

## Railroad Tycoon Prototyp

Erzeugt von Doxygen 1.8.17



# Inhaltsverzeichnis



# Kapitel 1

## SWT Praktikum

Hier ist eine kleine Anleitung wie man das Projekt auf seinem eigenen Rechner synchronisiert:

1. git installieren
2. `>> git clone https://github.com/davidtraum/swt/`
3. `>> cd swt`

Wenn man was geändert hat:

(0. Ins Basisverzeichnis vom Projekt gehen)

1. `>> git add *`
1. `>> git commit -m "Kurze Nachricht was man gemacht hat"`
2. `>> git push origin master` (Oder eigenen Branch angeben)



## Kapitel 2

# Hierarchie-Verzeichnis

### 2.1 Klassenhierarchie

Die Liste der Ableitungen ist -mit Einschränkungen- alphabetisch sortiert:

City . . . . .	??
GraphicsManager . . . . .	??
MapTile . . . . .	??
QGraphicsScene	
Scene . . . . .	??
QGraphicsView	
View . . . . .	??
QMainWindow	
MainWindow . . . . .	??





## Kapitel 3

# Klassen-Verzeichnis

### 3.1 Auflistung der Klassen

Hier folgt die Aufzählung aller Klassen, Strukturen, Varianten und Schnittstellen mit einer Kurzbeschreibung:

City	??
GraphicsManager	??
MainWindow	??
MapTile	??
Scene	??
View	??



# Kapitel 4

## Klassen-Dokumentation

### 4.1 City Klassenreferenz

#### Öffentliche Methoden

- `City` (int pX, int pY, int pSize)  
*`City::City` Erzeugt eine Stadt mit vorgegebenen Parametern.*
- `City` ()  
*`City::City` Erzeugt eine leere Stadt.*
- int `getSize` ()  
*`City::getSize` Gibt die Anzahl der Felder zurück die zur Stadt gehören.*
- int `getCenterX` ()  
*`City::getCenterX` Gibt den X-Index des Mittelpunktes.*
- int `getCenterY` ()  
*`City::getCenterY` Gibt den Y-Index des Mittelpunktes.*
- std::string `getName` ()  
*`City::getName` Gibt den Namen der Stadt.*
- void `setSize` (int pSize)  
*`City::setSize` Gibt die Größe der Stadt zurück (Anzahl der Gebäude)*
- void `setCenter` (int pX, int pY)  
*`City::setCenter` Setzt den Mittelpunkt der Stadt.*
- void `setName` (std::string pName)  
*`City::setName` Setzt den Namen der Stadt.*

#### Private Attribute

- int `size`
- int `centerX`
- int `centerY`
- std::string `name`

#### 4.1.1 Beschreibung der Konstruktoren und Destruktoren

#### 4.1.1.1 City()

```
City::City (
    int pX,
    int pY,
    int pSize )
```

[City::City](#) Erzeugt eine Stadt mit vorgegebenen Parametern.

##### Parameter

<i>pX</i>	Der X-Index des Mittelpunktes.
<i>pY</i>	Der Y-Index des Mittelpunktes.
<i>pSize</i>	Die gröÙe der Stadt.

#### 4.1.2 Dokumentation der Elementfunktionen

##### 4.1.2.1 getCenterX()

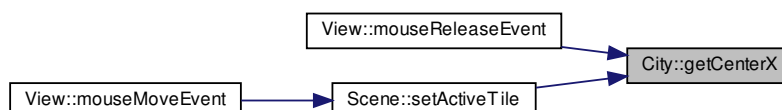
```
int City::getCenterX ( )
```

[City::getCenterX](#) Gibt den X-Index des Mittelpunktes.

##### Rückgabe

Der X-Index des Mittelpunktes der Stadt.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



#### 4.1.2.2 getCenterY()

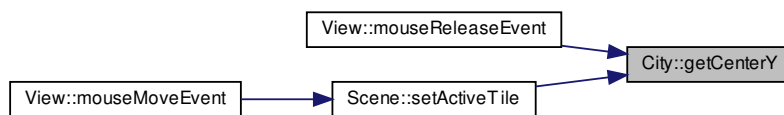
```
int City::getCenterY ( )
```

[City::getCenterX](#) Gibt den Y-Index des Mittelpunktes.

