Model of Road Traffic

AUTHOR

Koshman Anasteysha

Версия 0.1.0.0.

Вт 19 Дек 2017

**Алфавитный указатель классов**

## Классы

Классы с их кратким описанием.

**Car (Автомобиль )**

**Crash (Авария )**

**Map (Карта )**

**RoadSign (Дорожный знак )**

**TrafficLight (Светофор )**

**View (Вид )**

# Список файлов

## Файлы

Полный список файлов.

**Car.cpp**

**Car.h**

**Crash.cpp**

**Crash.h**

**Function.cpp**

**Function.h**

**FunctionForMain.cpp**

**FunctionForMain.h**

**Include.h**

**Main.cpp**

**Map.cpp**

**Map.h**

**resource.h**

**RoadSign.cpp**

**RoadSign.h**

**TrafficLight.cpp**

**TrafficLight.h**

**UnitTest.**

**UnitTest.h**

**View.cpp**

**View.h**

# Классы

## Класс Car

Автомобиль

#include <Car.h>

### Открытые члены

**Car** (const **Map** &map, const sf::Texture &texture, sf::Color col)

**Car** (const sf::Texture &texture, sf::Color col, const float x, const float y, const int direction)

**~Car** ()

*Дeструктор - обнуляет поля объекта*

void **go** (sf::RenderWindow &window, const **Map** &map, bool &crash, bool &pause)

std::vector< int > **freeDirections** (const **Map** &map)

bool **entryExit** (const **Map** &map)

bool **crashAround** () const

void **signAround** (const **Map** &map)

int **carAround** (const **Map** &map, const bool crash)

bool **lightAround** (const **Map** &map) const

void **leaveCrossroad** (const **Map** &map)

*Метод изменяющий поля СhangeDir и Crossroad при покидании обьекта данного класса перекрестка*

void **setDirection** (const int direct)

void **setSpeed** (const float newSpeed)

void **setSizeSprite** ()

*Метод, устанавливающий размер спрайта*

bool **getLife** () const

float **getSpeed** () const

float **getX** () const

float **getY** () const

int **getDirection** () const

int **getStopDirection** () const

bool **getChangeDir** () const

int **getBeginDirection** (const **Map** &map)

int **mX** () const

int **mY** () const

int **mdX** () const

int **mdY** () const

float **okrugl** (const float xy) const

### Открытые статические члены

static std::vector< **Car** > & **Vec** ()

### Закрытые данные

float **X**

*Координата по оси Х*

float **Y**

*Координата по оси Y.*

float **dX**

*Приращение по оси Х*

float **dY**

*Приращение по оси У*

float **Speed**

*Скорость*

int **Direction**

*Направление*

sf::Sprite **Sprite**

*Спрайт обьекта*

sf::Color **Color**

*Цвет*

bool **ChangeDir**

*Смена направления*

bool **Life**

*Жизнь*

int **EntryExit**

*Переменная, отвечающая за анимация вьезда на карту и выезда с нее обьекта* ***Car****.*

int **StopDirection**

*Направление, поворот в которое, запрещен*

bool **Crossroad**

*Нахождение авто на перекрестке*

### Закрытые статические данные

static std::vector< **Car** > **AllCars**

*Вектор, хранящий все обьекты данного класса*

### Подробное описание

Автомобиль

### Конструктор(ы)

#### Car::Car (const Map & *map*, const sf::Texture & *texture*, sf::Color *col*)

##### Аргументы:

|  |  |
| --- | --- |
| *map* | - карта |
| *texture* | - текстура для спрайта обьекта |
| *col* | - цвет обьекта |

#### Car::Car (const sf::Texture & *texture*, sf::Color *col*, const float *x*, const float *y*, const int *direction*)

##### Аргументы:

|  |  |
| --- | --- |
| *texture* | - текстура для спрайта обьекта |
| *col* | - цвет обьекта |
| *x* | - начальная координата по оси x |
| *y* | - начальная координата по оси y |
| *direction* | - начальное направление движения обьекта |

#### Car::~Car ()

Дeструктор - обнуляет поля объекта

### Методы

#### int Car::carAround (const Map & *map*, const bool *crash*)

Метод определения возможности двигаться в заданном направлении, проверяет возможность нахождения другого обьекта данного класса по координатам действующего обьекта

##### Аргументы:

|  |  |
| --- | --- |
| *map* | - карта |
| *crash* | - возможность аварии |

##### Возвращает:

TRUE - можно двигаться

FALSE - движение запрещено

позицию обьекта **Car** в векторе, хранящем все обьекты **Car**, движение разрешено

#### bool Car::crashAround () const

Метод определения возможности двигаться в заданном направлении, проверяет возможность нахождения обьекта **Crash** по координатам обьекта **Car**

##### Возвращает:

true - можно двигаться

false - движение запрещено

#### bool Car::entryExit (const Map & *map*)

Метод анамации вьезда на карту и выезда с нее обьекта

##### Аргументы:

|  |  |
| --- | --- |
| *map* | - карта |

##### Возвращает:

true - вьезд / выезд произошел

false - вьезда / выезда не было

#### std::vector< int > Car::freeDirections (const Map & *map*)

Метод определения возможных направлений движения обьекта

##### Аргументы:

|  |  |
| --- | --- |
| *map* | - карта |

##### Возвращает:

вектор возможных направлений движения обьекта

#### int Car::getBeginDirection (const Map & *map*)

Метод, который находит и возвращает направление движения обьекта, когда он только создан

