to after the intersection in free.
- bool YieldRight (Intersection *intersection, std::vector< Car * > cars)

  If intersection does not have traffic lights cars should yield to other cars coming from right.
- std::string & GetDirection ()

  Get the direction of the car.
- void Draw (sf::RenderWindow &window, int cellSize)

  Draw the car into the SFML-window.
- void AddEvent (int time, Node *node)

  Add an event to the cars schedule.
- void InitializeSchedule (std::map< int, Node * > schedule)

  Initialize the cars schedule.
- bool CarInFront (std::vector< Car * > cars)

  Check if there is a car in front of this car.
- void SetColor (PersonType pType)

  Set the color for the car according to the PersonType.
- std::pair< float, float > GetLocation ()

  Get the location of the car.
- std::vector< Node * > Dijkstra (Node *source, Node *destination, std::vector< Node * > allNodes)

  An algorithm to calculate the shortest route from source node to destination node.

4.3.1 Detailed Description

A class used for representing a car in the simulated city. The cars move around the city to varius destinations during the day.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 Car()

```
Car::Car (
            Node * startingNode )
```

Constructor. Constructs a new car object.

Parameters

startingNode	The Node where the car starts at is the home of its owner by default.

4.3.3 Member Function Documentation

4.3.3.1 AddEvent()

```
void Car::AddEvent (
            int time,
            Node * node )
```

Add an event to the cars schedule.

Parameters

time	What time should the car move to the destination
node	The node that the car should move to

4.3.3.2 AtDestination()

```
bool Car::AtDestination (
            float destinationX,
            float destinationY )
```

Check if the car is at destination.

Parameters

destinationX	The x-coordinate of the destination
destinationY	The y-coordinate of the destination

Returns

True if the car is at destination, false otherwise

4.3.3.3 CarInFront()

```
bool Car::CarInFront (
            std::vector< Car * > cars )
```

Check if there is a car in front of this car.

Parameters

cars	The cars in the city

4.3.3.4 CheckIntersection()

```
bool Car::CheckIntersection (
            Intersection * intersection,
            std::vector< Car * > cars )
```

Check that the intersection is ok and car can proceed.

Parameters

intersection	The intersection in front of the car
cars	All the cars in the city

Returns

True if the car can proceed

4.3.3.5 Dijkstra()

```
std::vector< Node * > Car::Dijkstra (
            Node * source,
            Node * destination,
            std::vector< Node * > allNodes )
```

An algorithm to calculate the shortest route from source node to destination node.

Parameters

source	The node where the car starts
destination	The node where the car should move next
allNodes	All the nodes in the city

4.3.3.6 Draw()

```
void Car::Draw (
            sf::RenderWindow & window,
            int cellSize )
```

Draw the car into the SFML-window.

Parameters

window	The SFML-window
cellSize	The size of the cell in the GUI

4.3.3.7 GetDirection()

```
std::string & Car::GetDirection ( )
```

Get the direction of the car.

Returns

The direction that the car is going to

4.3.3.8 GetIntersection()

```
Intersection * Car::GetIntersection (
            std::vector< Intersection * > intersections )
```

Used to return the intersection at the vicinity of the car.

Parameters

intersections	All the intersections in the city.

Returns

Intersection near the cars location

4.3.3.9 InitializeSchedule()

```
void Car::InitializeSchedule (
            std::map< int, Node * > schedule )
```

Initialize the cars schedule.

Parameters

schedule	The schedule for the car

4.3.3.10 LaneIsFree()

```
bool Car::LaneIsFree (
            Intersection * intersection,
            std::vector< Car * > cars,
            std::string nextDirection )
```

Check that the lane that the car will go to after the intersection in free.

Parameters

intersection	The intersection in front of the car
cars	The cars in the city
nextDirection	The direction that the car will go to after the intersection

Returns

True if the lane is free and false otherwise

4.3.3.11 SetColor()

```
void Car::SetColor (
            PersonType pType )
```

Set the color for the car according to the PersonType.

Parameters

pType	the PersonType of the car's owner

4.3.3.12 SetDestination()

```
void Car::SetDestination (
            Node * destination )
```

Set a new destination for the car.

Parameters

destination	New destination Node.

4.3.3.13 SetDirection()

```
void Car::SetDirection (
            std::pair< int, int > current,
            std::pair< int, int > destination )
```

Set the direction for the car.

Parameters

current	Current location of the car.
destination	The location of the destination.

4.3.3.14 SetSpeedLimit()

```
void Car::SetSpeedLimit (
            std::vector< Road * > roads )
```

Set the speedlimit for the car according to the current road.

Parameters

roads	All the roads in the city

4.3.3.15 Update()

```
void Car::Update (
            float deltaTime,
            float currentTime,
            std::vector< Node * > allNodes,
            std::vector< Intersection * > intersections,
            std::vector< Car * > cars,
            std::vector< Road * > roads )
```

Updates the cars location and state. Checks the cars schedule to find the next destination.

Parameters

deltaTime	How much time has passed since the car was last updated.
currentTime	How much time has passed in the simulation in total. Used to check the schedule.
allNodes	All the nodes currently in the city.
intersections	All the intersections currently in the city.
cars	All the cars currently in the city.
roads	All the roads currently in the city.

4.3.3.16 YieldRight()

```
bool Car::YieldRight (
            Intersection * intersection,
            std::vector< Car * > cars )
```

If intersection does not have traffic lights cars should yield to other cars coming from right.

Parameters

intersection	The intersection in front of the car
cars	All the cars in the city

Returns

True if the car should yield

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

- src/car.hpp
- src/car.cpp

4.4 Cell Class Reference

A class for representing a single cell in the graphical simulation.