##### Rückgabe

Der Y-Index des Mittelpunktes der Stadt.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



#### 4.1.2.3 getName()

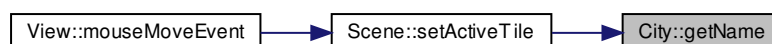
```
std::string City::getName ( )
```

[City::getName](#) Gibt den Namen der Stadt.

##### Rückgabe

Der Name der Stadt.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



#### 4.1.2.4 getSize()

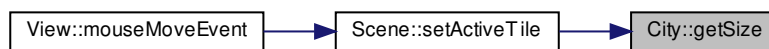
```
int City::getSize ( )
```

[City::getSize](#) Gibt die Anzahl der Felder zurück die zur Stadt gehören.

##### Rückgabe

Die Größe.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



#### 4.1.2.5 setCenter()

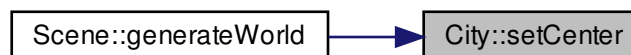
```
void City::setCenter (
    int pX,
    int pY )
```

[City::setCenter](#) Setzt den Mittelpunkt der Stadt.

##### Parameter

<i>pX</i>	Der X-Index.
<i>pY</i>	Der Y-Index.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



#### 4.1.2.6 setName()

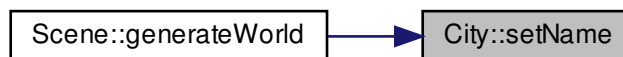
```
void City::setName (
    std::string pName )
```

[City::setName](#) Setzt den Namen der Stadt.

##### Parameter

<code>pName</code>	Der neue Name.
--------------------	----------------

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



#### 4.1.2.7 setSize()

```
void City::setSize (
    int pSize )
```

[City::setSize](#) Gibt die Größe der Stadt zurück (Anzahl der Gebäude)

##### Parameter

<code>pSize</code>	Die Größe der Stadt
--------------------	---------------------

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- `src/application/city.h`
- `src/application/city.cpp`

## 4.2 GraphicsManager Klassenreferenz

### Öffentliche Methoden

- [GraphicsManager \(\)](#)  
*[GraphicsManager::GraphicsManager](#) Lädt alle Grafiken. Neue bitte im selben Stil ergänzen.*
- QPixmap [get](#) (std::string key)  
*[GraphicsManager::get](#) Liefert eine Grafik mit einem bestimmten Namen.*

### Öffentliche Attribute

- std::map< std::string, QPixmap > **GRAPHICS**

### 4.2.1 Dokumentation der Elementfunktionen

#### 4.2.1.1 get()

```
QPixmap GraphicsManager::get (
    std::string key )
```

[GraphicsManager::get](#) Liefert eine Grafik mit einem bestimmten Namen.

#### Parameter

<i>key</i>	Name der Grafik.
------------	------------------

#### Rückgabe

Die Grafik.

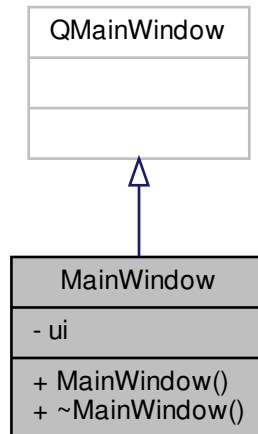
Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- src/application/graphicsmanager.h
- src/application/graphicsmanager.cpp



## 4.3 MainWindow Klassenreferenz

Klassendiagramm für MainWindow:



### Öffentliche Methoden

- [MainWindow](#) (QWidget \*parent=nullptr)  
*MainWindow::MainWindow.*
- [~MainWindow](#) ()  
*MainWindow::~~MainWindow.*

### Private Attribute

- Ui::MainWindow \* **ui**

### 4.3.1 Beschreibung der Konstruktoren und Destruktoren

#### 4.3.1.1 MainWindow()

```
MainWindow::MainWindow (
    QWidget * parent = nullptr )
```

[MainWindow::MainWindow.](#)