##### Аргументы:

|  |  |
| --- | --- |
| *map* | - карта |

##### Возвращает:

начальное направление движения обьекта

#### bool Car::getChangeDir () const

Метод получения значения переменной, отвечающей за смену направления на данном перекрестке

##### Возвращает:

поле ChangeDir

#### int Car::getDirection () const

Метод получения значения направления

##### Возвращает:

поле Direction

#### bool Car::getLife () const

Метод получения значения жизни

##### Возвращает:

поле Life

#### float Car::getSpeed () const

Метод получения значения скорости

##### Возвращает:

скорость

#### int Car::getStopDirection () const

Метод получения значения времено запрещенного направления

##### Возвращает:

поле StopDirection

#### float Car::getX () const

Метод получения значения координаты по оси Х

##### Возвращает:

координата по оси Х

#### float Car::getY () const

Метод получения значения координаты по оси Y

##### Возвращает:

координата по оси Y

#### void Car::go (sf::RenderWindow & *window*, const Map & *map*, bool & *crash*, bool & *pause*)

Метод движения и вывода в окно спрайта обьекта

##### Аргументы:

|  |  |
| --- | --- |
| *window* | - окно |
| *map* | - карта |
| *crash* | - возможность аварии |

#### void Car::leaveCrossroad (const Map & *map*)

Метод изменяющий поля СhangeDir и Crossroad при покидании обьекта данного класса перекрестка

#### bool Car::lightAround (const Map & *map*) const

Метод определения возможности двигаться в заданном направлении, проверяет возможность нахождения обьекта **TrafficLight** по координатам обьекта **Car**

##### Аргументы:

|  |  |
| --- | --- |
| *map* | - карта |

##### Возвращает:

true - можно двигаться

false - движение запрещено

#### int Car::mdX () const

Метод переводa приращения dX обьекта в доступное для подстановки в карту

##### Возвращает:

округленное приращение по оси Х

#### int Car::mdY () const

Метод перевод приращения dY обьекта в доступное для подстановки в карту

##### Возвращает:

округленное приращениепо оси Y

#### int Car::mX () const

Метод переводa координаты X обьекта в доступную для подстановки в карту

##### Возвращает:

округленная координата по оси Х

#### int Car::mY () const

Метод переводa координаты Y обьекта в доступную для подстановки в карту

##### Возвращает:

округленная координата по оси Y

#### float Car::okrugl (const float *xy*) const

Метод перевода координат в доступные для подстановки в карту, округление в меньшую сторону, если направление положительное, в большую, если отрицательное

##### Аргументы:

|  |  |
| --- | --- |
| *xy* | - координата |

##### Возвращает:

округленная координата

#### void Car::setDirection (const int *direct*)

Метод, устанавливающий значение направления

##### Аргументы:

|  |  |
| --- | --- |
| *direct* | - новое значение направления |

#### void Car::setSizeSprite ()

Метод, устанавливающий размер спрайта

#### void Car::setSpeed (const float *newSpeed*)

Метод, устанавливающий значение скорости

##### Аргументы:

|  |  |
| --- | --- |
| *newSpeed* | - новое значение скорости |

#### void Car::signAround (const Map & *map*)

Метод определения возможности нахождения обьекта Sign, меняет скорость обьекта (Speed) на значение, заданное в Sign (Limit)

##### Аргументы:

|  |  |
| --- | --- |
| *map* | - карта |

#### std::vector< Car > & Car::Vec ()[static]

Метод для получения статического вектора класса

##### Возвращает:

AllCars - статический вектор, хранящий все обьекты данного класса

### Данные класса

#### std::vector< Car > Car::AllCars[static], [private]

Вектор, хранящий все обьекты данного класса

#### bool Car::ChangeDir[private]

Смена направления

#### sf::Color Car::Color[private]

Цвет

#### bool Car::Crossroad[private]

Нахождение авто на перекрестке

#### int Car::Direction[private]

Направление

#### float Car::dX[private]

Приращение по оси Х

#### float Car::dY[private]

Приращение по оси У

#### int Car::EntryExit[private]

Переменная, отвечающая за анимация вьезда на карту и выезда с нее обьекта **Car**.

#### bool Car::Life[private]

Жизнь

#### float Car::Speed[private]

Скорость

#### sf::Sprite Car::Sprite[private]

Спрайт обьекта

#### int Car::StopDirection[private]

Направление, поворот в которое, запрещен

#### float Car::X[private]

Координата по оси Х

#### float Car::Y[private]

Координата по оси Y.