`#include <cell.hpp>`

Public Member Functions

- Cell (std::pair< int, int > location)

  Construct a new Cell object.
- bool IsOccupied () const

  Returns a boolean value indicating if the cell is occupied or not.
- void Occupy (std::string type)

  Occupies the cell with an object of the type which is given as a parameter.
- void Clear ()

  Empties the cell (i.e. sets the value of the type_ variable to "Empty").
- std::string GetType () const

  Get the type of the cell.
- int GetX () const

  Get the x-coordinate of the cell.
- int GetY () const

  Get the y-coordinate of the cell.
- void Draw (sf::RenderWindow &window, int cellSize, int x, int y)

  Draws the cell to the SFML window given as a parameter.

4.4.1 Detailed Description

A class for representing a single cell in the graphical simulation.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 Cell()

```
Cell::Cell (
            std::pair< int, int > location )
```

Construct a new Cell object.

Parameters

location	the coordinates of the cell

4.4.3 Member Function Documentation

4.4.3.1 Draw()

```
void Cell::Draw (
            sf::RenderWindow & window,
```

```
            int cellSize,
            int x,
            int y )
```

Draws the cell to the SFML window given as a parameter.

Parameters

window	A reference to the SFML window where the cell should be drawn
cellSize	The size of the cell to be drawn
x	The x-coordinate of the cell to be drawn
y	The y-coordinate of the cell to be drawn

4.4.3.2 GetType()

```
std::string Cell::GetType ( ) const
```

Get the type of the cell.

Returns

The type of the cell as std::string

4.4.3.3 GetX()

```
int Cell::GetX ( ) const
```

Get the x-coordinate of the cell.

Returns

The x-coordinate as an integer

4.4.3.4 GetY()

```
int Cell::GetY ( ) const
```

Get the y-coordinate of the cell.

Returns

The y-coordinate as an integer

4.4.3.5 IsOccupied()

```
bool Cell::IsOccupied ( ) const
```

Returns a boolean value indicating if the cell is occupied or not.

Returns

True if the cell is occupied, false otherwise

4.4.3.6 Occupy()

```
void Cell::Occupy (
            std::string type )
```

Occupies the cell with an object of the type which is given as a parameter.

Parameters

type	The type of the object being placed in the cell ("Horizontal road", "Vertical road", "Intersection", "industrial", "residential", "shop", "gym", or "restaurant")

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

- src/cell.hpp
- src/cell.cpp

4.5 City Class Reference

A class used for keeping track of the different elements (e.g buildings, roads, intersectons, etc.) in the simulated city.

```
#include <city.hpp>
```

Public Member Functions

- City (int sizeX, int sizeY)

  Construct a new City object.
- ~City ()

  Destroy the City object.
- bool IsValidRoad (std::pair< int, int > start, std::pair< int, int > end) const

  Checks if the road with the given start and end coordinates is a valid road. Requirements:
- void AddRoad (std::pair< int, int > start, std::pair< int, int > end)

  Adds a road to the city. Throws an InvalidCityException if the road is invalid.
- bool IsValidBuilding (Building *b) const

Checks if the building that the pointer given as parameter is pointing to is valid. The building must be inside city boundaries and the location of the building must be unoccupied.

- void AddBuilding (std::string name, std::pair< int, int > location, const std::string &buildingType)

  Adds a building to the city if it is valid. Otherwise throws an InvalidCityException.
- Node * GetNode (std::pair< int, int > location) const

  Get a pointer to the Node object in the given location.
- std::vector< Node * > GetBuildingNodes () const

  Get a list of the Node pointers that have type "Building".
- void AddPersonAndCar (std::string &name, PersonType personType, std::string &workplacename, std::string &homename)

  Adds a person and its car to the city.
- bool IsValidPerson (Person *p)

  Checks if the person that is trying to be added is valid.
- bool IsValidIntersection (Intersection *i) const

  Checks if the intersection that the pointer given as parameter is pointing to is valid. The intersection must be inside city boundaries and the location of the intersection must be unoccupied.
- void AddIntersection (std::pair< int, int > location)

  Attempts to create an intersection to the given location. Returns an InvalidCityException if the location is occupied already.
- Intersection * GetIntersection (std::pair< int, int > location) const

  Get the a pointer to the Intersection object in the given location.
- void UpdateIntersections (float deltaTime) const

  Updates the status of all the traffic lights in the city.
- void DrawIntersections (sf::RenderWindow &window) const

  Draws the intersections in the city to the SFML window given as parameter.
- void ChooseRoad (int roadIndex)

  Chooses the road that the user wants to analyze and highlights it.
- void AddTrafficLight (std::pair< int, int > location, int redAndGreenDuration, int yellowDuration)

  Adds a traffic light to the city.
- void UpdateCars (float deltaTime, float currentTime)

  Updates the locations and destinations of all the cars in the city.
- void DrawCars (sf::RenderWindow &window) const

  Draws the cars in the city to the SFML window given as parameter.
- void DrawBuildings (sf::RenderWindow &window) const

  Draws the buildings in the city to the SFML window given as parameter.
- void PrintCity (sf::RenderWindow &window) const

  Draws the city to the SFML window given as parameter.
- Grid * GetGrid () const

  Get a pointer to the Grid object of the city.
- std::vector< Road * > GetRoads () const

  Get a vector containing pointers to all the roads in the city.
- std::vector< Car * > GetCars () const

  Get a vector containing pointers to all the cars in the city.
- int TimeUntilNextEvent (Person *p) const

  Get time until next event.
- void AddClock (SimulationClock *clock)

  Add the simulation clock to city.
- bool IsBusy (Person *p) const

  Check if person has an event going on.
- void AddEvent (Person *p)

  Add event to schedule.