## Parameter

<i>parent</i>	
---------------	--

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- src/application/mainwindow.h
- src/application/mainwindow.cpp

## 4.4 MapTile Klassenreferenz

### Öffentliche Typen

- enum **TYPE** {  
**GRASS, FORREST, CITY, CITY\_CENTER,**  
**RIVER\_H, RIVER\_V, RIVER\_LB, RIVER\_LT,**  
**RIVER\_RT, RIVER\_RB, RAIL\_H, RAIL\_CURVE }**

### Öffentliche Methoden

- [MapTile \(\)](#)  
[MapTile::MapTile](#) Konstruktor.
- void [setType](#) (TYPE pType)  
[MapTile::setType](#) Setzt den Typ der Kachel.
- TYPE [getType](#) ()  
[MapTile::getType](#) Liefert den Typ des Quadranten.
- void [setRotation](#) (int pRotation)  
[MapTile::setRotation](#) Hilfsfunktion zur Rotation im Quadrat.
- int [getRotation](#) ()  
[MapTile::getRotation](#) Liefert die aktuelle Rotation. (Himmelsrichtung)
- void [setPosition](#) (int posX, int posY)  
[MapTile::setPosition](#) Setzt die Position der Kachel. (In Pixeln)
- bool [isRiver](#) ()  
[MapTile::isRiver](#) Checkt ob die Kachel ein Fluss ist.
- QGraphicsPixmapItem \* [getPixmapItem](#) ()  
[MapTile::getPixmapItem](#) Liefert das QPixmap Item.
- City \* [getCity](#) ()  
[MapTile::getCity](#) Die Informationen. Falls keine Stadt: null.
- void [setCity](#) (City \*pCity)  
[MapTile::setCity](#).
- int [getX](#) ()  
[MapTile::getX](#).
- int [getY](#) ()  
[MapTile::getY](#).

## Private Attribute

- TYPE **type**
- QGraphicsPixmapItem \* **pixmapItem**
- int **currentRotation**
- [City](#) \* **city**

## 4.4.1 Dokumentation der Elementfunktionen

### 4.4.1.1 `getCity()`

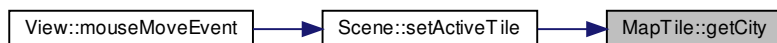
```
City * MapTile::getCity ( )
```

[MapTile::getCity](#) Die Informationen. Falls keine Stadt: null.

#### Rückgabe

Liefert die Informationen über eine Stadt auf der Kachel.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



### 4.4.1.2 `getPixmapItem()`

```
QGraphicsPixmapItem * MapTile::getPixmapItem ( )
```

[MapTile::getPixmapItem](#) Liefert das Pixmap Item.

#### Rückgabe

Das Pixmap Item.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



#### 4.4.1.3 getRotation()

```
int MapTile::getRotation ( )
```

[MapTile::getRotation](#) Liefert die aktuelle Rotation. (Himmelsrichtung)

##### Rückgabe

Die aktuelle Rotation (0-3)

#### 4.4.1.4 getType()

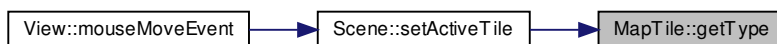
```
MapTile::TYPE MapTile::getType ( )
```

[MapTile::getType](#) Liefert den Typ des Quadranten.

##### Rückgabe

Den Typ.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



#### 4.4.1.5 getX()

```
int MapTile::getX ( )
```

[MapTile::getX](#).

##### Rückgabe

Der X Index des Quadranten.

#### 4.4.1.6 getY()

```
int MapTile::getY ( )
```

[MapTile::getY](#).

##### Rückgabe

Der Y Index des Quadranten.

#### 4.4.1.7 isRiver()

```
bool MapTile::isRiver ( )
```

[MapTile::isRiver](#) Checkt ob die Kachel ein Fluss ist.

##### Rückgabe

Ob die Kachel ein Fluss ist.

