ZPR 23L Projekt - EvoRacer

Generated by Doxygen 1.9.7

1 Class Index 1

1.1 Class List . . . 1

2 File Index 3

2.1 File List . . . 3

3 Class Documentation 5

3.1 Box Struct Reference . . . 5

3.1.1 Detailed Description . . . 6

3.1.2 Member Data Documentation . . . 6

3.1.2.1 body . . . 6

3.1.2.2 color . . . 6

3.1.2.3 height . . . 6

3.1.2.4 width . . . 6

3.2 Car Class Reference . . . 7

3.2.1 Detailed Description . . . 8

3.2.2 Constructor & Destructor Documentation . . . 8

3.2.2.1 Car() . . . 8

3.2.3 Member Function Documentation . . . 10

3.2.3.1 getBackWheel() . . . 10

3.2.3.2 getBody() . . . 10

3.2.3.3 getBodyColor() . . . 11

3.2.3.4 getFrontWheel() . . . 11

3.2.3.5 getPosX() . . . 12

3.2.3.6 getPosXVec() . . . 12

3.2.3.7 getPosY() . . . 12

3.2.3.8 getPosYVec() . . . 13

3.2.3.9 getVelocity() . . . 13

3.2.3.10 getVelocityVec() . . . 13

3.2.3.11 getVelX() . . . 14

3.2.3.12 getVelY() . . . 14

3.2.3.13 setCollisionFilter() . . . 14

3.3 CarConfig Class Reference . . . 15

3.3.1 Detailed Description . . . 16

3.3.2 Member Data Documentation . . . 16

3.3.2.1 CAR_TORQUE . . . 16

3.3.2.2 CAR_VERTICES . . . 16

3.3.2.3 MAX_JOINT_LENGTH . . . 16

3.4 Chromosome Struct Reference . . . 17

3.4.1 Detailed Description . . . 17

3.4.2 Member Data Documentation . . . 18

3.4.2.1 bodyDensity . . . 18

3.4.2.2 bodyLengths . . . 18

3.4.2.3 fitness 18
3.4.2.4 wheelDensity 18
3.4.2.5 wheelRadius 18
3.5 Circle Struct Reference 19
3.5.1 Detailed Description 19
3.5.2 Member Data Documentation 19
3.5.2.1 body 19
3.5.2.2 color 20
3.5.2.3 radius 20
3.6 Config Class Reference 20
3.6.1 Detailed Description 21
3.6.2 Member Data Documentation 21
3.6.2.1 BACK_WHEEL_POS 21
3.6.2.2 CATEGORY_BITS 21
3.6.2.3 DEBUG 21
3.6.2.4 DEG_PER_RAD 21
3.6.2.5 FRICTION 21
3.6.2.6 FRONT_WHEEL_POS 21
3.6.2.7 GENERATION_TIME 22
3.6.2.8 GRAVITATIONAL_ACCELERATION 22
3.6.2.9 GROUND_PARTS_RENDERED 22
3.6.2.10 MASK_BITS 22
3.6.2.11 MAX_FPS 22
3.6.2.12 PI 22
3.6.2.13 PPM 22
3.6.2.14 SAVE_FILE_NAME 23
3.6.2.15 SAVE_TO_FILE 23
3.6.2.16 VELOCITY_ARRAY_SIZE 23
3.6.2.17 WINDOW_HEIGHT 23
3.6.2.18 WINDOW_WIDTH 23
3.7 EvolutionaryAlgorithm Class Reference 24
3.7.1 Detailed Description 25
3.7.2 Constructor & Destructor Documentation 25
3.7.2.1 EvolutionaryAlgorithm() 25
3.7.3 Member Function Documentation 25
3.7.3.1 exportPopulation() 25
3.7.3.2 getGeneration() 26
3.7.3.3 getPopulation() 26
3.7.3.4 getPopulationSize() 27
3.7.3.5 mutate() 28
3.7.3.6 nextGeneration() 29
3.7.3.7 setFitness() 30

3.7.3.8 tournamentSelection() 30
3.8 EvolutionaryAlgorithmConfig Class Reference 31
3.8.1 Detailed Description 33
3.8.2 Member Data Documentation 33
3.8.2.1 INITIAL_BODY_DENSITY_MEAN 33
3.8.2.2 INITIAL_BODY_DENSITY_VARIANCE 33
3.8.2.3 INITIAL_BODY_LENGTH_MEAN 33
3.8.2.4 INITIAL_BODY_LENGTH_VARIANCE 34
3.8.2.5 INITIAL_WHEEL_DENSITY_MEAN 34
3.8.2.6 INITIAL_WHEEL_DENSITY_VARIANCE 34
3.8.2.7 INITIAL_WHEEL_RADIUS_MEAN 34
3.8.2.8 INITIAL_WHEEL_RADIUS_VARIANCE 34
3.8.2.9 MAX_BODY_DENSITY 34
3.8.2.10 MAX_BODY_LENGTH 34
3.8.2.11 MAX_WHEEL_DENSITY 35
3.8.2.12 MAX_WHEEL_RADIUS 35
3.8.2.13 MIN_BODY_DENSITY 35
3.8.2.14 MIN_BODY_LENGTH 35
3.8.2.15 MIN_WHEEL_DENSITY 35
3.8.2.16 MIN_WHEEL_RADIUS 35
3.8.2.17 MUTATION_FACTOR_BODY_DENSITY 35
3.8.2.18 MUTATION_FACTOR_BODY_LENGTHS 36
3.8.2.19 MUTATION_FACTOR_WHEEL_DENSITY 36
3.8.2.20 MUTATION_FACTOR_WHEEL_RADIUS 36
3.8.2.21 MUTATION_RATE_BODY_DENSITY 36
3.8.2.22 MUTATION_RATE_BODY_LENGTHS 36
3.8.2.23 MUTATION_RATE_WHEEL_DENSITY 36
3.8.2.24 MUTATION_RATE_WHEEL_RADIUS 36
3.8.2.25 POPULATION_SIZE 37
3.8.2.26 TOURNAMENT_SIZE 37
3.9 MapGenConfig Class Reference 37
3.9.1 Detailed Description 38
3.9.2 Member Data Documentation 38
3.9.2.1 BG_SPRITES_COUNT 38
3.9.2.2 CAR_STARTING_X 38
3.9.2.3 CAR_STARTING_Y 38
3.9.2.4 GENERATE_DISTANCE 38
3.9.2.5 GROUND_DEGREE_DEVIATION 38
3.9.2.6 GROUND_LEG_LENGTH 39
3.9.2.7 GROUND_PART_LENGTH 39
3.9.2.8 GROUND_STARTING_X 39
3.9.2.9 GROUND_STARTING_Y 39

3.9.2.10 MAX_GROUND_DEGREE 39
3.10 Polygon Struct Reference 40
3.10.1 Detailed Description 40
3.10.2 Member Data Documentation 41
3.10.2.1 body 41
3.10.2.2 color 41
3.10.2.3 vertices 41

4 File Documentation **43**
4.1 config/CarConfig.h File Reference 43
4.1.1 Detailed Description 43
4.2 CarConfig.h 44
4.3 config/Config.h File Reference 44
4.3.1 Detailed Description 44
4.4 Config.h 45
4.5 config/EvolutionaryAlgorithmConfig.h File Reference 45
4.5.1 Detailed Description 46
4.6 EvolutionaryAlgorithmConfig.h 46
4.7 config/MapGenConfig.h File Reference 47
4.7.1 Detailed Description 47
4.8 MapGenConfig.h 48
4.9 src/Car.cc File Reference 48
4.9.1 Detailed Description 48
4.9.2 Function Documentation 49
4.9.2.1 createVertices() 49
4.10 Car.cc 49
4.11 src/Car.h File Reference 51
4.11.1 Detailed Description 52
4.11.2 Typedef Documentation 52
4.11.2.1 b2WorldPtr 52
4.11.3 Function Documentation 52
4.11.3.1 createVertices() 52
4.12 Car.h 53
4.13 src/EvolutionaryAlgorithm.cc File Reference 54
4.13.1 Detailed Description 54
4.14 EvolutionaryAlgorithm.cc 55
4.15 src/EvolutionaryAlgorithm.h File Reference 57
4.15.1 Detailed Description 58
4.16 EvolutionaryAlgorithm.h 58
4.17 src/GUI.cc File Reference 59
4.17.1 Detailed Description 59
4.17.2 Function Documentation 60

4.17.2.1 printEAInfo() 60
4.17.2.2 renderPositionPlot() 60
4.17.2.3 renderVelocityPlot() 61
4.18 GUI.cc 63
4.19 src/GUI.h File Reference 64
4.19.1 Detailed Description 65
4.19.2 Function Documentation 66
4.19.2.1 printEAInfo() 66
4.19.2.2 renderPositionPlot() 66
4.19.2.3 renderVelocityPlot() 67
4.20 GUI.h 69
4.21 src/Main.cc File Reference 70
4.21.1 Detailed Description 71
4.21.2 Typedef Documentation 71
4.21.2.1 b2WorldPtr 71
4.21.3 Function Documentation 71
4.21.3.1 main() 71
4.21.4 Variable Documentation 74
4.21.4.1 world 74
4.22 Main.cc 75
4.23 src/Render.cc File Reference 76
4.23.1 Detailed Description 77
4.23.2 Function Documentation 77
4.23.2.1 render() 77
4.23.2.2 renderCar() 79
4.23.2.3 renderCircle() 80
4.23.2.4 renderCircleDebug() 81
4.23.2.5 renderPolygon() 82
4.23.2.6 renderPolygonDebug() 83
4.24 Render.cc 83
4.25 src/Render.h File Reference 85
4.25.1 Detailed Description 86
4.25.2 Function Documentation 86
4.25.2.1 render() 86
4.25.2.2 renderCar() 88
4.25.2.3 renderCircle() 89
4.25.2.4 renderCircleDebug() 90
4.25.2.5 renderPolygon() 91
4.25.2.6 renderPolygonDebug() 92
4.26 Render.h 92
4.27 src/Shape.cc File Reference 93
4.27.1 Detailed Description 93