4.5.1 Detailed Description

A class used for keeping track of the different elements (e.g buildings, roads, intersectons, etc.) in the simulated city.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 City()

```
City::City (
            int sizeX,
            int sizeY )
```

Construct a new City object.

Parameters

sizeX	The vertical size of the city
sizeY	The horizontal size of the city

4.5.3 Member Function Documentation

4.5.3.1 AddBuilding()

```
void City::AddBuilding (
            std::string name,
            std::pair< int, int > location,
            const std::string & buildingType )
```

Adds a building to the city if it is valid. Otherwise throws an InvalidCityException.

Parameters

name	The name of the building
location	The coordinates of the building
buildingType	The type of the building (should be: industrial, residential, shop, gym, or restaurant)

4.5.3.2 AddClock()

```
void City::AddClock (
```

```
            SimulationClock * clock )
```

Add the simulation clock to city.

Parameters

clock	Clock implemented in SimulationClock, used in simulator.cpp.

4.5.3.3 AddEvent()

```
void City::AddEvent (
            Person * p )
```

Add event to schedule.

Parameters

p	This is the person that is going to have new event.

4.5.3.4 AddIntersection()

```
void City::AddIntersection (
            std::pair< int, int > location )
```

Attempts to create an intersection to the given location. Returns an InvalidCityException if the location is occupied already.

Parameters

location	The coordinates of the intersection

4.5.3.5 AddPersonAndCar()

```
void City::AddPersonAndCar (
            std::string & name,
            PersonType personType,
            std::string & workplacename,
            std::string & homename )
```

Adds a person and its car to the city.

Parameters

name	Name of the person.
personType	Type of the person.
workplacename	Name of working place.
homename	Name of home.

4.5.3.6 AddRoad()

```
void City::AddRoad (
            std::pair< int, int > start,
            std::pair< int, int > end )
```

Adds a road to the city. Throws an InvalidCityException if the road is invalid.

Parameters

start	The start coordinates of the road
end	The end coordinates of the road

4.5.3.7 AddTrafficLight()

```
void City::AddTrafficLight (
            std::pair< int, int > location,
            int redAndGreenDuration,
            int yellowDuration )
```

Adds a traffic light to the city.

Parameters

t	A pointer to the traffic light object to be added

4.5.3.8 ChooseRoad()

```
void City::ChooseRoad (
            int roadIndex )
```

Chooses the road that the user wants to analyze and highlights it.

Parameters

roadIndex	The index of the chosen road

4.5.3.9 DrawBuildings()

```
void City::DrawBuildings (
            sf::RenderWindow & window ) const
```

Draws the buildings in the city to the SFML window given as parameter.

Parameters

window	A reference to an SFML window

4.5.3.10 DrawCars()

```
void City::DrawCars (
            sf::RenderWindow & window ) const
```

Draws the cars in the city to the SFML window given as parameter.

Parameters

window	A reference to an SFML window

4.5.3.11 DrawIntersections()

```
void City::DrawIntersections (
            sf::RenderWindow & window ) const
```

Draws the intersections in the city to the SFML window given as parameter.

Parameters

window	A reference to a SFML window

4.5.3.12 GetBuildingNodes()

```
std::vector< Node * > City::GetBuildingNodes ( ) const
```

Get a list of the Node pointers that have type "Building".

Returns

A list of pointers to Nodes with type "Building".

4.5.3.13 GetCars()

```
std::vector< Car * > City::GetCars ( ) const
```

Get a vector containing pointers to all the cars in the city.

Returns

std::vector<Car*>

4.5.3.14 GetGrid()

```
Grid * City::GetGrid ( ) const
```

Get a pointer to the Grid object of the city.

Returns

Grid*

4.5.3.15 GetIntersection()

```
Intersection * City::GetIntersection (
            std::pair< int, int > location ) const
```

Get the a pointer to the Intersection object in the given location.

Parameters

location	The location of the intersection

Returns

A pointer to an Intersection object if there is an Intersection in the given location, or a nullptr if there is no Intersection in the given location.

4.5.3.16 GetNode()

```
Node * City::GetNode (
            std::pair< int, int > location ) const
```

Get a pointer to the Node object in the given location.

Parameters

location	The location of the node

Returns

A pointer to a Node object if there is a Node in the given location or a nullptr otherwise

4.5.3.17 GetRoads()

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

Get a vector containing pointers to all the roads in the city.

Returns

std::vector<Road*>

4.5.3.18 IsBusy()

```
bool City::IsBusy (
            Person * p ) const
```

Check if person has an event going on.

Parameters

p	Check if this exact person is busy.

Returns

Boolean value if a person is busy at the moment.

4.5.3.19 IsValidBuilding()

```
bool City::IsValidBuilding (
            Building * b ) const
```

Checks if the building that the pointer given as parameter is pointing to is valid. The building must be inside city boundaries and the location of the building must be unoccupied.

Parameters

b	A pointer to a building object

Returns

true if the building is valid, false otherwise

4.5.3.20 IsValidIntersection()

```
bool City::IsValidIntersection (
            Intersection * i ) const
```

Checks if the intersection that the pointer given as parameter is pointing to is valid. The intersection must be inside city boundaries and the location of the intersection must be unoccupied.

Parameters

i	A pointer to a intersection object

Returns

true if the intersection is valid, false otherwise

4.5.3.21 IsValidPerson()

```
bool City::IsValidPerson (
            Person * p )
```

Checks if the person that is trying to be added is valid.