#### Объявления и описания членов классов находятся в файлах:

**Car.h**

**Car.cpp**

## Класс Crash

Авария

#include <Crash.h>

### Открытые члены

**Crash** (sf::RenderWindow &window, const float x1, const float y1, const float x2, const float y2)

**~Crash** ()

*Деструктор - обнуляет поля обьекта*

int **work** (sf::RenderWindow &window)

float **getX1** () const

float **getY1** () const

float **getX2** () const

float **getY2** () const

### Открытые статические члены

static std::vector< **Crash** > & **Vec** ()

### Закрытые данные

sf::CircleShape **Red1**

*Красный круг для анимации взрыва, расположенный по координатам 1-го автомобиля*

sf::CircleShape **Red2**

*Красный круг для анимации взрыва, расположенный по координатам 2-го автомобиля*

sf::CircleShape **Yellow1**

*Желтый круг для анимации взрыва, расположенный по координатам 1-го автомобиля*

sf::CircleShape **Yellow2**

*Желтый круг для анимации взрыва, расположенный по координатам 2-го автомобиля*

int **Time**

*Время анимации аварии*

float **X1**

*Координата по оси Х взрыва 1-го автомобиля*

float **Y1**

*Координата по оси У взрыва 1-го автомобиля*

float **X2**

*Координата по оси Х взрыва 2-го автомобиля*

float **Y2**

*Координата по оси У взрыва 2-го автомобиля*

float **RadiusRed**

*Радиус красного круга*

float **RadiusYellow**

*Радиус желтого круга*

### Закрытые статические данные

static std::vector< **Crash** > **AllCrash**

*Вектор, хранящий все обьекты данного класса*

### Подробное описание

Авария

### Конструктор(ы)

#### Crash::Crash (sf::RenderWindow & *window*, const float *x1*, const float *y1*, const float *x2*, const float *y2*)

##### Аргументы:

|  |  |
| --- | --- |
| *window* | - окно |
| *x1* | - координата 1 по оси Х |
| *y1* | - координата 1 по оси Y |
| *x2* | - координата 2 по оси Х |
| *y2* | - координата 2 по оси Y |

#### Crash::~Crash ()

Деструктор - обнуляет поля обьекта

### Методы

#### float Crash::getX1 () const

Метод получения координаты 1 по оси Х

##### Возвращает:

X1 - координата 1 по оси Х

#### float Crash::getX2 () const

Метод получения координаты 2 по оси Х

##### Возвращает:

X2 - координата 2 по оси X

#### float Crash::getY1 () const

Метод получения координаты 1 по оси Y

##### Возвращает:

Y1 - координата 1 по оси Y

#### float Crash::getY2 () const

Метод получения координаты 2 по оси Y

##### Возвращает:

Y2 - координата 2 по оси Y

#### std::vector< Crash > & Crash::Vec ()[static]

Метод для получения статического вектора класса

##### Возвращает:

AllCrash - статический вектор, хранящий все обьекты данного класса

#### int Crash::work (sf::RenderWindow & *window*)

Метод анимации врзыва

##### Аргументы:

|  |  |
| --- | --- |
| *window* | - окно |

##### Возвращает:

время существования обьекта **Crash**

### Данные класса

#### std::vector< Crash > Crash::AllCrash[static], [private]

Вектор, хранящий все обьекты данного класса

#### float Crash::RadiusRed[private]

Радиус красного круга

#### float Crash::RadiusYellow[private]

Радиус желтого круга

#### sf::CircleShape Crash::Red1[private]

Красный круг для анимации взрыва, расположенный по координатам 1-го автомобиля

#### sf::CircleShape Crash::Red2[private]

Красный круг для анимации взрыва, расположенный по координатам 2-го автомобиля

#### int Crash::Time[private]

Время анимации аварии

#### float Crash::X1[private]

Координата по оси Х взрыва 1-го автомобиля

#### float Crash::X2[private]

Координата по оси Х взрыва 2-го автомобиля

#### float Crash::Y1[private]

Координата по оси У взрыва 1-го автомобиля

#### float Crash::Y2[private]

Координата по оси У взрыва 2-го автомобиля

#### sf::CircleShape Crash::Yellow1[private]

Желтый круг для анимации взрыва, расположенный по координатам 1-го автомобиля

#### sf::CircleShape Crash::Yellow2[private]

Желтый круг для анимации взрыва, расположенный по координатам 2-го автомобиля

#### Объявления и описания членов классов находятся в файлах:

**Crash.h**

**Crash.cpp**

## Класс Map

Карта

#include <Map.h>

### Открытые члены

**Map** (const char scale)

**Map** (const **vector2d** tilemap)

void **work** (sf::RenderWindow &window)

**~Map** ()

*Деструктор - обнуляет поля*

size\_t **getHeight** () const

size\_t **getWidth** () const

std::vector< std::string > **getTM** () const

char **getScale** () const

void **createRoadSign** (const size\_t y, const size\_t x)

void **createRoad** (const size\_t y, const size\_t x)

### Закрытые данные

sf::Sprite **Sprite**

*Спрайт обьекта*

sf::Texture **Texture**

*Тeкстура для спрайта*

size\_t **Height**

*Высота карты*

size\_t **Width**

*Ширина карты*

**vector2d** **TileMap**

*Вектор, определяющий карту*

char **Scale**

*Маркер, определяющий тип карты*

### Подробное описание

Карта

### Конструктор(ы)

#### Map::Map (const char *scale*)

##### Аргументы:

|  |  |
| --- | --- |
| *scale* | - маркер, определяющий тип карты |

#### Map::Map (const vector2d *tilemap*)

##### Аргументы:

|  |  |
| --- | --- |
| *tilemap* | - схема карты |

#### Map::~Map ()

Деструктор - обнуляет поля

### Методы

#### void Map::createRoad (const size\_t *y*, const size\_t *x*)

Метод для выбора спрайта дороги

##### Аргументы:

|  |  |
| --- | --- |
| *x* | - координата по оси Х |
| *у* | - координата по оси У |