4.27.2 Function Documentation 94
4.27.2.1 createBox() 94
4.27.2.2 createCircle() 95
4.27.2.3 createGround() 96
4.27.2.4 createPolygon() 98
4.28 Shape.cc 99
4.29 src/Shape.h File Reference 100
4.29.1 Detailed Description 102
4.29.2 Typedef Documentation 102
4.29.2.1 b2WorldPtr 102
4.29.3 Function Documentation 102
4.29.3.1 createBox() 102
4.29.3.2 createCircle() 104
4.29.3.3 createGround() 105
4.29.3.4 createPolygon() 107
4.30 Shape.h 108
4.31 src/UserInput.cc File Reference 109
4.31.1 Detailed Description 109
4.31.2 Function Documentation 110
4.31.2.1 handleEvents() 110
4.31.2.2 handleUserInput() 111
4.32 UserInput.cc 111
4.33 src/UserInput.h File Reference 112
4.33.1 Detailed Description 113
4.33.2 Function Documentation 114
4.33.2.1 handleEvents() 114
4.33.2.2 handleUserInput() 115
4.34 UserInput.h 115
4.35 src/Utility.cc File Reference 116
4.35.1 Detailed Description 117
4.35.2 Function Documentation 117
4.35.2.1 applyAirResistance() 117
4.35.2.2 generateCar() 118
4.35.2.3 generateGround() 119
4.35.2.4 getFurthestCarPos() 120
4.35.2.5 getIdxOfGrndClstToLoc() 121
4.35.2.6 getNextGroundPartDegree() 122
4.35.2.7 getRootDir() 123
4.35.2.8 loadBGSprite() 123
4.35.2.9 loadBGSprites() 124
4.35.2.10 loadBGTextures() 125
4.35.2.11 removeCars() 126

4.35.2.12 setIcon() . . . 127
4.35.2.13 SFMLColorToImVec4() . . . 128
4.36 Utility.cc . . . 129
4.37 src/Utility.h File Reference . . . 130
4.37.1 Detailed Description . . . 132
4.37.2 Typedef Documentation . . . 132
4.37.2.1 b2WorldPtr . . . 132
4.37.3 Function Documentation . . . 132
4.37.3.1 applyAirResistance() . . . 132
4.37.3.2 generateCar() . . . 133
4.37.3.3 generateGround() . . . 134
4.37.3.4 getFurthestCarPos() . . . 135
4.37.3.5 getIdxOfGrndClstToLoc() . . . 136
4.37.3.6 getNextGroundPartDegree() . . . 137
4.37.3.7 getRootDir() . . . 138
4.37.3.8 loadBGSprite() . . . 138
4.37.3.9 loadBGSprites() . . . 139
4.37.3.10 loadBGTextures() . . . 140
4.37.3.11 removeCars() . . . 141
4.37.3.12 setIcon() . . . 142
4.37.3.13 SFMLColorToImVec4() . . . 143
4.38 Utility.h . . . 144
4.39 test/CarTest.cc File Reference . . . 144
4.39.1 Detailed Description . . . 145
4.39.2 Function Documentation . . . 145
4.39.2.1 TEST() . . . 145
4.40 CarTest.cc . . . 146
4.41 test/EvolutionaryAlgorithmTest.cc File Reference . . . 147
4.41.1 Detailed Description . . . 147
4.41.2 Function Documentation . . . 148
4.41.2.1 TEST() [1/4] . . . 148
4.41.2.2 TEST() [2/4] . . . 148
4.41.2.3 TEST() [3/4] . . . 149
4.41.2.4 TEST() [4/4] . . . 150
4.42 EvolutionaryAlgorithmTest.cc . . . 150
4.43 test/ShapeTest.cc File Reference . . . 151
4.43.1 Detailed Description . . . 152
4.43.2 Function Documentation . . . 152
4.43.2.1 TEST() [1/16] . . . 152
4.43.2.2 TEST() [2/16] . . . 153
4.43.2.3 TEST() [3/16] . . . 153
4.43.2.4 TEST() [4/16] . . . 154

4.43.2.5 TEST() [5/16] 154
4.43.2.6 TEST() [6/16] 155
4.43.2.7 TEST() [7/16] 155
4.43.2.8 TEST() [8/16] 156
4.43.2.9 TEST() [9/16] 156
4.43.2.10 TEST() [10/16] 157
4.43.2.11 TEST() [11/16] 157
4.43.2.12 TEST() [12/16] 158
4.43.2.13 TEST() [13/16] 158
4.43.2.14 TEST() [14/16] 159
4.43.2.15 TEST() [15/16] 159
4.43.2.16 TEST() [16/16] 160
4.44 ShapeTest.cc 160
4.45 test/UtilityTest.cc File Reference 162
4.45.1 Function Documentation 163
4.45.1.1 TEST() [1/10] 163
4.45.1.2 TEST() [2/10] 164
4.45.1.3 TEST() [3/10] 164
4.45.1.4 TEST() [4/10] 165
4.45.1.5 TEST() [5/10] 166
4.45.1.6 TEST() [6/10] 166
4.45.1.7 TEST() [7/10] 167
4.45.1.8 TEST() [8/10] 167
4.45.1.9 TEST() [9/10] 168
4.45.1.10 TEST() [10/10] 168
4.46 UtilityTest.cc 169
Index **171**

Chapter 1

Class Index

1.1 Class List

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

Box
Struct representing a box 5
Car
Class representing a car 7
CarConfig 15
Chromosome 17
Circle
Struct representing a circle 19
Config 20
EvolutionaryAlgorithm 24
EvolutionaryAlgorithmConfig 31
MapGenConfig 37
Polygon
Struct representing a polygon 40

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

config/CarConfig.h
This file contains all the constant values for the car class 43
config/Config.h
This file contains all the constant values used in the program 44
config/EvolutionaryAlgorithmConfig.h
This file contains all the constant values used in the evolutionary algorithm 45
config/MapGenConfig.h
This file contains all the constant values used in the map generation algorithm 47
src/Car.cc
Creates a car with a polygon (car's body) and two circles (front and back wheels) 48
src/Car.h
Header file for the Car class 51
src/EvolutionaryAlgorithm.cc
Implementation file for EvolutionaryAlgorithm class, Algorithm used for evolving the cars 54
src/EvolutionaryAlgorithm.h
Header file for EvolutionaryAlgorithm class 57
src/GUI.cc
File containing GUI functions 59
src/GUI.h
Header for a file containing GUI functions 64
src/Main.cc
Main file for the project, contains the main loop 70
src/Render.cc
This file contains the render function, which is responsible for rendering all the shapes in the world 76
src/Render.h
Header file for render function 85
src/Shape.cc
This file contains functions for creating Box2D objects 93
src/Shape.h
Header file for functions for creating Box2D objects 100
src/UserInput.cc
File containing user input functions 109
src/UserInput.h
Header for a file containing user input functions 112

src/Utility.cc
File containing utility functions 116
src/Utility.h
Header for a file containing utility functions 130
test/CarTest.cc
This file contains tests for functions from src/Car.h 144
test/EvolutionaryAlgorithmTest.cc
This file contains tests for functions from src/EvolutionaryAlgorithm.h 147
test/ShapeTest.cc
This file contains tests for functions from src/Shape.h 151
test/UtilityTest.cc 162

Chapter 3

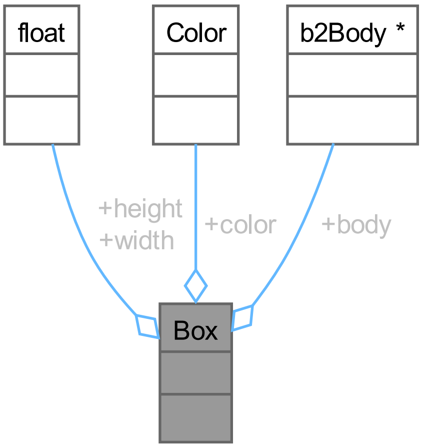
Class Documentation

3.1 Box Struct Reference

Struct representing a box.

```
#include <Shape.h>
```

Collaboration diagram for Box:



Public Attributes

- float width {}
- float height {}
- sf::Color color
- b2Body ∗ body {}

3.1.1 Detailed Description

Struct representing a box.

Definition at line 23 of file Shape.h.

3.1.2 Member Data Documentation

3.1.2.1 body

```
b2Body* Box::body {}
```

Definition at line 27 of file Shape.h.

3.1.2.2 color

```
sf::Color Box::color
```

Definition at line 26 of file Shape.h.

3.1.2.3 height

```
float Box::height {}
```

Definition at line 25 of file Shape.h.

3.1.2.4 width

```
float Box::width {}
```

Definition at line 24 of file Shape.h.