Parameters

p	A pointer to the Person object

4.5.3.22 IsValidRoad()

```
bool City::IsValidRoad (
            std::pair< int, int > start,
            std::pair< int, int > end ) const
```

Checks if the road with the given start and end coordinates is a valid road. Requirements:

- the road must be inside the boundaries of the city
- the length of the road must be more than zero
- the road can only be horizontal or vertical (not diagonal)
- there cannot be any other roads or buildings between the start and end coordinates of the road

Parameters

start	The start coordinates of the road
end	The end coordinates of the road

Returns

true if the road is valid, false otherwise

4.5.3.23 PrintCity()

```
void City::PrintCity (
            sf::RenderWindow & window ) const
```

Draws the city to the SFML window given as parameter.

Parameters

window	A reference to an SFML window

4.5.3.24 TimeUntilNextEvent()

```
int City::TimeUntilNextEvent (
```

```
            Person * p ) const
```

Get time until next event.

Parameters

p	Next event from a specified person.

4.5.3.25 UpdateCars()

```
void City::UpdateCars (
            float deltaTime,
            float currentTime )
```

Updates the locations and destinations of all the cars in the city.

Parameters

deltaTime	
currentTime	

4.5.3.26 UpdateIntersections()

```
void City::UpdateIntersections (
            float deltaTime ) const
```

Updates the status of all the traffic lights in the city.

Parameters

deltaTime	

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

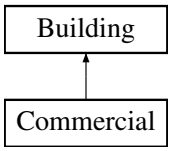
- src/city.hpp
- src/city.cpp

4.6 Commercial Class Reference

One of the concretizing classes for the abstract class Building.

```
#include <commercial.hpp>
```

Inheritance diagram for Commercial:



Public Member Functions

- Commercial (const std::string &buildingName, const std::pair< int, int > &location, const std::string &commercialType)

  Construct a new Commercial object.
- virtual void Draw (sf::RenderWindow &window, int cellSize) override

  Draws the building to the SFML window given as a parameter. Each subclass defines the draw function themselves.
- std::string GetType () const override

  Prints information about the commercial building to the standard output.

Additional Inherited Members

4.6.1 Detailed Description

One of the concretizing classes for the abstract class Building.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 Commercial()

```
Commercial::Commercial (
            const std::string & buildingName,
            const std::pair< int, int > & location,
            const std::string & commercialType )
```

Construct a new Commercial object.

Parameters

buildingName	The name of the commercial building
location	The coordinates of the commercial building
commercialType	The type of the commercial building ("gym", "restaurant", or "shop")

4.6.3 Member Function Documentation

4.6.3.1 Draw()

```
void Commercial::Draw (
            sf::RenderWindow & window,
            int cellSize )  [override], [virtual]
```

Draws the building to the SFML window given as a parameter. Each subclass defines the draw function themselves.

Parameters

window	A reference to the SFML window where the building should be drawn
cellSize	The size of the cell to be drawn

Implements Building.

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

- src/commercial.hpp
- src/commercial.cpp

4.7 Event Class Reference

The Event class is responsible for creating random schedules that determine the events a person will have over a period of time. It considers the person's type and adjusts the schedule accordingly.

```
#include <event.hpp>
```

Public Member Functions

- Event (Person *person, std::vector< Node * > buildingNodes)

  Construct a new Event object.
- std::map< int, Node * > CreateSchedule ()

  Create a new random schedule for a person.

4.7.1 Detailed Description

The Event class is responsible for creating random schedules that determine the events a person will have over a period of time. It considers the person's type and adjusts the schedule accordingly.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 Event()

```
Event::Event (
            Person * person,
            std::vector< Node * > buildingNodes )
```

Construct a new Event object.

Parameters

person	The person whose schedule is created.
buildingNodes	The nodes that contain a building.

4.7.3 Member Function Documentation

4.7.3.1 CreateSchedule()

```
std::map< int, Node * > Event::CreateSchedule ( )
```

Create a new random schedule for a person.

Returns

Returns the schedule in map format, where a time value is mapped to a corresponding event node.

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

- src/event.hpp
- src/event.cpp

4.8 Grid Class Reference

A class for storing all the cells of the graphical simulation.

```
#include <grid.hpp>
```

Public Member Functions

- Grid (int xSize, int ySize)

  Construct a new Grid object.
- ~Grid ()

  Destroy the Grid object.
- Cell * GetCell (int x, int y)

  Get a pointer to the cell at the given location.
- std::vector< Cell * > GetNeighborCells (int x, int y)

  Get the neighbouring cells of the cell in the given location.
- int GetSizeX () const

  Get the vertical size of the grid.
- int GetSizeY () const

  Get the horizontal size of the grid.

4.8.1 Detailed Description

A class for storing all the cells of the graphical simulation.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 Grid()

```
Grid::Grid (
            int xSize,
            int ySize )
```

Construct a new Grid object.

Parameters

xSize	The vertical size of the grid
ySize	The horizontal size of the grid

4.8.3 Member Function Documentation

4.8.3.1 GetCell()

```
Cell * Grid::GetCell (
            int x,
            int y )
```

Get a pointer to the cell at the given location.

Parameters

x	The x-coordinate of the cell
y	The y-coordinate of the cell

Returns

A pointer to a Cell object if there is one in the given location, a nullptr otherwise

4.8.3.2 GetNeighborCells()

```
std::vector< Cell * > Grid::GetNeighborCells (
            int x,
            int y )
```

Get the neighbouring cells of the cell in the given location.

Parameters

x	The x-coordinate of the cell
y	The y-coordinate of the cell

Returns

A vector containing pointers to all the neighbouring cells

4.8.3.3 GetSizeX()

```
int Grid::GetSizeX ( ) const
```

Get the vertical size of the grid.

Returns

The vertical size of the grid as an integer

4.8.3.4 GetSizeY()

```
int Grid::GetSizeY ( ) const
```

Get the horizontal size of the grid.