#### void Map::createRoadSign (const size\_t *y*, const size\_t *x*)

Метод для выбора спрайта обьекта **RoadSign**

##### Аргументы:

|  |  |
| --- | --- |
| *x* | - координата по оси Х |
| *у* | - координата по оси У |

#### size\_t Map::getHeight () const

Метод для получения высоты карты

##### Возвращает:

Height - высота

#### char Map::getScale () const

Метод для получения маркера, определяещего тип и размер карты

##### Возвращает:

Scale

#### std::vector< std::string > Map::getTM () const

Метод для получения вектора, определяющего карту

##### Возвращает:

TileMap

#### size\_t Map::getWidth () const

Метод для получения ширины карты

##### Возвращает:

Weight - ширина

#### void Map::work (sf::RenderWindow & *window*)

Метод вывода спрайта карты

##### Аргументы:

|  |  |
| --- | --- |
| *window* | - окно |

### Данные класса

#### size\_t Map::Height[private]

Высота карты

#### char Map::Scale[private]

Маркер, определяющий тип карты

#### sf::Sprite Map::Sprite[private]

Спрайт обьекта

#### sf::Texture Map::Texture[private]

Тeкстура для спрайта

#### vector2d Map::TileMap[private]

Вектор, определяющий карту

#### size\_t Map::Width[private]

Ширина карты

#### Объявления и описания членов классов находятся в файлах:

**Map.h**

**Map.cpp**

## Класс RoadSign

Дорожный знак

#include <RoadSign.h>

### Открытые члены

**RoadSign** (const **Map** &map, const float x, const float y)

**~RoadSign** ()

*Обнуляет поля обьекта*

float **getLimit** () const

*Метод получения значения ограничения скорости*

float **getX** () const

*Метод получения значения координаты по оси Х*

float **getY** () const

*Метод получения значения координаты по оси Y.*

### Открытые статические члены

static std::vector< **RoadSign** > & **Vec** ()

*Метод для получения статического вектора класса*

### Закрытые данные

float **Limit**

*Ограничение скорости*

float **X**

*Координата по оси Х*

float **Y**

*Координата по оси Y.*

### Закрытые статические данные

static std::vector< **RoadSign** > **AllRoadSign**

*Вектор, хранящий все обьекты данного класса*

### Подробное описание

Дорожный знак

### Конструктор(ы)

#### RoadSign::RoadSign (const Map & *map*, const float *x*, const float *y*)

##### Аргументы:

|  |  |
| --- | --- |
| *map* | - карта |
| *x* | - координата по оси Х |
| *y* | - координата по оси Y |

#### RoadSign::~RoadSign ()

Обнуляет поля обьекта

### Методы

#### float RoadSign::getLimit () const

Метод получения значения ограничения скорости

##### Возвращает:

ограничение скорости

#### float RoadSign::getX () const

Метод получения значения координаты по оси Х

##### Возвращает:

координата по оси Х

#### float RoadSign::getY () const

Метод получения значения координаты по оси Y.

##### Возвращает:

координата по оси Y

#### std::vector< RoadSign > & RoadSign::Vec ()[static]

Метод для получения статического вектора класса

##### Возвращает:

AllRoadSign - статический вектор, хранящий все обьекты данного класса

### Данные класса

#### std::vector< RoadSign > RoadSign::AllRoadSign[static], [private]

Вектор, хранящий все обьекты данного класса

#### float RoadSign::Limit[private]

Ограничение скорости

#### float RoadSign::X[private]

Координата по оси Х

#### float RoadSign::Y[private]

Координата по оси Y.

#### Объявления и описания членов классов находятся в файлах:

**RoadSign.h**

**RoadSign.cpp**

## Класс TrafficLight

Светофор

#include <TrafficLight.h>

### Открытые члены

**TrafficLight** (const **Map** &map, const float x, const float y)

**~TrafficLight** ()

*Деструктор - обнуляет поля обьекта*

void **work** (const int clock, sf::RenderWindow &window)

sf::Color **getColor** () const

float **getX** () const

float **getY** () const

int **getTimeChange** () const

bool **getLastColor** () const

### Открытые статические члены

static std::vector< **TrafficLight** > & **Vec** ()

### Закрытые данные

sf::CircleShape **Circle**

*Графическое изображение обьекта - круг*

sf::CircleShape **BlackCircle**

*Графическое изображение обьекта - обводка вокруг круга*

sf::Color **Color**

*Цвет*

float **X**

*Координата по оси Х*

float **Y**

*Координата по оси У*

bool **LastColor**

*Цвет до смены на желтый : true - если был зеленый, false - если был красный*

int **TimeChange**

*Время изменения цвета*

### Закрытые статические данные

static std::vector< **TrafficLight** > **AllTrafficLight**

*Вектор, хранящий все обьекты данного класса*

### Подробное описание

Светофор

### Конструктор(ы)

#### TrafficLight::TrafficLight (const Map & *map*, const float *x*, const float *y*)

##### Аргументы:

|  |  |
| --- | --- |
| *map* | - карта |
| *x* | - координата по оси Х |
| *y* | - координата по оси Y |

#### TrafficLight::~TrafficLight ()

Деструктор - обнуляет поля обьекта

### Методы

#### sf::Color TrafficLight::getColor () const

Метод для получения значения цвета обьекта

##### Возвращает:

Color - цвет обьекта

#### bool TrafficLight::getLastColor () const

Метод для получения значения поля, хранящего прдыдущий цвет светофора (перед сменой на желтый)