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

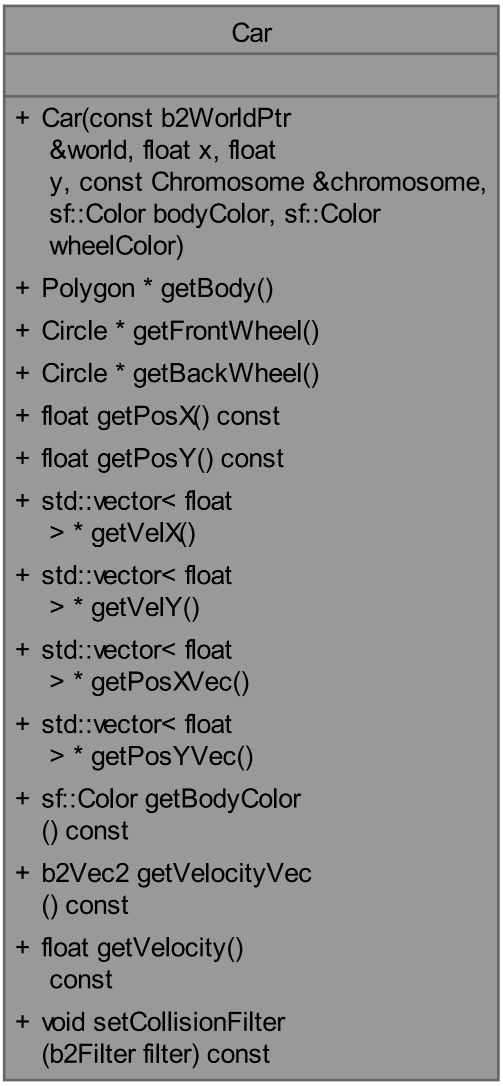
- src/Shape.h

3.2 Car Class Reference

Class representing a car.

```
#include <Car.h>
```

Collaboration diagram for Car:


Car
+ Car(const b2WorldPtr &world, float x, float y, const Chromosome &chromosome, sf::Color bodyColor, sf::Color wheelColor)
+ Polygon * getBody()
+ Circle * getFrontWheel()
+ Circle * getBackWheel()
+ float getPosX() const
+ float getPosY() const
+ std::vector< float > * getVelX()
+ std::vector< float > * getVelY()
+ std::vector< float > * getPosXVec()
+ std::vector< float > * getPosYVec()
+ sf::Color getBodyColor () const
+ b2Vec2 getVelocityVec () const
+ float getVelocity() const
+ void setCollisionFilter (b2Filter filter) const

Public Member Functions

- Car (const b2WorldPtr &world, float x, float y, const Chromosome &chromosome, sf::Color bodyColor, sf::Color wheelColor)

    Construct a new Car object.
- Polygon ∗ getBody ()

Get the car's body (polygon).
- Circle ∗ getFrontWheel ()

  Get the car's front wheel.
- Circle ∗ getBackWheel ()

  Get the car's front wheel.
- float getPosX () const

  Get the car's X position.
- float getPosY () const

  Get the car's X position.
- std::vector< float > ∗ getVelX ()

  Get the object holding car's data for X axis in the velocity graph.
- std::vector< float > ∗ getVelY ()

  Get the object holding car's data for Y axis in the velocity graph.
- std::vector< float > ∗ getPosXVec ()

  Get the object holding car's data for X axis in the position graph.
- std::vector< float > ∗ getPosYVec ()

  Get the object holding car's data for Y axis in the position graph.
- sf::Color getBodyColor () const

  Get the color of the car's body.
- b2Vec2 getVelocityVec () const

  Get the b2Vec2 value of car's velocity.
- float getVelocity () const

  Get the value of car's speed.
- void setCollisionFilter (b2Filter filter) const

  Set the collision filter so that cars pass through each-other.

3.2.1 Detailed Description

Class representing a car.

Definition at line 26 of file Car.h.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 Car()

```
Car::Car (
            const b2WorldPtr & world,
            float x,
            float y,
            const Chromosome & chromosome,
            sf::Color bodyColor,
            sf::Color wheelColor )
```

Construct a new Car object.

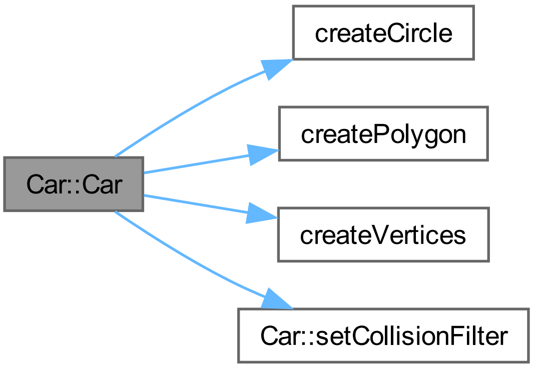
Parameters

world	A shared pointer to the Box2D world.
x	X coordinate of the car's center.
y	Y coordinate of the car's center.
chromosome	Genome of the car.
bodyColor	Color of the car's body.
wheelColor	Color of the car's wheels.

Definition at line 12 of file Car.cc.

```
                                                {
    // Create a polygon (octagon)

    auto vertices = createVertices(chromosome.bodyLengths);

    body_ =
        createPolygon(world, x, y, vertices, chromosome.bodyDensity, Config::FRICTION, bodyColor);

    // Create a circle
    frontWheel_ = createCircle(world, x, y, chromosome.wheelRadius.first,
                               chromosome.wheelDensity.first, Config::FRICTION, wheelColor);

    // Create another circle
    backWheel_ = createCircle(world, x, y, chromosome.wheelRadius.second,
                              chromosome.wheelDensity.second, Config::FRICTION, wheelColor);

    b2DistanceJointDef jointDef2;
    jointDef2.bodyA = body_.body;
    jointDef2.bodyB = frontWheel_.body;
    jointDef2.localAnchorA = vertices[Config::BACK_WHEEL_POS];
    jointDef2.localAnchorB = b2Vec2(0.0f, 0.0f);
    jointDef2.maxLength = CarConfig::MAX_JOINT_LENGTH;
    jointDef2.collideConnected = false;
    world->CreateJoint(&jointDef2);

    jointDef2.bodyA = body_.body;
    jointDef2.bodyB = backWheel_.body;
    jointDef2.localAnchorA = vertices[Config::FRONT_WHEEL_POS];
    jointDef2.localAnchorB = b2Vec2(0.0f, 0.0f);
    jointDef2.maxLength = CarConfig::MAX_JOINT_LENGTH;
    jointDef2.collideConnected = false;
    world->CreateJoint(&jointDef2);

    // Make cars pass through each-other
    // by setting collision filtering
    b2Filter filter;
    filter.categoryBits = Config::CATEGORY_BITS;
    filter.maskBits = Config::MASK_BITS;
    this->setCollisionFilter(filter);

    std::vector<float> v_axis(Config::VELOCITY_ARRAY_SIZE);
    std::vector<float> v_values(Config::VELOCITY_ARRAY_SIZE);

    std::iota(std::begin(v_axis), std::end(v_axis), 1);

    velX_ = v_axis;
    velY_ = v_values;

    posX_ = v_axis;
    posY_ = v_values;
}
```

Here is the call graph for this function:



3.2.3 Member Function Documentation

3.2.3.1 getBackWheel()

```
Circle * Car::getBackWheel ( )
```

Get the car's front wheel.

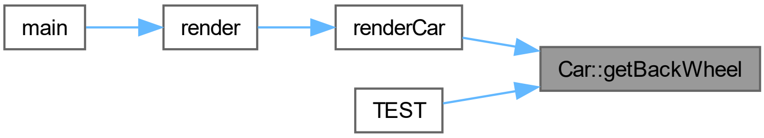
Returns

Circle∗ Pointer to the car's back wheel.

Definition at line 69 of file Car.cc.

```
00069 { return &backWheel_; }
```

Here is the caller graph for this function:



3.2.3.2 getBody()

```
Polygon * Car::getBody ( )
```

Get the car's body (polygon).

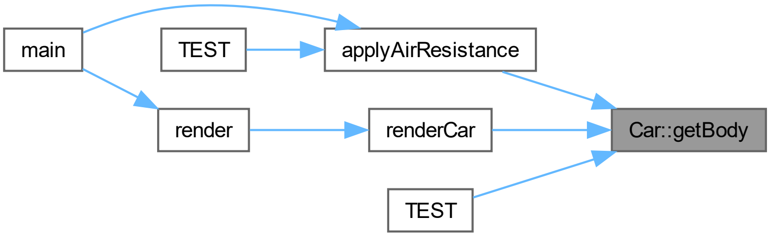
Returns

Polygon∗ Pointer to the car's body.

Definition at line 65 of file Car.cc.

```
00065 { return &body_; }
```

Here is the caller graph for this function:



3.2.3.3 getBodyColor()

```
sf::Color Car::getBodyColor ( ) const
```

Get the color of the car's body.

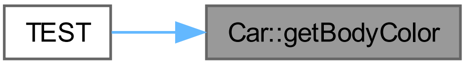
Returns

sf::Color Color of the car's body.

Definition at line 83 of file Car.cc.

```
00083 { return body_.color; }
```

Here is the caller graph for this function:



3.2.3.4 getFrontWheel()

```
Circle * Car::getFrontWheel ( )
```

Get the car's front wheel.

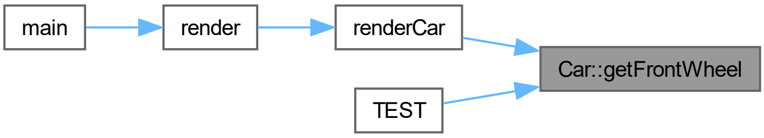
Returns

Circle∗ Pointer to the car's front wheel.