#### 4.4.1.8 setCity()

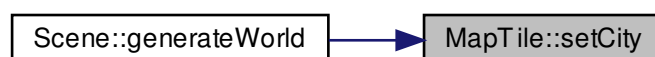
```
void MapTile::setCity (
    City * pCity )
```

[MapTile::setCity](#).

##### Parameter

<i>pCity</i>	Fügt dem Quadranten Daten über eine Stadt hinzu.
--------------	--

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



#### 4.4.1.9 setPosition()

```
void MapTile::setPosition (
    int posX,
    int posY )
```

[MapTile::setPosition](#) Setzt die Position der Kachel. (In Pixeln)

##### Parameter

<i>posX</i>	Die X Koordinate.
<i>posY</i>	Die Y Koordinate.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



#### 4.4.1.10 setRotation()

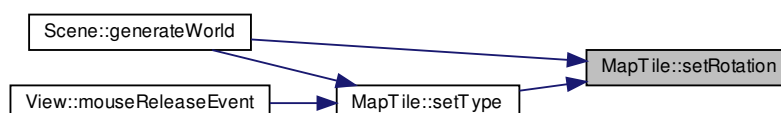
```
void MapTile::setRotation (
    int pRotation )
```

[MapTile::setRotation](#) Hilfsfunktion zur Rotation im Quadrat.

##### Parameter

<i>pRotation</i>	0=Ursprung 1=90° Grad 2=180° Grad 3=270°
------------------	--

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



#### 4.4.1.11 setType()

```
void MapTile::setType (
    MapTile::TYPE pType )
```

[MapTile::setType](#) Setzt den Typ der Kachel.

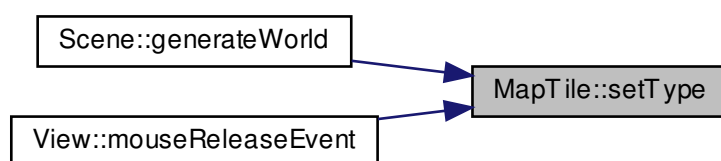
Parameter

<i>pType</i>	Der Typ.
--------------	----------

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

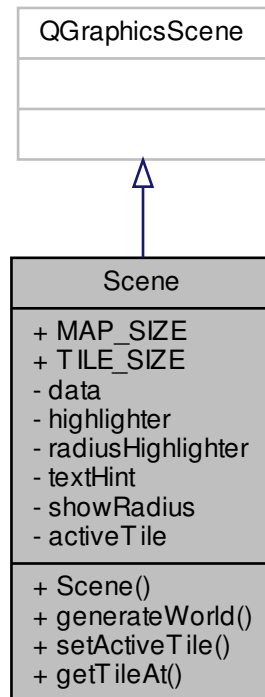


Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- `src/application/maptile.h`
- `src/application/maptile.cpp`

## 4.5 Scene Klassenreferenz

Klassendiagramm für Scene:



### Öffentliche Methoden

- [Scene](#) ()  
*Scene::Scene* Konstruktor.
- void [generateWorld](#) ()  
*Scene::generateWorld* Diese Methode generiert eine neue Welt.
- void [setActiveTile](#) (QGraphicsItem \*pltem)  
*Scene::setActiveTile* Setzt den [MapTile](#) über dem die Maus gerade ist. Wird von view aufgerufen.
- [MapTile](#) \* [getTileAt](#) (int posX, int posY, bool isPixelCoordinate)  
*Scene::getTileAt* Liefert ein [MapTile](#) anhand der Pixel-Koordinaten.

### Statische öffentliche Attribute

- const static int **MAP\_SIZE** {300}
- const static int **TILE\_SIZE** {64}



## Private Attribute

- [MapTile](#) **data** [Scene::MAP\_SIZE][Scene::MAP\_SIZE]
- QGraphicsRectItem \* **highlighter**
- QGraphicsEllipseItem \* **radiusHighlighter**
- QGraphicsTextItem \* **textHint**
- bool **showRadius**
- [MapTile](#) \* **activeTile**

### 4.5.1 Dokumentation der Elementfunktionen

#### 4.5.1.1 `getTileAt()`

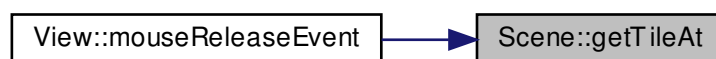
```
MapTile * Scene::getTileAt (
    int posX,
    int posY,
    bool isPixelCoordinate )
```

[Scene::getTileAt](#) Liefert ein [MapTile](#) anhand der Pixel-Koordinaten.