##### Возвращает:

поле LastColor

#### int TrafficLight::getTimeChange () const

Метод для получения значения времени от последнего изменения цвета обьекта

##### Возвращает:

TimeChange

#### float TrafficLight::getX () const

Метод для получения значения координаты по оси Х

##### Возвращает:

X - координата по оси X

#### float TrafficLight::getY () const

Метод для получения значения координаты по оси Y

##### Возвращает:

Y - координата по оси Y

#### std::vector< TrafficLight > & TrafficLight::Vec ()[static]

Метод для получения статического вектора класса

##### Возвращает:

AllTrafficLight - статический вектор, хранящий все обьекты данного класса

#### void TrafficLight::work (const int *clock*, sf::RenderWindow & *window*)

Метод, запускающий смену цвета обьекта в зависимости от времени

##### Аргументы:

|  |  |
| --- | --- |
| *clock* | - текущее время |
| *window* | - окно |

### Данные класса

#### std::vector< TrafficLight > TrafficLight::AllTrafficLight[static], [private]

Вектор, хранящий все обьекты данного класса

#### sf::CircleShape TrafficLight::BlackCircle[private]

Графическое изображение обьекта - обводка вокруг круга

#### sf::CircleShape TrafficLight::Circle[private]

Графическое изображение обьекта - круг

#### sf::Color TrafficLight::Color[private]

Цвет

#### bool TrafficLight::LastColor[private]

Цвет до смены на желтый : true - если был зеленый, false - если был красный

#### int TrafficLight::TimeChange[private]

Время изменения цвета

#### float TrafficLight::X[private]

Координата по оси Х

#### float TrafficLight::Y[private]

Координата по оси У

#### Объявления и описания членов классов находятся в файлах:

**TrafficLight.h**

**TrafficLight.cpp**

## Класс View

Вид

#include <View.h>

### Открытые члены

**View** (const **Map** &map)

void **work** (sf::RenderWindow &window, const float time)

**~View** ()

### Закрытые данные

sf::View **Camera**

*Камера вида*

sf::Vector3f **Size**

*Размер*

### Подробное описание

Вид

### Конструктор(ы)

#### View::View (const Map & *map*)

Инициализирует поля обьекта

##### Аргументы:

|  |  |
| --- | --- |
| *map* | - карта |

#### View::~View ()

### Методы

#### void View::work (sf::RenderWindow & *window*, const float *time*)

Метод движения, приближения и отдаления камеры вида

##### Аргументы:

|  |  |
| --- | --- |
| *window* | - окно |
| *time* | - время |

### Данные класса

#### sf::View View::Camera[private]

Камера вида

#### sf::Vector3f View::Size[private]

Размер

#### Объявления и описания членов классов находятся в файлах:

**View.h**

**View.cpp**

## Файл Function.cpp

#include "Function.h"

### Функции

bool **IsRoadSign** (const char marker)

bool **IsRoad** (const char marker)

float **Mod** (const float num)

### Функции

#### bool IsRoad (const char *marker*)

Функция, проверяющая является ли переданный аргумент маркером дороги

##### Аргументы:

|  |  |
| --- | --- |
| *marker* | - символ, который надо проверить |

##### Возвращает:

true - да, является

#### bool IsRoadSign (const char *marker*)

Функция, проверяющая является ли переданный аргумент маркером дрожного знака

##### Аргументы:

|  |  |
| --- | --- |
| *marker* | - символ, который надо проверить |

##### Возвращает:

true - да, является

#### float Mod (const float *num*)

Функция нахождения модуля значения переменной

##### Аргументы:

|  |  |
| --- | --- |
| *num* | - переменная, модуль которой надо найти |

##### Возвращает:

модуль данной переменной

## Файл Function.h

#include "Include.h"

### Функции

bool **IsRoadSign** (const char marker)

bool **IsRoad** (const char marker)

float **Mod** (const float num)

### Функции

#### bool IsRoad (const char *marker*)

Функция, проверяющая является ли переданный аргумент маркером дороги

##### Аргументы:

|  |  |
| --- | --- |
| *marker* | - символ, который надо проверить |

##### Возвращает:

true - да, является

#### bool IsRoadSign (const char *marker*)

Функция, проверяющая является ли переданный аргумент маркером дрожного знака

##### Аргументы:

|  |  |
| --- | --- |
| *marker* | - символ, который надо проверить |

##### Возвращает:

true - да, является

#### float Mod (const float *num*)

Функция нахождения модуля значения переменной

##### Аргументы:

|  |  |
| --- | --- |
| *num* | - переменная, модуль которой надо найти |

##### Возвращает:

модуль данной переменной

## Файл FunctionForMain.cpp

#include "FunctionForMain.h"

#include "FUnction.h"

### Функции

void **CheckSpeed** ()

*Функция для проверки корректности скорости*

void **CreateTrLightAndRSign** (const **Map** &map)

sf::Color & **chooseColor** ()

*Функция, возвращающая цвет, выбранный с помощью функции rand()*

void **AddCar** (const sf::Texture &texture, const **Map** &map, sf::Vector3i &currentClock)

bool **IsCrash** (sf::Vector3i &currentClock, const **Map** &map)

void **Work** (sf::RenderWindow &window, **Map** &map, **View** &view, const sf::Texture &texture1, const sf::Texture &texture2, sf::Vector3i &currentClock, float time, bool &crash, bool &pause)