Definition at line 67 of file Car.cc.

```
00067 { return &frontWheel_; }
```

Here is the caller graph for this function:



3.2.3.5 getPosX()

```
float Car::getPosX ( ) const
```

Get the car's X position.

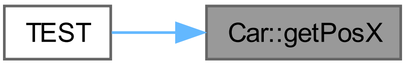
Returns

float X position.

Definition at line 71 of file Car.cc.

```
00071 { return body_.body->GetPosition().x; }
```

Here is the caller graph for this function:



3.2.3.6 getPosXVec()

```
std::vector< float > * Car::getPosXVec ( )
```

Get the object holding car's data for X axis in the position graph.

Returns

std::vector<float>∗ Pointer to the vector.

Definition at line 79 of file Car.cc.

```
00079 { return &posX_; }
```

3.2.3.7 getPosY()

```
float Car::getPosY ( ) const
```

Get the car's X position.

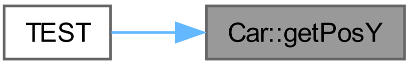
Returns

float Y position.

Definition at line 73 of file Car.cc.

```
00073 { return body_.body->GetPosition().y; }
```

Here is the caller graph for this function:



3.2.3.8 getPosYVec()

```
std::vector< float > * Car::getPosYVec ( )
```

Get the object holding car's data for Y axis in the position graph.

Returns

std::vector<float>* Pointer to the vector.

Definition at line 81 of file Car.cc.

```
00081 { return &posY_; }
```

3.2.3.9 getVelocity()

```
float Car::getVelocity ( ) const
```

Get the value of car's speed.

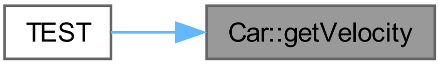
Returns

float Speed.

Definition at line 87 of file Car.cc.

```
00087 { return body_.body->GetLinearVelocity().Length(); }
```

Here is the caller graph for this function:



3.2.3.10 getVelocityVec()

```
b2Vec2 Car::getVelocityVec ( ) const
```

Get the b2Vec2 value of car's velocity.

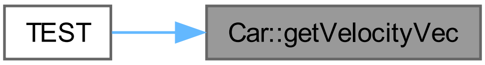
Returns

b2Vec2 Velocity vector.

Definition at line 85 of file Car.cc.

```
00085 { return body_.body->GetLinearVelocity(); }
```

Here is the caller graph for this function:



3.2.3.11 getVelX()

```
std::vector< float > * Car::getVelX ( )
```

Get the object holding car's data for X axis in the velocity graph.

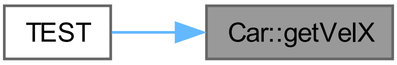
Returns

std::vector<float>∗ Pointer to the vector.

Definition at line 75 of file Car.cc.

```
00075 { return &velX_; }
```

Here is the caller graph for this function:



3.2.3.12 getVelY()

```
std::vector< float > * Car::getVelY ( )
```

Get the object holding car's data for Y axis in the velocity graph.

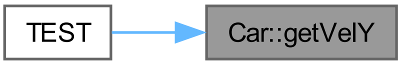
Returns

std::vector<float>∗ Pointer to the vector.

Definition at line 77 of file Car.cc.

```
00077 { return &velY_; }
```

Here is the caller graph for this function:



3.2.3.13 setCollisionFilter()

```
void Car::setCollisionFilter (
                b2Filter filter ) const
```

Set the collision filter so that cars pass through each-other.

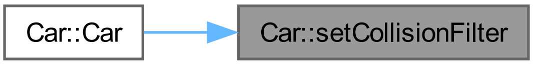
Parameters

filter	Collision filter.

Definition at line 89 of file Car.cc.

```
00089                                               {
00090     body_.body->GetFixtureList()->SetFilterData(filter);
00091     frontWheel_.body->GetFixtureList()->SetFilterData(filter);
00092     backWheel_.body->GetFixtureList()->SetFilterData(filter);
00093 }
```

Here is the caller graph for this function:



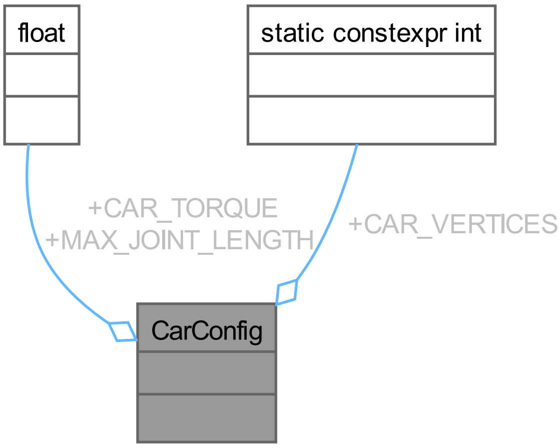
The documentation for this class was generated from the following files:

- src/Car.h
- src/Car.cc

3.3 CarConfig Class Reference

```
#include <CarConfig.h>
```

Collaboration diagram for CarConfig:



Static Public Attributes

- static constexpr float CAR_TORQUE = 2000.0f
- static constexpr float MAX_JOINT_LENGTH = 0.01f
- static constexpr int CAR_VERTICES = 8

3.3.1 Detailed Description

Definition at line 12 of file CarConfig.h.

3.3.2 Member Data Documentation

3.3.2.1 CAR_TORQUE

```
constexpr float CarConfig::CAR_TORQUE = 2000.0f  [static], [constexpr]
```

Definition at line 16 of file CarConfig.h.

3.3.2.2 CAR_VERTICES

```
constexpr int CarConfig::CAR_VERTICES = 8  [static], [constexpr]
```

Definition at line 21 of file CarConfig.h.

3.3.2.3 MAX_JOINT_LENGTH

```
constexpr float CarConfig::MAX_JOINT_LENGTH = 0.01f  [static], [constexpr]
```

Definition at line 19 of file CarConfig.h.

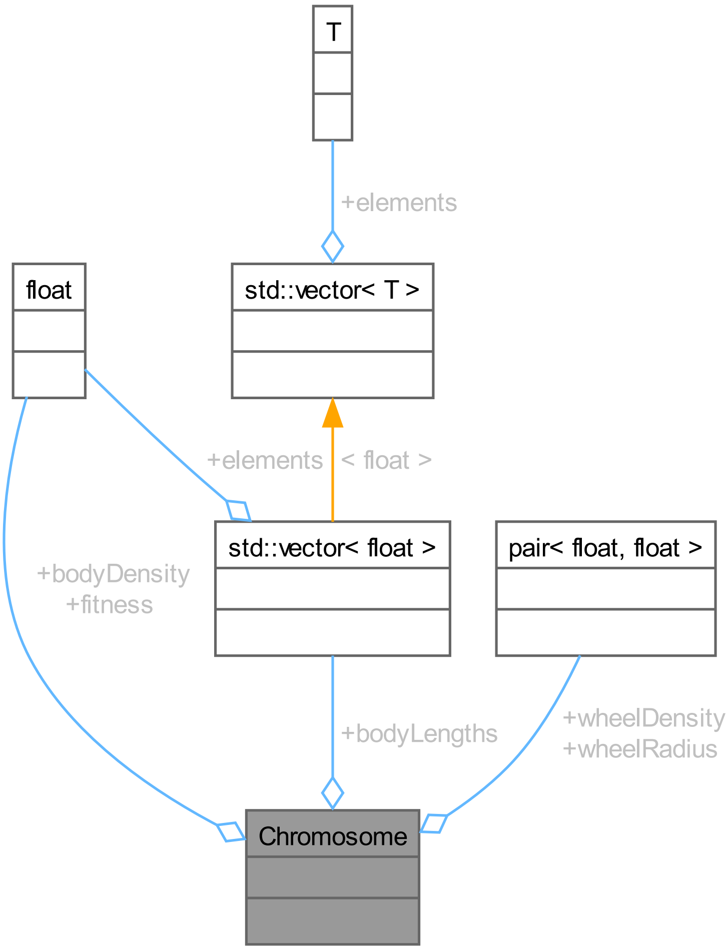
The documentation for this class was generated from the following file:

- config/CarConfig.h

3.4 Chromosome Struct Reference

```
#include <EvolutionaryAlgorithm.h>
```

Collaboration diagram for Chromosome:



Public Attributes

- std::vector< float > bodyLengths
- float bodyDensity
- std::pair< float, float > wheelRadius
- std::pair< float, float > wheelDensity
- float fitness

3.4.1 Detailed Description

Definition at line 21 of file EvolutionaryAlgorithm.h.

3.4.2 Member Data Documentation

3.4.2.1 bodyDensity

```
float Chromosome::bodyDensity
```

Definition at line 23 of file EvolutionaryAlgorithm.h.

3.4.2.2 bodyLengths

```
std::vector<float> Chromosome::bodyLengths
```

Definition at line 22 of file EvolutionaryAlgorithm.h.

3.4.2.3 fitness

```
float Chromosome::fitness
```

Definition at line 26 of file EvolutionaryAlgorithm.h.

3.4.2.4 wheelDensity

```
std::pair<float, float> Chromosome::wheelDensity
```

Definition at line 25 of file EvolutionaryAlgorithm.h.

3.4.2.5 wheelRadius

```
std::pair<float, float> Chromosome::wheelRadius
```

Definition at line 24 of file EvolutionaryAlgorithm.h.