Returns

The horizontal size of the grid as an integer

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

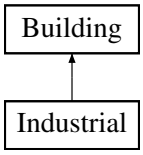
- src/grid.hpp
- src/grid.cpp

4.9 Industrial Class Reference

One of the concretizing classes for the abstract class Building.

`#include <industrial.hpp>`

Inheritance diagram for Industrial:



Public Member Functions

- Industrial (const std::string &buildingName, const std::pair< int, int > &location)

  Construct a new Industrial object.
- virtual void Draw (sf::RenderWindow &window, int cellSize) override

  Draws the building to the SFML window given as a parameter. Each subclass defines the draw function themselves.
- std::string GetType () const override

  Prints information about the industrial building to the standard output.

Additional Inherited Members

4.9.1 Detailed Description

One of the concretizing classes for the abstract class Building.

4.9.2 Constructor & Destructor Documentation

4.9.2.1 Industrial()

```
Industrial::Industrial (
            const std::string & buildingName,
            const std::pair< int, int > & location )  [inline]
```

Construct a new Industrial object.

Parameters

buildingName	The name of the industrial building
location	The coordinates of the idnustrial building

4.9.3 Member Function Documentation

4.9.3.1 Draw()

```
virtual void Industrial::Draw (
            sf::RenderWindow & window,
            int cellSize )  [inline], [override], [virtual]
```

Draws the building to the SFML window given as a parameter. Each subclass defines the draw function themselves.

Parameters

window	A reference to the SFML window where the building should be drawn
cellSize	The size of the cell to be drawn

Implements Building.

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

- src/industrial.hpp

4.10 Intersection Class Reference

A class used for representing an intersection in the simulated city.

`#include <intersection.hpp>`

Public Member Functions

- Intersection (std::pair< int, int > location)

  Construct a new Intersection object.
- std::pair< int, int > GetLocation () const

  Get the location of the intersection.
- void Update (float deltaTime)

  Attempts to update the status of the traffic lights in the intersection, if there are any.
- void AddTrafficLight (TrafficLight *t)

  Adds a traffic light to the intersection.
- bool HasTrafficLight () const

  Check if the intersection has a traffic light.
- bool AllowVertical ()

  Returns a boolean value indicating if cars are currently allowed to move vertically through the intersection.
- bool AllowHorizontal ()

  Returns a boolean value indicating if cars are currently allowed to move horizontally through the intersection.
- void Draw (sf::RenderWindow &window, int cellSize, Grid *grid)

  Draws the intersection into the SFML window given as parameter.

4.10.1 Detailed Description

A class used for representing an intersection in the simulated city.

4.10.2 Constructor & Destructor Documentation

4.10.2.1 Intersection()

```
Intersection::Intersection (
            std::pair< int, int > location )
```

Construct a new Intersection object.

Parameters

location	the coordinates of the intersection as std::pair<int, int>

4.10.3 Member Function Documentation

4.10.3.1 AddTrafficLight()

```
void Intersection::AddTrafficLight (
            TrafficLight * t )
```

Adds a traffic light to the intersection.

Parameters

t	A pointer to a TrafficLight object

4.10.3.2 AllowHorizontal()

```
bool Intersection::AllowHorizontal ( )
```

Returns a boolean value indicating if cars are currently allowed to move horizontally through the intersection.

Returns

bool

4.10.3.3 AllowVertical()

```
bool Intersection::AllowVertical ( )
```

Returns a boolean value indicating if cars are currently allowed to move vertically through the intersection.

Returns

bool

4.10.3.4 Draw()

```
void Intersection::Draw (
            sf::RenderWindow & window,
            int cellSize,
            Grid * grid )
```

Draws the intersection into the SFML window given as parameter.

Parameters

window	The SFML window to draw the intersection to
cellSize	The size of a single cell in the window

4.10.3.5 GetLocation()

```
std::pair< int, int > Intersection::GetLocation ( ) const
```

Get the location of the intersection.

Returns

The coordinates of the intersection as std::pair<int, int>

4.10.3.6 Update()

```
void Intersection::Update (
            float deltaTime )
```

Attempts to update the status of the traffic lights in the intersection, if there are any.

Parameters

deltaTime	

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

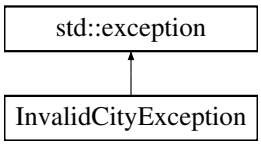
- src/intersection.hpp
- src/intersection.cpp

4.11 InvalidCityException Class Reference

An exception class extending the std::exception class. Used in situations where the JSON file provided by the user is invalid.

`#include <invalidcityexception.hpp>`

Inheritance diagram for InvalidCityException:



Public Member Functions

- InvalidCityException (std::string errorMessage)

  Construct a new InvalidCityException object.
- std::string GetError () const

  Return the error message describing what went wrong.

4.11.1 Detailed Description

An exception class extending the std::exception class. Used in situations where the JSON file provided by the user is invalid.

4.11.2 Constructor & Destructor Documentation

4.11.2.1 InvalidCityException()

```
InvalidCityException::InvalidCityException (
            std::string errorMessage )  [inline]
```

Construct a new InvalidCityException object.

Parameters

errorMessage	

4.11.3 Member Function Documentation

4.11.3.1 GetError()

```
std::string InvalidCityException::GetError ( ) const  [inline]
```

Return the error message describing what went wrong.

Returns

std::string

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

- src/invalidcityexception.hpp

4.12 Node Class Reference

A class for representing a node in the simulated city. A node can be either a Building or an Intersection, and two nodes can be connected with a Road object.

```
#include <node.hpp>
```

Public Member Functions

- Node (NodeType type, const std::pair< int, int > location)
  Construct a new Node object.
- NodeType GetType () const
  Returns the type of the node.
- std::vector< std::pair< Node *, int > > GetConnections () const
  Get the nodes that this node is connected to.
- void AddConnection (Node *node, int weight)
  Connects this node with the node that is given as parameter.
- std::pair< int, int > GetLocation () const
  Get the location of the node.