##### Parameter

<i>posX</i>	Die X-Koordinate
<i>posY</i>	Die Y-Koordinate

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



#### 4.5.1.2 `setActiveTile()`

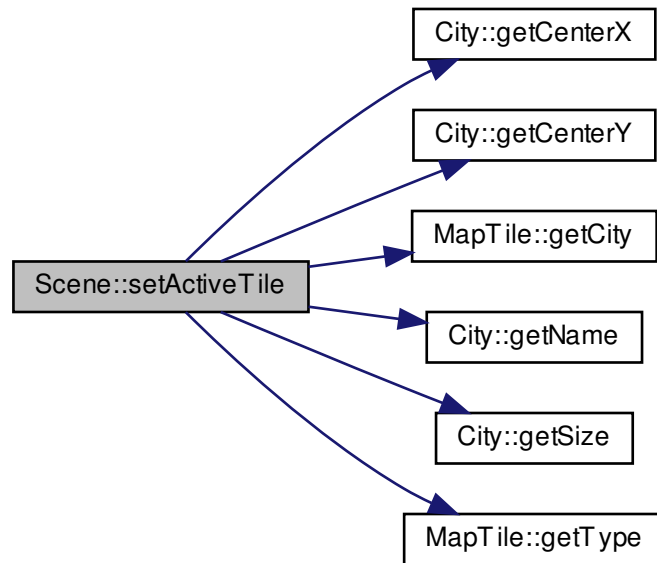
```
void Scene::setActiveTile (
    QGraphicsItem * pItem )
```

[Scene::setActiveTile](#) Setzt den [MapTile](#) über dem die Maus gerade ist. Wird von view aufgerufen.

## Parameter

<i>pltem</i>	Ein Grafikitem zu dem die Methode den zugehörigen Maptile bestimmt.
--------------	---

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

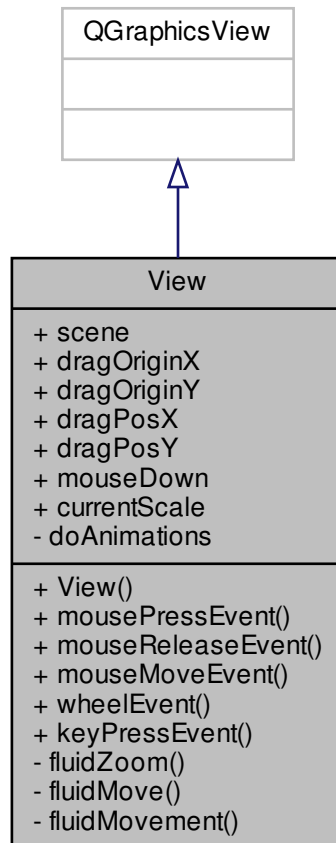


Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- `src/application/scene.h`
- `src/application/scene.cpp`

## 4.6 View Klassenreferenz

Klassendiagramm für View:



### Öffentliche Methoden

- [View](#) ([Scene](#) \*pScene)  
*View::View* Konstruktor. Versteckt u.a. die Scrollbars und aktiviert Mousetracking.
- void [mousePressEvent](#) (QMouseEvent \*event) override  
*View::mousePressEvent* QT Methode. Wird aufgerufen wenn die Maus gedrückt wurde.
- void [mouseReleaseEvent](#) (QMouseEvent \*event) override  
*View::mouseReleaseEvent* QT Methode. Wird aufgerufen wenn die Maus losgelassen wird.
- void [mouseMoveEvent](#) (QMouseEvent \*event) override  
*View::mouseMoveEvent* QT Methode. Wird aufgerufen wenn die Maus bewegt wird.
- void [wheelEvent](#) (QWheelEvent \*event) override  
*View::wheelEvent* QT Methode. Wird aufgerufen wenn das Mousrad gedreht wird.
- void [keyPressEvent](#) (QKeyEvent \*event) override  
*View::keyPressEvent* QT Methode. Wird Aufgerufen wenn eine Taste gedrückt wird.