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

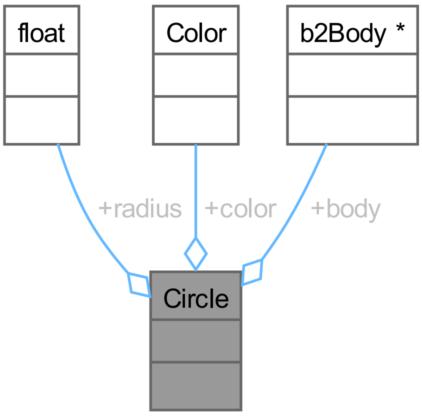
- src/EvolutionaryAlgorithm.h

3.5 Circle Struct Reference

Struct representing a circle.

```
#include <Shape.h>
```

Collaboration diagram for Circle:



Public Attributes

- float radius {}
- sf::Color color
- b2Body ∗ body {}

3.5.1 Detailed Description

Struct representing a circle.

Definition at line 33 of file Shape.h.

3.5.2 Member Data Documentation

3.5.2.1 body

```
b2Body* Circle::body {}
```

Definition at line 36 of file Shape.h.

3.5.2.2 color

```
sf::Color Circle::color
```

Definition at line 35 of file Shape.h.

3.5.2.3 radius

```
float Circle::radius {}
```

Definition at line 34 of file Shape.h.

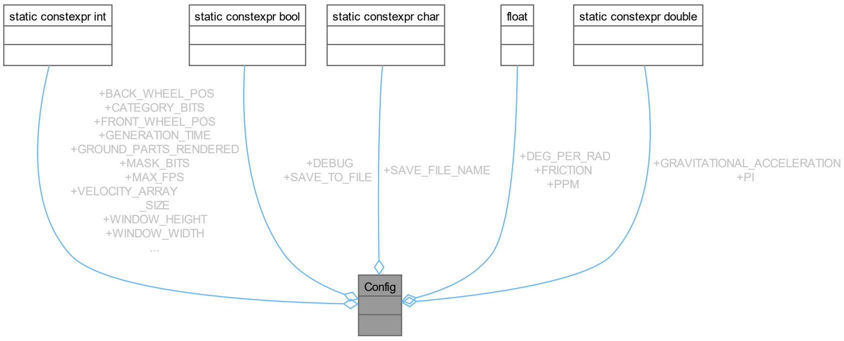
The documentation for this struct was generated from the following file:

- src/Shape.h

3.6 Config Class Reference

```
#include <Config.h>
```

Collaboration diagram for Config:



Static Public Attributes

- static constexpr int WINDOW_WIDTH = 1280
- static constexpr int WINDOW_HEIGHT = 720
- static constexpr int MAX_FPS = 60
- static constexpr int GROUND_PARTS_RENDERED = 32
- static constexpr bool SAVE_TO_FILE = true
- static constexpr char SAVE_FILE_NAME [ ] = ¨evoRacerOutput.json¨
- static constexpr int GENERATION_TIME
- static constexpr float PPM = 30.0F
- static constexpr float DEG_PER_RAD = 57.2957795F
- static constexpr bool DEBUG = true
- static constexpr double GRAVITATIONAL_ACCELERATION = -9.81f
- static constexpr float FRICTION = 0.3f
- static constexpr int VELOCITY_ARRAY_SIZE = 1000
- static constexpr double PI = 3.14159265358979323846
- static constexpr int BACK_WHEEL_POS = 1
- static constexpr int FRONT_WHEEL_POS = 3
- static constexpr int CATEGORY_BITS = 2
- static constexpr int MASK_BITS = 1

3.6.1 Detailed Description

Definition at line 12 of file Config.h.

3.6.2 Member Data Documentation

3.6.2.1 BACK_WHEEL_POS

```
constexpr int Config::BACK_WHEEL_POS = 1  [static], [constexpr]
```

Definition at line 43 of file Config.h.

3.6.2.2 CATEGORY_BITS

```
constexpr int Config::CATEGORY_BITS = 2  [static], [constexpr]
```

Definition at line 45 of file Config.h.

3.6.2.3 DEBUG

```
constexpr bool Config::DEBUG = true  [static], [constexpr]
```

Definition at line 35 of file Config.h.

3.6.2.4 DEG_PER_RAD

```
constexpr float Config::DEG_PER_RAD = 57.2957795F  [static], [constexpr]
```

Definition at line 32 of file Config.h.

3.6.2.5 FRICTION

```
constexpr float Config::FRICTION = 0.3f  [static], [constexpr]
```

Definition at line 39 of file Config.h.

3.6.2.6 FRONT_WHEEL_POS

```
constexpr int Config::FRONT_WHEEL_POS = 3  [static], [constexpr]
```

Definition at line 44 of file Config.h.

3.6.2.7 GENERATION_TIME

```
constexpr int Config::GENERATION_TIME  [static], [constexpr]
```

Initial value:

```
=
       3000
```

Definition at line 26 of file Config.h.

3.6.2.8 GRAVITATIONAL_ACCELERATION

```
constexpr double Config::GRAVITATIONAL_ACCELERATION = -9.81f  [static], [constexpr]
```

Definition at line 38 of file Config.h.

3.6.2.9 GROUND_PARTS_RENDERED

```
constexpr int Config::GROUND_PARTS_RENDERED = 32  [static], [constexpr]
```

Definition at line 20 of file Config.h.

3.6.2.10 MASK_BITS

```
constexpr int Config::MASK_BITS = 1  [static], [constexpr]
```

Definition at line 46 of file Config.h.

3.6.2.11 MAX_FPS

```
constexpr int Config::MAX_FPS = 60  [static], [constexpr]
```

Definition at line 18 of file Config.h.

3.6.2.12 PI

```
constexpr double Config::PI = 3.14159265358979323846  [static], [constexpr]
```

Definition at line 42 of file Config.h.

3.6.2.13 PPM

```
constexpr float Config::PPM = 30.0F  [static], [constexpr]
```

Definition at line 29 of file Config.h.

3.6.2.14 SAVE_FILE_NAME

```
constexpr char Config::SAVE_FILE_NAME[ ] = ¨evoRacerOutput.json¨  [static], [constexpr]
```

Definition at line 24 of file Config.h.

3.6.2.15 SAVE_TO_FILE

```
constexpr bool Config::SAVE_TO_FILE = true  [static], [constexpr]
```

Definition at line 23 of file Config.h.

3.6.2.16 VELOCITY_ARRAY_SIZE

```
constexpr int Config::VELOCITY_ARRAY_SIZE = 1000  [static], [constexpr]
```

Definition at line 40 of file Config.h.

3.6.2.17 WINDOW_HEIGHT

```
constexpr int Config::WINDOW_HEIGHT = 720  [static], [constexpr]
```

Definition at line 15 of file Config.h.

3.6.2.18 WINDOW_WIDTH

```
constexpr int Config::WINDOW_WIDTH = 1280  [static], [constexpr]
```

Definition at line 14 of file Config.h.

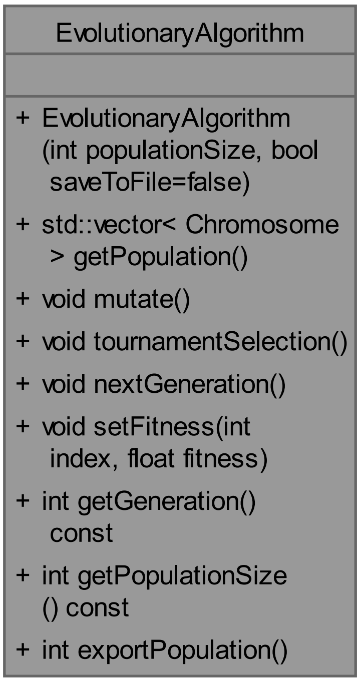
The documentation for this class was generated from the following file:

- config/Config.h

3.7 EvolutionaryAlgorithm Class Reference

```
#include <EvolutionaryAlgorithm.h>
```

Collaboration diagram for EvolutionaryAlgorithm:


EvolutionaryAlgorithm
+ EvolutionaryAlgorithm (int populationSize, bool saveToFile=false)
+ std::vector< Chromosome > getPopulation()
+ void mutate()
+ void tournamentSelection()
+ void nextGeneration()
+ void setFitness(int index, float fitness)
+ int getGeneration() const
+ int getPopulationSize () const
+ int exportPopulation()

Public Member Functions

- EvolutionaryAlgorithm (int populationSize, bool saveToFile=false)

  Construct a new Evolutionary Algorithm:: Evolutionary Algorithm object.
- std::vector< Chromosome > getPopulation ()
- void mutate ()

  mutates the population
- void tournamentSelection ()

  performs tournament selection on the population
- void nextGeneration ()

  steps the algorithm to the next generation
- void setFitness (int index, float fitness)
- int getGeneration () const
- int getPopulationSize () const
- int exportPopulation ()

  exports the population to a json file

3.7.1 Detailed Description

Definition at line 29 of file EvolutionaryAlgorithm.h.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 EvolutionaryAlgorithm()

```
EvolutionaryAlgorithm::EvolutionaryAlgorithm (
              int populationSize,
              bool saveToFile = false )  [explicit]
```

Construct a new Evolutionary Algorithm:: Evolutionary Algorithm object.