### Функции

#### void AddCar (const sf::Texture & *texture*, const Map & *map*, sf::Vector3i & *currentClock*)

#### void CheckSpeed ()

Функция для проверки корректности скорости

#### sf::Color& chooseColor ()

Функция, возвращающая цвет, выбранный с помощью функции rand()

#### void CreateTrLightAndRSign (const Map & *map*)

Функция для создания обьектов **TrafficLight** и **RoadSign**

##### Аргументы:

|  |  |
| --- | --- |
| *map* | - карта |

#### bool IsCrash (sf::Vector3i & *currentClock*, const Map & *map*)

#### void Work (sf::RenderWindow & *window*, Map & *map*, View & *view*, const sf::Texture & *texture1*, const sf::Texture & *texture2*, sf::Vector3i & *currentClock*, float *time*, bool & *crash*, bool & *pause*)

Функция для запуска основных методов движения, изменения и создания возможных обьектов

##### Аргументы:

|  |  |
| --- | --- |
| *window* | - окно |
| *map* | - карта |
| *view* | - камера вида |
| *texture1* | - текстура для спрайта обьекта **Car** |
| *texture2* | - текстура для спрайта обьекта **Car** |
| *currentClock* | - вектор, в котором в поле х хранится текущее время, в поле y - время последнего добавления новой машины, в поле z - время последней аварии |
| *time* | - время для правильной работы камеры вида |

## Файл FunctionForMain.h

#include "TrafficLight.h"

#include "RoadSign.h"

#include "UnitTest.h"

#include "Include.h"

#include "Crash.h"

#include "View.h"

#include "Car.h"

#include "Map.h"

### Функции

void **CheckSpeed** ()

*Функция для проверки корректности скорости*

void **CreateTrLightAndRSign** (const **Map** &map)

void **AddCar** (const **Map** &map, const sf::Texture &texture, sf::Vector3i &currentClock)

bool **IsCrash** (const **Map** &map, sf::Vector3i &currentClock)

void **Work** (sf::RenderWindow &window, **Map** &map, **View** &view, const sf::Texture &texture1, const sf::Texture &texture2, sf::Vector3i &currentClock, float time, bool &crash, bool &pause)

sf::Color & **chooseColor** ()

*Функция, возвращающая цвет, выбранный с помощью функции rand()*

### Функции

#### void AddCar (const Map & *map*, const sf::Texture & *texture*, sf::Vector3i & *currentClock*)

Функция для добавления нового обьекта **Car**

##### Аргументы:

|  |  |
| --- | --- |
| *map* | - карта |
| *texture* | - текстура для спрайта обьекта |
| *currentClock* | - вектор, в котором в поле х хранится текущее время, в поле y - время последнего добавления новой машины |

#### void CheckSpeed ()

Функция для проверки корректности скорости

#### sf::Color& chooseColor ()

Функция, возвращающая цвет, выбранный с помощью функции rand()

#### void CreateTrLightAndRSign (const Map & *map*)

Функция для создания обьектов **TrafficLight** и **RoadSign**

##### Аргументы:

|  |  |
| --- | --- |
| *map* | - карта |

#### bool IsCrash (const Map & *map*, sf::Vector3i & *currentClock*)

Функция для определения возможности аварии

##### Аргументы:

|  |  |
| --- | --- |
| *map* | - карта |
| *currentClock* | - вектор, в котором в поле х хранится текущее время, в поле z - время последней аварии |

##### Возвращает:

true - если авария возможна

false - если не возможна

#### void Work (sf::RenderWindow & *window*, Map & *map*, View & *view*, const sf::Texture & *texture1*, const sf::Texture & *texture2*, sf::Vector3i & *currentClock*, float *time*, bool & *crash*, bool & *pause*)

Функция для запуска основных методов движения, изменения и создания возможных обьектов

##### Аргументы:

|  |  |
| --- | --- |
| *window* | - окно |
| *map* | - карта |
| *view* | - камера вида |
| *texture1* | - текстура для спрайта обьекта **Car** |
| *texture2* | - текстура для спрайта обьекта **Car** |
| *currentClock* | - вектор, в котором в поле х хранится текущее время, в поле y - время последнего добавления новой машины, в поле z - время последней аварии |
| *time* | - время для правильной работы камеры вида |

## Файл Include.h

#include <SFML/Graphics.hpp>

#include <exception>

#include <iostream>

#include <string>

#include <vector>

#include <ctime>

### Макросы

#define **TEXTURE\_CAR1**  "images/car.png"

*Постоянная текстура для спрайта обьекта* ***Car****.*

#define **TEXTURE\_CAR2**  "images/sport\_car.png"

*Постоянная спорт - текстура для спрайта обьекта* ***Car****.*

#define **TEXTURE\_MAP**  "images/map.png"

*Постоянная текстура для спрайта обьекта* ***Map****.*

#define **FLOAT**(a)  static\_cast<float>(a)

*Сокращение длинного пребразования типов*

#define **INT**(a)  static\_cast<int>(a)

*Сокращение длинного пребразования типов*

#define **vector2d**  std::vector<std::string>

*Сокращение длинного названия типа*

### Перечисления

enum { **X\_PLUS**, **Y\_PLUS**, **X\_MINUS**, **Y\_MINUS** }*Направления движения обьекта* ***Car****.*