## Öffentliche Attribute

- [Scene](#) \* scene
- int dragOriginX
- int dragOriginY
- int dragPosX
- int dragPosY
- bool mouseDown
- double currentScale {1.0}

## Private Methoden

- void [fluidZoom](#) (double target, bool in)  
*[View::fluidZoom](#) Startet eine Zoom-Animation. Zuvor muss doAnimations=true gesetzt sein. Bsp: fluidZoom(3, true) zoomt 3x in die Karte hinein.*
- void [fluidMove](#) (int vX, int vY)  
*[View::fluidMove](#) Verschiebt die Karte animiert und relativ zur aktuellen Position.*
- void [fluidMovement](#) (int pX, int pY)  
*[View::fluidMovement](#) Verschiebt die Karte animiert an zu einer absoluten Koordinate.*

## Private Attribute

- bool doAnimations

## 4.6.1 Beschreibung der Konstruktoren und Destruktoren

### 4.6.1.1 View()

```
View::View (
    Scene * pScene )
```

[View::View](#) Konstruktor. Versteckt u.a. die Scrollbars und aktiviert Mousetracking.

#### Parameter

<a href="#">pScene</a>	Das Zugehörige Szenenobjekt.
------------------------	------------------------------

## 4.6.2 Dokumentation der Elementfunktionen

### 4.6.2.1 fluidMove()

```
void View::fluidMove (
    int vX,
```

```
int vY ) [private]
```

**View::fluidMove** Verschiebt die Karte animiert und relativ zur aktuellen Position.

#### Parameter

vX	Verschiebung in X-Richtung.
vY	Verschiebung in Y-Richtung.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



#### 4.6.2.2 fluidMovement()

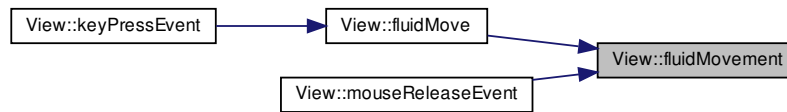
```
void View::fluidMovement (  
    int pX,  
    int pY ) [private]
```

**View::fluidMovement** Verschiebt die Karte animiert an zu einer absoluten Koordinate.

#### Parameter

pX	Die X-Koordinate.
pY	Die Y-Koordinate.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



#### 4.6.2.3 fluidZoom()

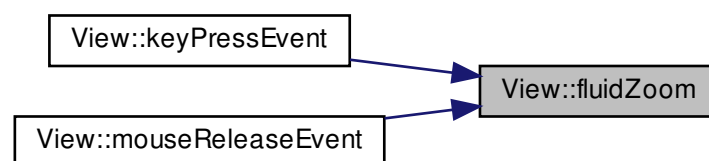
```
void View::fluidZoom (
    double target,
    bool in ) [private]
```

[View::fluidZoom](#) Startet eine Zoom-Animation. Zuvor muss `doAnimations=true` gesetzt sein. Bsp: `fluidZoom(3, true)` zoomt 3x in die Karte hinein.

##### Parameter

<i>target</i>	Die angestrebte Skalierung.
<i>in</i>	Ob vergrößert oder verkleinert werden soll. (true = reinzoomen, false=rauszoomen).

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



#### 4.6.2.4 keyPressEvent()

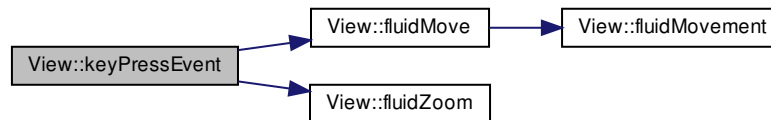
```
void View::keyPressEvent (
    QKeyEvent * event ) [override]
```

[View::keyPressEvent](#) QT Methode. Wird Aufgerufen wenn eine Taste gedrückt wird.