Parameters

populationSize	The size of the population
saveToFile	Whether to save the population to a file

Definition at line 12 of file EvolutionaryAlgorithm.cc.

```
00012                                                                         {
00013     populationSize_ = populationSize;
00014     generation_ = 0;
00015     initializePopulation();
00016     saveToFile_ = saveToFile;
00017 }
```

3.7.3 Member Function Documentation

3.7.3.1 exportPopulation()

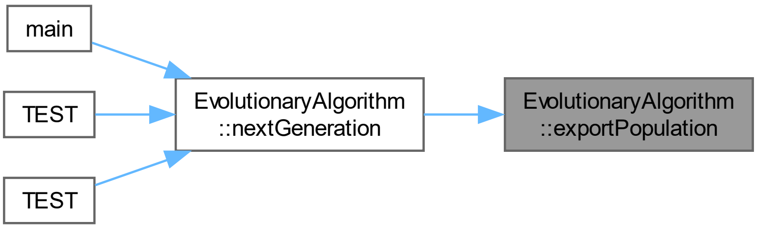
```
int EvolutionaryAlgorithm::exportPopulation ( )
```

exports the population to a json file

Definition at line 189 of file EvolutionaryAlgorithm.cc.

```
00189                                            {
00190     nlohmann::json jsonData;
00191     jsonData["generation"] = generation_;
00192 
00193     std::deque<nlohmann::json> populationData;
00194 
00195     for (const auto& chromosome : population_) {
00196         nlohmann::json chromosomeJson;
00197         chromosomeJson["bodyLengths"] = chromosome.bodyLengths;
00198         chromosomeJson["bodyDensity"] = chromosome.bodyDensity;
00199         chromosomeJson["wheelRadius"] = {chromosome.wheelRadius.first,
00200                                          chromosome.wheelRadius.second};
00201         chromosomeJson["wheelDensity"] = {chromosome.wheelDensity.first,
00202                                           chromosome.wheelDensity.second};
00203         chromosomeJson["fitness"] = chromosome.fitness;
00204         populationData.push_front(chromosomeJson);
00205     }
00206 
00207     jsonData["population"] = populationData;
00208 
00209     std::string jsonString = jsonData.dump(4);
00210 
00211     std::ofstream outputFile(Config::SAVE_FILE_NAME, std::ios::app);
00212     if (!outputFile.is_open()) {
00213         return 1;
00214     }
00215     outputFile « jsonString;
00216     outputFile.close();
00217
00218     return 0;
00219 }
```

Here is the caller graph for this function:


main
TEST
TEST
EvolutionaryAlgorithm ::nextGeneration
EvolutionaryAlgorithm ::exportPopulation

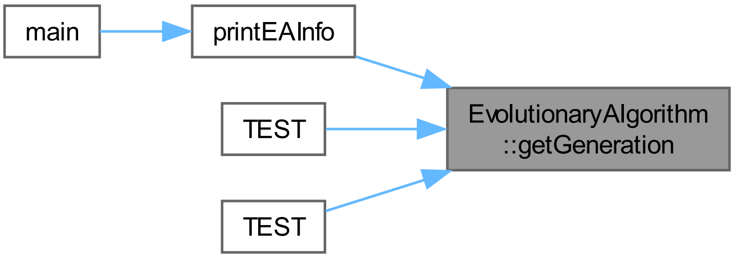
3.7.3.2 getGeneration()

```
int EvolutionaryAlgorithm::getGeneration ( ) const  [inline]
```

Definition at line 67 of file EvolutionaryAlgorithm.h.

```
00067 { return generation_; }
```

Here is the caller graph for this function:



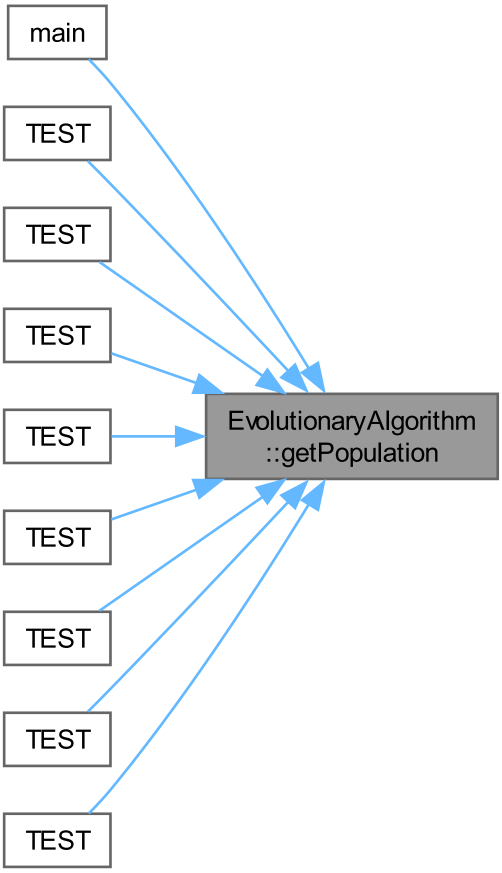
3.7.3.3 getPopulation()

```
std::vector< Chromosome > EvolutionaryAlgorithm::getPopulation ( )  [inline]
```

Definition at line 53 of file EvolutionaryAlgorithm.h.

```
00053 { return population_; }
```

Here is the caller graph for this function:



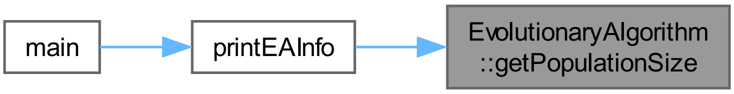
3.7.3.4 getPopulationSize()

```
int EvolutionaryAlgorithm::getPopulationSize ( ) const  [inline]
```

Definition at line 68 of file EvolutionaryAlgorithm.h.

```
00068 { return populationSize_; }
```

Here is the caller graph for this function:



3.7.3.5 mutate()

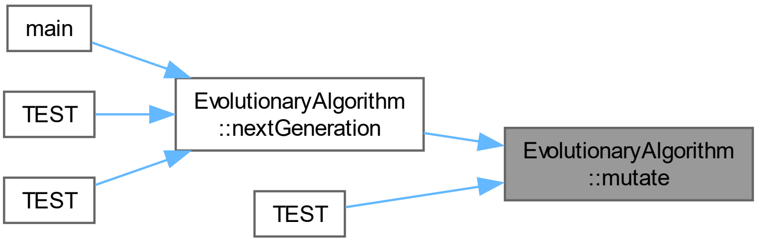
```
void EvolutionaryAlgorithm::mutate ( )
```

mutates the population

Definition at line 84 of file EvolutionaryAlgorithm.cc.

```
00084                                    {
00085     std::random_device rd;
00086     std::mt19937 gen(rd());
00087     std::normal_distribution<float> dist(0.0, 1.0);
00088 
00089     for (auto& chrom : population_) {
00090         // Mutate bodyLengths
00091         for (auto& length : chrom.bodyLengths) {
00092             if (dist(gen) < EvolutionaryAlgorithmConfig::MUTATION_RATE_BODY_LENGTHS) {
00093                 length += dist(gen) * EvolutionaryAlgorithmConfig::MUTATION_FACTOR_BODY_LENGTHS;
00094                 length = std::max(length, EvolutionaryAlgorithmConfig::MIN_BODY_LENGTH);
00095                 length = std::min(length, EvolutionaryAlgorithmConfig::MAX_BODY_LENGTH);
00096             }
00097         }
00098 
00099         // Mutate bodyDensity
00100         if (dist(gen) < EvolutionaryAlgorithmConfig::MUTATION_RATE_BODY_DENSITY) {
00101             chrom.bodyDensity +=
00102                 dist(gen) * EvolutionaryAlgorithmConfig::MUTATION_FACTOR_BODY_DENSITY;
00103 
00104             chrom.bodyDensity =
00105                 std::max(chrom.bodyDensity, EvolutionaryAlgorithmConfig::MIN_BODY_DENSITY);
00106             chrom.bodyDensity =
00107                 std::min(chrom.bodyDensity, EvolutionaryAlgorithmConfig::MAX_BODY_DENSITY);
00108         }
00109 
00110         // Mutate wheelRadius
00111         if (dist(gen) < EvolutionaryAlgorithmConfig::MUTATION_RATE_WHEEL_RADIUS) {
00112             chrom.wheelRadius.first +=
00113                 dist(gen) * EvolutionaryAlgorithmConfig::MUTATION_FACTOR_WHEEL_RADIUS;
00114             chrom.wheelRadius.first =
00115                 std::max(chrom.wheelRadius.first, EvolutionaryAlgorithmConfig::MIN_WHEEL_RADIUS);
00116             chrom.wheelRadius.first =
00117                 std::min(chrom.wheelRadius.first, EvolutionaryAlgorithmConfig::MAX_WHEEL_RADIUS);
00118         }
00119         if (dist(gen) < EvolutionaryAlgorithmConfig::MUTATION_RATE_WHEEL_RADIUS) {
00120             chrom.wheelRadius.second +=
00121                 dist(gen) * EvolutionaryAlgorithmConfig::MUTATION_FACTOR_WHEEL_RADIUS;
00122             chrom.wheelRadius.second =
00123                 std::max(chrom.wheelRadius.second, EvolutionaryAlgorithmConfig::MIN_WHEEL_RADIUS);
00124             chrom.wheelRadius.second =
00125                 std::min(chrom.wheelRadius.second, EvolutionaryAlgorithmConfig::MAX_WHEEL_RADIUS);
00126         }
00127 
00128         // Mutate wheelDensity
00129         if (dist(gen) < EvolutionaryAlgorithmConfig::MUTATION_RATE_WHEEL_DENSITY) {
00130             chrom.wheelDensity.first +=
00131                 dist(gen) * EvolutionaryAlgorithmConfig::MUTATION_FACTOR_WHEEL_DENSITY;
00132             chrom.wheelDensity.first =
00133                 std::max(chrom.wheelDensity.first, EvolutionaryAlgorithmConfig::MIN_WHEEL_DENSITY);
00134             chrom.wheelDensity.first =
00135                 std::min(chrom.wheelDensity.first, EvolutionaryAlgorithmConfig::MAX_WHEEL_DENSITY);
00136         }
00137         if (dist(gen) < EvolutionaryAlgorithmConfig::MUTATION_RATE_WHEEL_DENSITY) {
00138             chrom.wheelDensity.second +=
00139                 dist(gen) * EvolutionaryAlgorithmConfig::MUTATION_FACTOR_WHEEL_DENSITY;
00140             chrom.wheelDensity.second =
00141                 std::max(chrom.wheelDensity.second, EvolutionaryAlgorithmConfig::MIN_WHEEL_DENSITY);
00142             chrom.wheelDensity.second =
00143                 std::min(chrom.wheelDensity.second, EvolutionaryAlgorithmConfig::MAX_WHEEL_DENSITY);
00144         }
00145     }
00146 }
```