### Переменные

const int **PIXEL** = 120

*Каждый квадрат имеет размер 120 \* 120.*

const float **MAX\_SPEED1** = 3.f

*Mаксимальная скорость для Release для MIDDLE\_MAP или LITTLE\_MAP.*

const float **MIN\_SPEED1** = 2.f

*Mинимальная скорость для Release для MIDDLE\_MAP или LITTLE\_MAP.*

const float **MAX\_SPEED2** = 60.f

*Mаксимальная скорость для Release для BIG\_MAP.*

const float **MIN\_SPEED2** = 40.f

*Mинимальная скорость для Release для BIG\_MAP.*

const size\_t **CRASH\_TIME** = 450

*Для анимации аварии*

const size\_t **CRASH\_END** = 4000

*Длительность аварии*

const size\_t **LIMIT\_CRASH1** = 5500

*Период между авариями*

const size\_t **LIMIT\_CRASH2** = 3500

*Период между авариями*

const float **HEIGHT\_WINDOW** = 480.f

*Ширина окна*

const float **WIDTH\_WINDOW** = 640.f

*Высота окна*

const size\_t **TIME\_CHANGE\_LIGHT** = 300

*Время смены цвета светофора*

const size\_t **TIME\_YELLOW\_LIGHT** = 1000

*Время смены цвета светофора*

const size\_t **LIMIT\_ADD\_CAR** = 400

*Период между добавлениями новых машин*

const size\_t **BIG\_NUM\_CAR** = 50

*Количество машин, при достижении которого возможны аварии*

const size\_t **LARGE\_NUM\_CAR** = 100

*Количество машин, при достижении которого частота аварий увеличивается*

const size\_t **MAX\_NUM\_CAR** = 120

*Максимальное кол-во машин для маленькой карты*

const char **BIG\_MAP** = 'a'

*Маркер для создания большой карты*

const char **MIDDLE\_MAP** = 'b'

*Маркер для создания средней карты*

const char **LITTLE\_MAP** = 'c'

*Маркер для создания маленькой карты*

const char **CUSTOM** = 'u'

*Маркер пользовательской схемы карты*

const char **ROAD** = 'r'

*Символ, обозначающий в обьекте* ***Map*** *дорогу*

const char **CROSSROAD** = 'p'

*Символ, обозначающий в обьекте* ***Map*** *перекресток*

const char **EXIT** = 'e'

*Символ, обозначающий в обьекте* ***Map*** *выезд*

const char **SPEED\_LIMIT\_ON** = 'k'

*Символ, обозначающий в обьекте* ***Map*** *знак дорожного движения, ограничивающий скорость*

const char **SPEED\_LIMIT\_OFF** = 'y'

*Символ, обозначающий в обьекте* ***Map*** *знак дорожного движения, снимающий ограничение по скорости*

const int **UNKNOWN\_VALUE** = -1

*Временное значение для поля обьекта*

const int **TRUE** = -1

*Истина для некоторых методов обьекта*

const int **FALSE** = -2

*Ложь для некоторых методов обьекта*

### Макросы

#### #define FLOAT( a)  static\_cast<float>(a)

Сокращение длинного пребразования типов

#### #define INT( a)  static\_cast<int>(a)

Сокращение длинного пребразования типов

#### #define TEXTURE\_CAR1  "images/car.png"

Постоянная текстура для спрайта обьекта **Car**.

#### #define TEXTURE\_CAR2  "images/sport\_car.png"

Постоянная спорт - текстура для спрайта обьекта **Car**.

#### #define TEXTURE\_MAP  "images/map.png"

Постоянная текстура для спрайта обьекта **Map**.

#### #define vector2d  std::vector<std::string>

Сокращение длинного названия типа

### Перечисления

#### anonymous enum

Направления движения обьекта **Car**.

##### Элементы перечислений:

|  |  |
| --- | --- |
| X\_PLUS |  |
| Y\_PLUS |  |
| X\_MINUS |  |
| Y\_MINUS |  |

### Переменные

#### const char BIG\_MAP = 'a'

Маркер для создания большой карты

#### const size\_t BIG\_NUM\_CAR = 50

Количество машин, при достижении которого возможны аварии

#### const size\_t CRASH\_END = 4000

Длительность аварии

#### const size\_t CRASH\_TIME = 450

Для анимации аварии

#### const char CROSSROAD = 'p'

Символ, обозначающий в обьекте **Map** перекресток

#### const char CUSTOM = 'u'

Маркер пользовательской схемы карты

#### const char EXIT = 'e'

Символ, обозначающий в обьекте **Map** выезд

#### const int FALSE = -2

Ложь для некоторых методов обьекта

#### const float HEIGHT\_WINDOW = 480.f

Ширина окна

#### const size\_t LARGE\_NUM\_CAR = 100

Количество машин, при достижении которого частота аварий увеличивается

#### const size\_t LIMIT\_ADD\_CAR = 400

Период между добавлениями новых машин

#### const size\_t LIMIT\_CRASH1 = 5500

Период между авариями

#### const size\_t LIMIT\_CRASH2 = 3500

Период между авариями

#### const char LITTLE\_MAP = 'c'

Маркер для создания маленькой карты

#### const size\_t MAX\_NUM\_CAR = 120

Максимальное кол-во машин для маленькой карты

#### const float MAX\_SPEED1 = 3.f

Mаксимальная скорость для Release для MIDDLE\_MAP или LITTLE\_MAP.