## Parameter

<i>event</i>	Event mit Informationen. Wichtig: <code>event-&gt;text()</code> : Text der Taste und <code>event-&gt;key()</code> : Id der Taste
--------------	--

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



## 4.6.2.5 mouseMoveEvent()

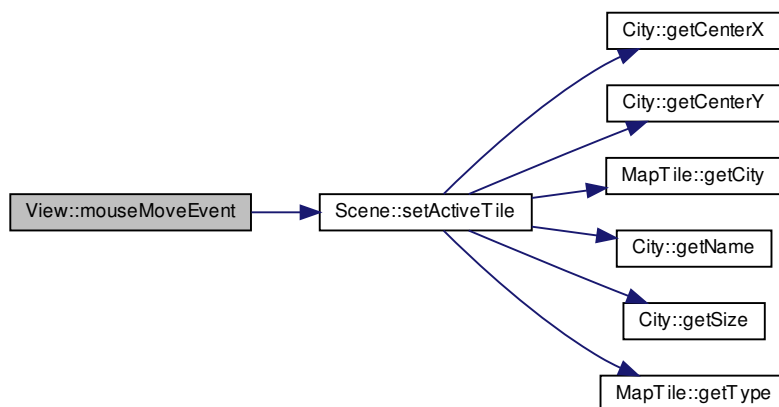
```
void View::mouseMoveEvent (
    QMouseEvent * event ) [override]
```

[View::mouseMoveEvent](#) QT Methode. Wird aufgerufen wenn die Maus bewegt wird.

## Parameter

<i>event</i>	Informationen über Position der Maus
--------------	--------------------------------------

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



#### 4.6.2.6 mousePressEvent()

```
void View::mousePressEvent (
    QMouseEvent * event ) [override]
```

[View::mousePressEvent](#) QT Methode. Wird aufgerufen wenn die Maus gedrückt wurde.

##### Parameter

<i>event</i>	Enthält Informationen über die Taste und Position.
--------------	--

#### 4.6.2.7 mouseReleaseEvent()

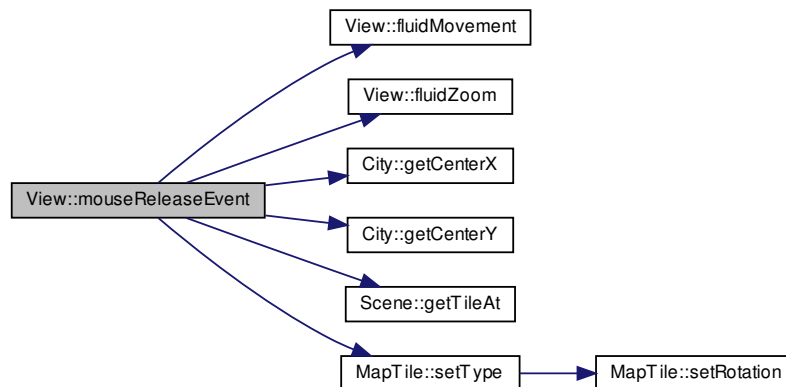
```
void View::mouseReleaseEvent (
    QMouseEvent * event ) [override]
```

[View::mouseReleaseEvent](#) QT Methode. Wird aufgerufen wenn die Maus losgelassen wird.

##### Parameter

<i>event</i>	Informationen über Position und Taste
--------------	---------------------------------------

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



#### 4.6.2.8 wheelEvent()

```
void View::wheelEvent (
    QWheelEvent * event ) [override]
```

[View::wheelEvent](#) QT Methode. Wird aufgerufen wenn das Mausexplorer gedreht wird.



## Parameter

<i>event</i>	Eventobjekt mit Infos. Wichtig: <code>event-&gt;delta()</code> : Positiv oder negativ jenachdem in welche Richtung gedreht wurde.
--------------	---

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- `src/application/view.h`
- `src/application/view.cpp`