4.12.1 Detailed Description

A class for representing a node in the simulated city. A node can be either a Building or an Intersection, and two nodes can be connected with a Road object.

4.12.2 Constructor & Destructor Documentation

4.12.2.1 Node()

```
Node::Node (
            NodeType type,
            const std::pair< int, int > location )
```

Construct a new Node object.

Parameters

type	The type of the node as NodeType (Building or Intersection)
location	The coordinates of the node as std::pair<int, int>

4.12.3 Member Function Documentation

4.12.3.1 AddConnection()

```
void Node::AddConnection (
            Node * node,
            int weight )
```

Connects this node with the node that is given as parameter.

Parameters

node	The node that the connection will be made with
weight	The weight of the node that the connection is made with

4.12.3.2 GetConnections()

```
std::vector< std::pair< Node *, int > > Node::GetConnections ( ) const
```

Get the nodes that this node is connected to.

Returns

std::vector<std::pair<Node*, int>>

4.12.3.3 GetLocation()

```
std::pair< int, int > Node::GetLocation ( ) const
```

Get the location of the node.

Returns

The coordinates of the node as std::pair<int, int>

4.12.3.4 GetType()

```
NodeType Node::GetType ( ) const
```

Returns the type of the node.

Returns

A NodeType object (Building or Intersection)

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

- src/node.hpp
- src/node.cpp

4.13 Person Class Reference

A class for persons in the city. Each person has a name, type, working place and home. In addition they will have a car and a schedule, once they are added into the city. Starting location for person and its car is at home.

```
#include <person.hpp>
```

Public Member Functions

- Person (const std::string &name, PersonType personType, Industrial *workplace, Residential *home)
  Construct a new Person object.
- Industrial * GetWorkplace () const
  Get the person's working place.
- Residential * GetResidence () const
  Get the person's home.
- Car * GetCar ()
  Get the person's car.
- std::string GetName () const
  Get the person's name.
- void BuyCar (Car *car)
  Buy a car for person. This is done when the person is added to city, so the person can travel.

- void InitializeSchedule (std::map< int, Node * > schedule)

  Initialize the schedule for the person and its car. This is done after person is added to a city and it has "bought" a car.
- bool isAtHome () const

  Gives boolean value that tells, is the person at home (true), or not (false).
- std::pair< int, int > GetLocation () const

  Gives the location of the person.
- PersonType GetPersonType () const

  Gives type of the person.
- std::map< int, Node * > GetSchedule () const

  Gives the schedule for the person.
- void AddEvent (int time, Node *node)

  Adds a new event to the schedule.
- void UpdateLocationFromCar (std::pair< float, float > location)

  When car travels, person travels in the car. This makes it possible.

4.13.1 Detailed Description

A class for persons in the city. Each person has a name, type, working place and home. In addition they will have a car and a schedule, once they are added into the city. Starting location for person and its car is at home.

4.13.2 Constructor & Destructor Documentation

4.13.2.1 Person()

```
Person::Person (
            const std::string & name,
            PersonType personType,
            Industrial * workplace,
            Residential * home )
```

Construct a new Person object.

Parameters

name	The name of the person
personType	The type of person (Neutral, Angry, Lazy, Gentleman, Active and Nocturnal)
workplace	Pointer to a Industrial object. Persons workplace
home	Pointer to a Residential object. Persons home.

4.13.3 Member Function Documentation

4.13.3.1 AddEvent()

```
void Person::AddEvent (
            int time,
            Node * node )
```

Adds a new event to the schedule.

Parameters

time	Time in minutes when this event occurs.
node	Pointer to the destination Node object.

4.13.3.2 BuyCar()

```
void Person::BuyCar (
            Car * car )
```

Buy a car for person. This is done when the person is added to city, so the person can travel.

Parameters

car	Pointer to a Car object.

4.13.3.3 GetCar()

```
Car * Person::GetCar ( )
```

Get the person's car.

Returns

Pointer to a Car object.

4.13.3.4 GetLocation()

```
std::pair< int, int > Person::GetLocation ( ) const
```

Gives the location of the person.

Returns

Coordinates of the location as a pair of ints.

4.13.3.5 GetName()

```
std::string Person::GetName ( ) const
```

Get the person's name.

Returns

Name as string.

4.13.3.6 GetPersonType()

```
PersonType Person::GetPersonType ( ) const
```

Gives type of the person.

Returns

The type of the person as PersonType, that is implemented in car.hpp

4.13.3.7 GetResidence()

```
Residential * Person::GetResidence ( ) const
```

Get the person's home.

Returns

Pointer to a Residential object.

4.13.3.8 GetSchedule()

```
std::map< int, Node * > Person::GetSchedule ( ) const
```

Gives the schedule for the person.

Returns

Schedule which is a map that contains minutes as keys, and pointers to Node objects as events.

4.13.3.9 GetWorkplace()

```
Industrial * Person::GetWorkplace ( ) const
```

Get the person's working place.

Returns

Pointer to an Industrial object.

4.13.3.10 InitializeSchedule()

```
void Person::InitializeSchedule (
            std::map< int, Node * > schedule )
```

Initialize the schedule for the person and its car. This is done after person is added to a city and it has "bought" a car.

Parameters

schedule	The random schedule that is given for a person.

4.13.3.11 isAtHome()

```
bool Person::isAtHome ( ) const
```

Gives boolean value that tells, is the person at home (true), or not (false).

Returns

Boolean value, false or true.

4.13.3.12 UpdateLocationFromCar()

```
void Person::UpdateLocationFromCar (
            std::pair< float, float > location )
```

When car travels, person travels in the car. This makes it possible.