Here is the caller graph for this function:


main
TEST
TEST
EvolutionaryAlgorithm ::nextGeneration
TEST
EvolutionaryAlgorithm ::mutate

3.7.3.6 nextGeneration()

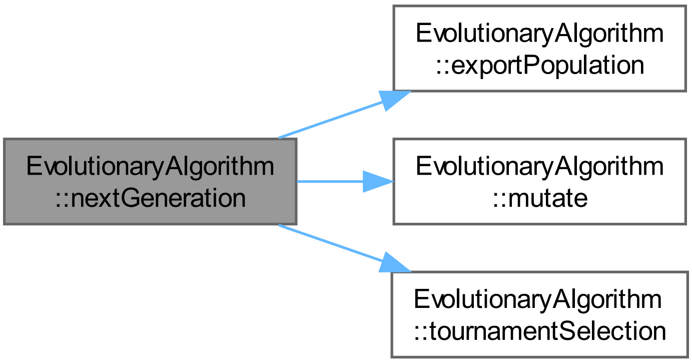
```
void EvolutionaryAlgorithm::nextGeneration ( )
```

steps the algorithm to the next generation

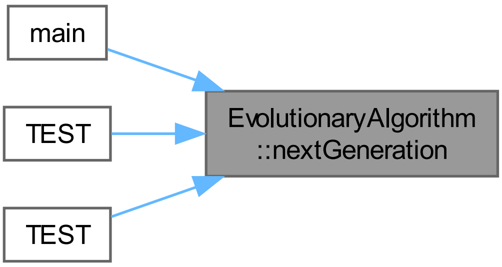
Definition at line 180 of file EvolutionaryAlgorithm.cc.

```
00180                                             {
00181     if (saveToFile_) {
00182         exportPopulation();
00183     }
00184     tournamentSelection();
00185     mutate();
00186     ++generation_;
00187 }
```

Here is the call graph for this function:


EvolutionaryAlgorithm ::exportPopulation
EvolutionaryAlgorithm ::nextGeneration
EvolutionaryAlgorithm ::mutate
EvolutionaryAlgorithm ::tournamentSelection

Here is the caller graph for this function:



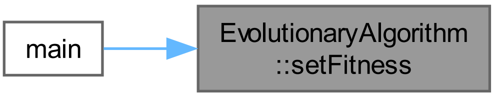
3.7.3.7 setFitness()

```
void EvolutionaryAlgorithm::setFitness (
             int index,
             float fitness )  [inline]
```

Definition at line 66 of file EvolutionaryAlgorithm.h.

```
00066 { population_[index].fitness = fitness; }
```

Here is the caller graph for this function:



3.7.3.8 tournamentSelection()

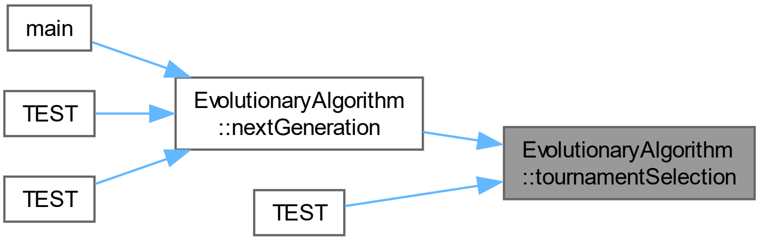
```
void EvolutionaryAlgorithm::tournamentSelection ( )
```

performs tournament selection on the population

Definition at line 169 of file EvolutionaryAlgorithm.cc.

```
00169                                              {
00170     std::vector<Chromosome> tournament_winners;
00171
00172     tournament_winners.reserve(populationSize_);
00173     for (int i = 0; i < populationSize_; ++i) {
00174         tournament_winners.push_back(tournament());
00175     }
00176
00177     population_ = tournament_winners;
00178 }
```

Here is the caller graph for this function:



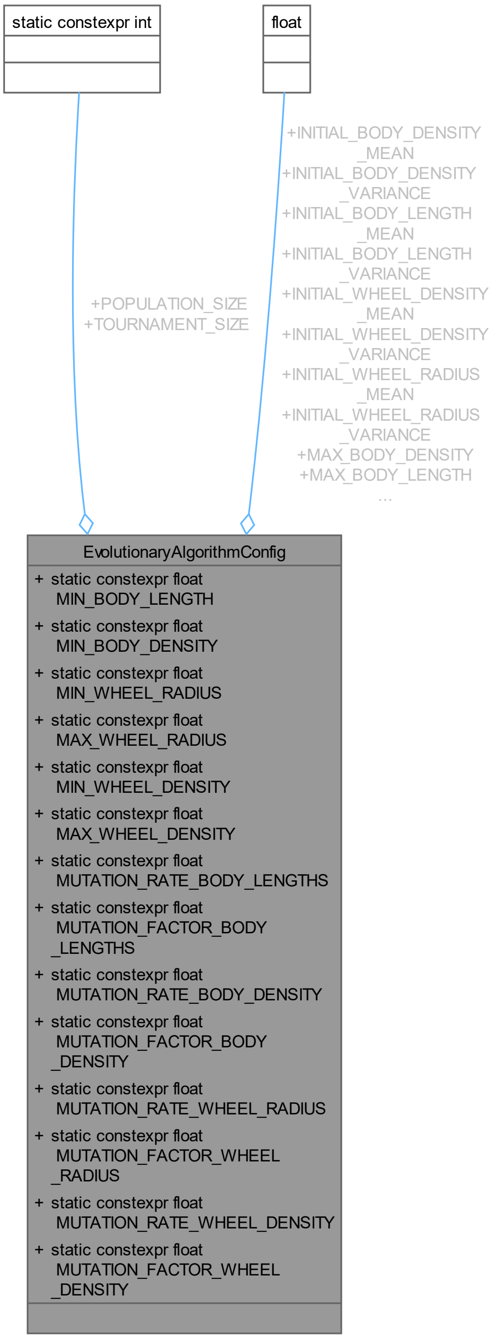
The documentation for this class was generated from the following files:

- src/EvolutionaryAlgorithm.h
- src/EvolutionaryAlgorithm.cc

3.8 EvolutionaryAlgorithmConfig Class Reference

```
#include <EvolutionaryAlgorithmConfig.h>
```

Collaboration diagram for EvolutionaryAlgorithmConfig:


static constexpr int
float
+INITIAL_BODY_DENSITY
_MEAN
+INITIAL_BODY_DENSITY
_VARIANCE
+INITIAL_BODY_LENGTH
_MEAN
+INITIAL_BODY_LENGTH
_VARIANCE
+INITIAL_WHEEL_DENSITY
_MEAN
+INITIAL_WHEEL_DENSITY
_VARIANCE
+INITIAL_WHEEL_RADIUS
_MEAN
+INITIAL_WHEEL_RADIUS
_VARIANCE
+MAX_BODY_DENSITY
+MAX_BODY_LENGTH
...
+POPULATION_SIZE
+TOURNAMENT_SIZE
EvolutionaryAlgorithmConfig
+ static constexpr float MIN_BODY_LENGTH
+ static constexpr float MIN_BODY_DENSITY
+ static constexpr float MIN_WHEEL_RADIUS
+ static constexpr float MAX_WHEEL_RADIUS
+ static constexpr float MIN_WHEEL_DENSITY
+ static constexpr float MAX_WHEEL_DENSITY
+ static constexpr float MUTATION_RATE_BODY_LENGTHS
+ static constexpr float MUTATION_FACTOR_BODY_LENGTHS
+ static constexpr float MUTATION_RATE_BODY_DENSITY
+ static constexpr float MUTATION_FACTOR_BODY_DENSITY
+ static constexpr float MUTATION_RATE_WHEEL_RADIUS
+ static constexpr float MUTATION_FACTOR_WHEEL_RADIUS
+ static constexpr float MUTATION_RATE_WHEEL_DENSITY
+ static constexpr float MUTATION_FACTOR_WHEEL_DENSITY

Static Public Attributes

- static constexpr int POPULATION_SIZE = 15
- static constexpr float MIN_BODY_LENGTH = 1.0f
- static constexpr float MAX_BODY_LENGTH = 5.0f
- static constexpr float MIN_BODY_DENSITY = 10.0f
- static constexpr float MAX_BODY_DENSITY = 1000.0f