#### const float MAX\_SPEED2 = 60.f

Mаксимальная скорость для Release для BIG\_MAP.

#### const char MIDDLE\_MAP = 'b'

Маркер для создания средней карты

#### const float MIN\_SPEED1 = 2.f

Mинимальная скорость для Release для MIDDLE\_MAP или LITTLE\_MAP.

#### const float MIN\_SPEED2 = 40.f

Mинимальная скорость для Release для BIG\_MAP.

#### const int PIXEL = 120

Каждый квадрат имеет размер 120 \* 120.

#### const char ROAD = 'r'

Символ, обозначающий в обьекте **Map** дорогу

#### const char SPEED\_LIMIT\_OFF = 'y'

Символ, обозначающий в обьекте **Map** знак дорожного движения, снимающий ограничение по скорости

#### const char SPEED\_LIMIT\_ON = 'k'

Символ, обозначающий в обьекте **Map** знак дорожного движения, ограничивающий скорость

#### const size\_t TIME\_CHANGE\_LIGHT = 300

Время смены цвета светофора

#### const size\_t TIME\_YELLOW\_LIGHT = 1000

Время смены цвета светофора

#### const int TRUE = -1

Истина для некоторых методов обьекта

#### const int UNKNOWN\_VALUE = -1

Временное значение для поля обьекта

#### const float WIDTH\_WINDOW = 640.f

## 

## Файл UnitTest.cpp

#include "TrafficLight.h"

#include "RoadSign.h"

#include "UnitTest.h"

#include "Crash.h"

#include "Car.h"

#include "Map.h"

### Функции

template<typename T > void **Assert** (const T true\_value, const T check\_value, const std::string functionName)

bool **UTest\_Car** (const **Map** &map, sf::RenderWindow &window, std::vector< bool > &was)

bool **UTest\_Classes** (const **Map** &map, sf::RenderWindow &window, std::vector< bool > &was)

### Функции

#### template<typename T > void Assert (const T *true\_value*, const T *check\_value*, const std::string *functionName*)

Шаблонная функция, сравнивающая два значения

##### Аргументы:

|  |  |
| --- | --- |
| *true\_value* | - изначально известное, правильное значение |
| *chack\_value* | - значение, которое надо проверить |
| *functionName* | - имя функции/метода, результат которой/которого проверяется |

#### bool UTest\_Car (const Map & *map*, sf::RenderWindow & *window*, std::vector< bool > & *was*)

Функция, тестирующая методы класса **Car**

##### Аргументы:

|  |  |
| --- | --- |
| *map* | - карта |
| *window* | - окно |
| *was* | - вектор, хранящий значение истина/ложь, в котором каждый элемент отвечает за тестирование одного метода |

##### Возвращает:

true - все тесты были пройдены, исход каждого выведен на экран

false - не все тесты были пройдены

#### bool UTest\_Classes (const Map & *map*, sf::RenderWindow & *window*, std::vector< bool > & *was*)

Функция, тестирующая методы классов **TrafficLight**, **RoadSign** и **Crash**

##### Аргументы:

|  |  |
| --- | --- |
| *map* | - карта |
| *window* | - окно |
| *was* | - вектор, хранящий значение истина/ложь, в котором каждый элемент отвечает за тестирование одного метода |

##### Возвращает:

true - все тесты были пройдены, исход каждого выведен на экран

false - не все тесты были пройдены

## Файл UnitTest.h

#include "Include.h"

### Функции

template<typename T > void **Assert** (const T true\_value, const T check\_value, const std::string functionName)

bool **UTest\_Car** (const **Map** &map, sf::RenderWindow &window, std::vector< bool > &was)

bool **UTest\_Classes** (const **Map** &map, sf::RenderWindow &window, std::vector< bool > &was)

### Функции

#### template<typename T > void Assert (const T *true\_value*, const T *check\_value*, const std::string *functionName*)

Шаблонная функция, сравнивающая два значения

##### Аргументы:

|  |  |
| --- | --- |
| *true\_value* | - изначально известное, правильное значение |
| *chack\_value* | - значение, которое надо проверить |
| *functionName* | - имя функции/метода, результат которой/которого проверяется |

#### bool UTest\_Car (const Map & *map*, sf::RenderWindow & *window*, std::vector< bool > & *was*)

Функция, тестирующая методы класса **Car**

##### Аргументы:

|  |  |
| --- | --- |
| *map* | - карта |
| *window* | - окно |
| *was* | - вектор, хранящий значение истина/ложь, в котором каждый элемент отвечает за тестирование одного метода |

##### Возвращает:

true - все тесты были пройдены, исход каждого выведен на экран

false - не все тесты были пройдены

#### bool UTest\_Classes (const Map & *map*, sf::RenderWindow & *window*, std::vector< bool > & *was*)

Функция, тестирующая методы классов **TrafficLight**, **RoadSign** и **Crash**

##### Аргументы:

|  |  |
| --- | --- |
| *map* | - карта |
| *window* | - окно |
| *was* | - вектор, хранящий значение истина/ложь, в котором каждый элемент отвечает за тестирование одного метода |

##### Возвращает:

true - все тесты были пройдены, исход каждого выведен на экран

false - не все тесты были пройдены