Parameters

location	The location of the car.

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

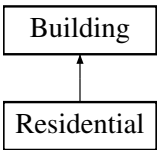
- src/person.hpp
- src/person.cpp

4.14 Residential Class Reference

One of the concretizing classes for the abstract class Building.

`#include <residential.hpp>`

Inheritance diagram for Residential:



Public Member Functions

- Residential (const std::string &buildingName, const std::pair< int, int > &location)

  Construct a new Residential object.
- virtual void Draw (sf::RenderWindow &window, int cellSize) override

  Draws the building to the SFML window given as a parameter. Each subclass defines the draw function themselves.
- std::string GetType () const override

  Prints information about the residential building to the standard output.

Additional Inherited Members

4.14.1 Detailed Description

One of the concretizing classes for the abstract class Building.

4.14.2 Constructor & Destructor Documentation

4.14.2.1 Residential()

```
Residential::Residential (
            const std::string & buildingName,
            const std::pair< int, int > & location )  [inline]
```

Construct a new Residential object.

Parameters

buildingName	The name of the residential building
location	The coordinates of the residential building

4.14.3 Member Function Documentation

4.14.3.1 Draw()

```
virtual void Residential::Draw (
            sf::RenderWindow & window,
            int cellSize )  [inline], [override], [virtual]
```

Draws the building to the SFML window given as a parameter. Each subclass defines the draw function themselves.

Parameters

window	A reference to the SFML window where the building should be drawn
cellSize	The size of the cell to be drawn

Implements Building.

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

- src/residential.hpp

4.15 Road Class Reference

A class for representing a road in the simulated city.

```
#include <road.hpp>
```

Public Member Functions

- Road (std::pair< int, int > start, std::pair< int, int > end, int speedLimit)
  Construct a new Road object.
- std::pair< int, int > GetEnd () const
  Get the coordinates of the end of the road.
- std::pair< int, int > GetStart () const
  Get the coordinates of the start of the road.
- int GetSpeedLimit () const
  Get the speed limit of the road.
- bool IsVertical () const
  Returns a boolean value indicating if the road os vertical or not.
- bool IsHorizontal () const
  Returns a boolean value indicating if the road os horizontal or not.

4.15.1 Detailed Description

A class for representing a road in the simulated city.

4.15.2 Constructor & Destructor Documentation

4.15.2.1 Road()

```
Road::Road (
            std::pair< int, int > start,
            std::pair< int, int > end,
            int speedLimit )
```

Construct a new Road object.

Parameters

start	The starting coordinates of the road
end	The ending coordinates of the road
speedLimit	The speed limit on the road

4.15.3 Member Function Documentation

4.15.3.1 GetEnd()

```
std::pair< int, int > Road::GetEnd ( ) const
```

Get the coordinates of the end of the road.

Returns

std::pair<int, int>

4.15.3.2 GetSpeedLimit()

```
int Road::GetSpeedLimit ( ) const
```

Get the speed limit of the road.

Returns

The speed limit as an integer

4.15.3.3 GetStart()

```
std::pair< int, int > Road::GetStart ( ) const
```

Get the coordinates of the start of the road.

Returns

std::pair<int, int>

4.15.3.4 IsHorizontal()

```
bool Road::IsHorizontal ( ) const
```

Returns a boolean value indicating if the road os horizontal or not.

Returns

True if the road is horizontal, false otherwise

4.15.3.5 IsVertical()

```
bool Road::IsVertical ( ) const
```

Returns a boolean value indicating if the road os vertical or not.

Returns

True if the road is vertical, false otherwise

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

- src/road.hpp
- src/road.cpp

4.16 SimulationClock Class Reference

Class used for time management in the simulation. Includes a clock that runs with regard to system time. One second in real life equals one minute in simulation time.

```
#include <simulationClock.hpp>
```

Public Member Functions

- SimulationClock ()

  Construct a new Simulation Clock object.
- void Start ()

  Function that starts the clock.
- double GetElapsedTime ()

  Returns the elapsed time in simulation.
- int GetDayNumber ()

  Returns the day number of the simulation.
- std::string GetSimulationTime ()

  Returns the simulation's time (1 second in real life = 1 minute in simulation time)
- void SetSimulationSpeed (int simulationSpeed)

  Set the Simulation Speed object.

4.16.1 Detailed Description

Class used for time management in the simulation. Includes a clock that runs with regard to system time. One second in real life equals one minute in simulation time.

4.16.2 Member Function Documentation

4.16.2.1 GetDayNumber()

```
int SimulationClock::GetDayNumber ( )
```

Returns the day number of the simulation.

Returns

An integer value that represents the day number

4.16.2.2 GetElapsedTime()

```
double SimulationClock::GetElapsedTime ( )
```

Returns the elapsed time in simulation.

Returns

A double value that represents the elapsed time in seconds

4.16.2.3 GetSimulationTime()

```
std::string SimulationClock::GetSimulationTime ( )
```

Returns the simulation's time (1 second in real life = 1 minute in simulation time)

Returns

A string value that contains the time in "xx:xx" form

4.16.2.4 SetSimulationSpeed()

```
void SimulationClock::SetSimulationSpeed (
            int simulationSpeed )  [inline]
```

Set the Simulation Speed object.

Parameters

simulationSpeed	

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

- src/simulationClock.hpp
- src/simulationClock.cpp

4.17 Simulator Class Reference

Class that handles the crucial parts of simulation. This class includes the main threads that the user uses to control the simulation. The simulation is initialized/finished in this class.

```
#include <simulator.hpp>
```

Public Member Functions

- Simulator ()

  Construct a new Simulator object.
- ~Simulator ()

  Destroy the Simulator object.
- void StartSimulation ()

  Starts and initializes the simulation.
- void SimulatorThread ()

  The simulator's main thread that is used to run the simulation.
- void UpdateSimulation (float deltaTime, float currentTime)

  Updates the simulation with regard to time changes.

- void DrawSimulation (Visualization *gui)

  Draws the simulation if the GUI is enabled.
- City * LoadCity ()

  Loads the city from a .json file.
- void InputThread (std::shared_future< void > exitFuture)

  Thread that is used for user input via terminal.
- void SetCity (City *c)

  Set the City object.

4.17.1 Detailed Description

Class that handles the crucial parts of simulation. This class includes the main threads that the user uses to control the simulation. The simulation is initialized/finished in this class.

4.17.2 Member Function Documentation

4.17.2.1 DrawSimulation()

```
void Simulator::DrawSimulation (
            Visualization * gui )
```

Draws the simulation if the GUI is enabled.

Parameters

gui	graphical user interface used in visualization

4.17.2.2 InputThread()

```
void Simulator::InputThread (
            std::shared_future< void > exitFuture )
```

Thread that is used for user input via terminal.

Parameters

exitFuture	information about the thread status (exit or not)

4.17.2.3 LoadCity()

```
City * Simulator::LoadCity ( )
```

Loads the city from a .json file.

Returns

City∗

4.17.2.4 SetCity()

```
void Simulator::SetCity (
            City * c )  [inline]
```

Set the City object.

Parameters

c	city that is set

4.17.2.5 UpdateSimulation()

```
void Simulator::UpdateSimulation (
            float deltaTime,
            float currentTime )
```

Updates the simulation with regard to time changes.

Parameters

deltaTime	tells how much time has passed in the main loop
currentTime	tells what is the current system time

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

- src/simulator.hpp
- src/simulator.cpp

4.18 TrafficLight Class Reference

A class for representing the traffic lights in the simulated city.

```
#include <trafficlight.hpp>
```

Public Member Functions

- TrafficLight (std::pair< int, int > location, int redAndGreenDuration, int yellowDuration)

  Construct a new Traffic Light object.
- std::pair< int, int > GetLocation () const

  Get the coordinates of the traffic light's location.
- int GetRedDuration () const

  Get the red light duration of the traffic light.
- int GetYellowDuration () const

  Get the yellow light duration of the traffic light.
- int GetGreenDuration () const

  Get the green light duration of the traffic light.
- void TurnRed ()

  Switches the state of the traffic light to red.
- void TurnYellow ()

  Switches the state of the traffic light to yellow.
- void TurnGreen ()

  Switches the state of the traffic light to green.
- std::string GetStatus () const

  Get the current color of the traffic light.

4.18.1 Detailed Description

A class for representing the traffic lights in the simulated city.

4.18.2 Constructor & Destructor Documentation

4.18.2.1 TrafficLight()

```
TrafficLight::TrafficLight (
            std::pair< int, int > location,
            int redAndGreenDuration,
            int yellowDuration )  [inline]
```

Construct a new Traffic Light object.

Parameters

location	The coordinates of the traffic light
redDuration	The duration of the red light
yellowDuration	The duration of the yellow light
greenDuration	The duration of the green light

4.18.3 Member Function Documentation

4.18.3.1 GetGreenDuration()

```
int TrafficLight::GetGreenDuration ( ) const  [inline]
```

Get the green light duration of the traffic light.

Returns

Green light duration

4.18.3.2 GetLocation()

```
std::pair<int, int> TrafficLight::GetLocation ( ) const  [inline]
```

Get the coordinates of the traffic light's location.

Returns

The coordinates of the traffic light as std::pair<int, int>

4.18.3.3 GetRedDuration()

```
int TrafficLight::GetRedDuration ( ) const  [inline]
```

Get the red light duration of the traffic light.

Returns

Red light duration

4.18.3.4 GetStatus()

```
std::string TrafficLight::GetStatus ( ) const  [inline]
```

Get the current color of the traffic light.

Returns

A string representing the current state of the traffic light ("red", "yellow", or "green")

4.18.3.5 GetYellowDuration()

```
int TrafficLight::GetYellowDuration ( ) const  [inline]
```

Get the yellow light duration of the traffic light.

Returns

Yellow light duration

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

- src/trafficlight.hpp

4.19 Visualization Class Reference

Class used for visualizing the traffic simulator in a separate GUI.

```
#include <visualization.hpp>
```

Public Member Functions

- Visualization (int cellSize, Grid *g)
  Construct a new Visualization object.
- sf::RenderWindow & GetWindow ()
  Used for returning the Window object.
- void PrintGrid (Grid *grid)
  Prints out the current grid.
- void PrintCars (std::vector< Car * > cars)
  Prints out the current cars.

4.19.1 Detailed Description

Class used for visualizing the traffic simulator in a separate GUI.

4.19.2 Constructor & Destructor Documentation

4.19.2.1 Visualization()

```
Visualization::Visualization (
            int cellSize,
            Grid * g )
```

Construct a new Visualization object.

Parameters

cellSize	cell size in the GUI
g	pointer to a grid

4.19.3 Member Function Documentation

4.19.3.1 GetWindow()

```
sf::RenderWindow & Visualization::GetWindow ( )
```

Used for returning the Window object.

Returns

sf::RenderWindow&

4.19.3.2 PrintCars()

```
void Visualization::PrintCars (
            std::vector< Car * > cars )
```

Prints out the current cars.

Parameters

cars	

4.19.3.3 PrintGrid()

```
void Visualization::PrintGrid (
            Grid * grid )
```

Prints out the current grid.

Parameters

grid	

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

- src/visualization.hpp
- src/visualization.cpp

Index