- static constexpr float MIN_WHEEL_RADIUS = 2.0f
- static constexpr float MAX_WHEEL_RADIUS = 40.0f
- static constexpr float MIN_WHEEL_DENSITY = 10.0f
- static constexpr float MAX_WHEEL_DENSITY = 1000.0f
- static constexpr float INITIAL_BODY_LENGTH_MEAN = 3.0f
- static constexpr float INITIAL_BODY_LENGTH_VARIANCE = 1.0f
- static constexpr float INITIAL_BODY_DENSITY_MEAN = 100.0f
- static constexpr float INITIAL_BODY_DENSITY_VARIANCE = 100.0f
- static constexpr float INITIAL_WHEEL_RADIUS_MEAN = 25.0f
- static constexpr float INITIAL_WHEEL_RADIUS_VARIANCE = 10.0f
- static constexpr float INITIAL_WHEEL_DENSITY_MEAN = 100.0f
- static constexpr float INITIAL_WHEEL_DENSITY_VARIANCE = 100.0f
- static constexpr float MUTATION_RATE_BODY_LENGTHS = 0.1f
- static constexpr float MUTATION_FACTOR_BODY_LENGTHS = 0.5f
- static constexpr float MUTATION_RATE_BODY_DENSITY = 0.2f
- static constexpr float MUTATION_FACTOR_BODY_DENSITY = 20.0f
- static constexpr float MUTATION_RATE_WHEEL_RADIUS = 0.3f
- static constexpr float MUTATION_FACTOR_WHEEL_RADIUS = 2.0f
- static constexpr float MUTATION_RATE_WHEEL_DENSITY = 0.1f
- static constexpr float MUTATION_FACTOR_WHEEL_DENSITY = 20.0f
- static constexpr int TOURNAMENT_SIZE = 3

3.8.1 Detailed Description

Definition at line 12 of file EvolutionaryAlgorithmConfig.h.

3.8.2 Member Data Documentation

3.8.2.1 INITIAL_BODY_DENSITY_MEAN

```
constexpr float EvolutionaryAlgorithmConfig::INITIAL_BODY_DENSITY_MEAN = 100.0f  [static],
[constexpr]
```

Definition at line 34 of file EvolutionaryAlgorithmConfig.h.

3.8.2.2 INITIAL_BODY_DENSITY_VARIANCE

```
constexpr float EvolutionaryAlgorithmConfig::INITIAL_BODY_DENSITY_VARIANCE = 100.0f  [static],
[constexpr]
```

Definition at line 35 of file EvolutionaryAlgorithmConfig.h.

3.8.2.3 INITIAL_BODY_LENGTH_MEAN

```
constexpr float EvolutionaryAlgorithmConfig::INITIAL_BODY_LENGTH_MEAN = 3.0f  [static], [constexpr]
```

Definition at line 31 of file EvolutionaryAlgorithmConfig.h.

3.8.2.4 INITIAL_BODY_LENGTH_VARIANCE

```
constexpr float EvolutionaryAlgorithmConfig::INITIAL_BODY_LENGTH_VARIANCE = 1.0f  [static],
[constexpr]
```

Definition at line 32 of file EvolutionaryAlgorithmConfig.h.

3.8.2.5 INITIAL_WHEEL_DENSITY_MEAN

```
constexpr float EvolutionaryAlgorithmConfig::INITIAL_WHEEL_DENSITY_MEAN = 100.0f  [static],
[constexpr]
```

Definition at line 40 of file EvolutionaryAlgorithmConfig.h.

3.8.2.6 INITIAL_WHEEL_DENSITY_VARIANCE

```
constexpr float EvolutionaryAlgorithmConfig::INITIAL_WHEEL_DENSITY_VARIANCE = 100.0f  [static],
[constexpr]
```

Definition at line 41 of file EvolutionaryAlgorithmConfig.h.

3.8.2.7 INITIAL_WHEEL_RADIUS_MEAN

```
constexpr float EvolutionaryAlgorithmConfig::INITIAL_WHEEL_RADIUS_MEAN = 25.0f  [static],
[constexpr]
```

Definition at line 37 of file EvolutionaryAlgorithmConfig.h.

3.8.2.8 INITIAL_WHEEL_RADIUS_VARIANCE

```
constexpr float EvolutionaryAlgorithmConfig::INITIAL_WHEEL_RADIUS_VARIANCE = 10.0f  [static],
[constexpr]
```

Definition at line 38 of file EvolutionaryAlgorithmConfig.h.

3.8.2.9 MAX_BODY_DENSITY

```
constexpr float EvolutionaryAlgorithmConfig::MAX_BODY_DENSITY = 1000.0f  [static], [constexpr]
```

Definition at line 22 of file EvolutionaryAlgorithmConfig.h.

3.8.2.10 MAX_BODY_LENGTH

```
constexpr float EvolutionaryAlgorithmConfig::MAX_BODY_LENGTH = 5.0f  [static], [constexpr]
```

Definition at line 19 of file EvolutionaryAlgorithmConfig.h.

3.8.2.11 MAX_WHEEL_DENSITY

```
constexpr float EvolutionaryAlgorithmConfig::MAX_WHEEL_DENSITY = 1000.0f  [static], [constexpr]
```

Definition at line 28 of file EvolutionaryAlgorithmConfig.h.

3.8.2.12 MAX_WHEEL_RADIUS

```
constexpr float EvolutionaryAlgorithmConfig::MAX_WHEEL_RADIUS = 40.0f  [static], [constexpr]
```

Definition at line 25 of file EvolutionaryAlgorithmConfig.h.

3.8.2.13 MIN_BODY_DENSITY

```
constexpr float EvolutionaryAlgorithmConfig::MIN_BODY_DENSITY = 10.0f  [static], [constexpr]
```

Definition at line 21 of file EvolutionaryAlgorithmConfig.h.

3.8.2.14 MIN_BODY_LENGTH

```
constexpr float EvolutionaryAlgorithmConfig::MIN_BODY_LENGTH = 1.0f  [static], [constexpr]
```

Definition at line 18 of file EvolutionaryAlgorithmConfig.h.

3.8.2.15 MIN_WHEEL_DENSITY

```
constexpr float EvolutionaryAlgorithmConfig::MIN_WHEEL_DENSITY = 10.0f  [static], [constexpr]
```

Definition at line 27 of file EvolutionaryAlgorithmConfig.h.

3.8.2.16 MIN_WHEEL_RADIUS

```
constexpr float EvolutionaryAlgorithmConfig::MIN_WHEEL_RADIUS = 2.0f  [static], [constexpr]
```

Definition at line 24 of file EvolutionaryAlgorithmConfig.h.

3.8.2.17 MUTATION_FACTOR_BODY_DENSITY

```
constexpr float EvolutionaryAlgorithmConfig::MUTATION_FACTOR_BODY_DENSITY = 20.0f  [static],
[constexpr]
```

Definition at line 49 of file EvolutionaryAlgorithmConfig.h.

3.8.2.18 MUTATION_FACTOR_BODY_LENGTHS

```
constexpr float EvolutionaryAlgorithmConfig::MUTATION_FACTOR_BODY_LENGTHS = 0.5f  [static],
[constexpr]
```

Definition at line 46 of file EvolutionaryAlgorithmConfig.h.

3.8.2.19 MUTATION_FACTOR_WHEEL_DENSITY

```
constexpr float EvolutionaryAlgorithmConfig::MUTATION_FACTOR_WHEEL_DENSITY = 20.0f  [static],
[constexpr]
```

Definition at line 55 of file EvolutionaryAlgorithmConfig.h.

3.8.2.20 MUTATION_FACTOR_WHEEL_RADIUS

```
constexpr float EvolutionaryAlgorithmConfig::MUTATION_FACTOR_WHEEL_RADIUS = 2.0f  [static],
[constexpr]
```

Definition at line 52 of file EvolutionaryAlgorithmConfig.h.

3.8.2.21 MUTATION_RATE_BODY_DENSITY

```
constexpr float EvolutionaryAlgorithmConfig::MUTATION_RATE_BODY_DENSITY = 0.2f  [static],
[constexpr]
```

Definition at line 48 of file EvolutionaryAlgorithmConfig.h.

3.8.2.22 MUTATION_RATE_BODY_LENGTHS

```
constexpr float EvolutionaryAlgorithmConfig::MUTATION_RATE_BODY_LENGTHS = 0.1f  [static],
[constexpr]
```

Definition at line 45 of file EvolutionaryAlgorithmConfig.h.

3.8.2.23 MUTATION_RATE_WHEEL_DENSITY

```
constexpr float EvolutionaryAlgorithmConfig::MUTATION_RATE_WHEEL_DENSITY = 0.1f  [static],
[constexpr]
```

Definition at line 54 of file EvolutionaryAlgorithmConfig.h.

3.8.2.24 MUTATION_RATE_WHEEL_RADIUS

```
constexpr float EvolutionaryAlgorithmConfig::MUTATION_RATE_WHEEL_RADIUS = 0.3f  [static],
[constexpr]
```

Definition at line 51 of file EvolutionaryAlgorithmConfig.h.

3.8.2.25 POPULATION_SIZE

```
constexpr int EvolutionaryAlgorithmConfig::POPULATION_SIZE = 15  [static], [constexpr]
```

Definition at line 15 of file EvolutionaryAlgorithmConfig.h.

3.8.2.26 TOURNAMENT_SIZE

```
constexpr int EvolutionaryAlgorithmConfig::TOURNAMENT_SIZE = 3  [static], [constexpr]
```

Definition at line 58 of file EvolutionaryAlgorithmConfig.h.

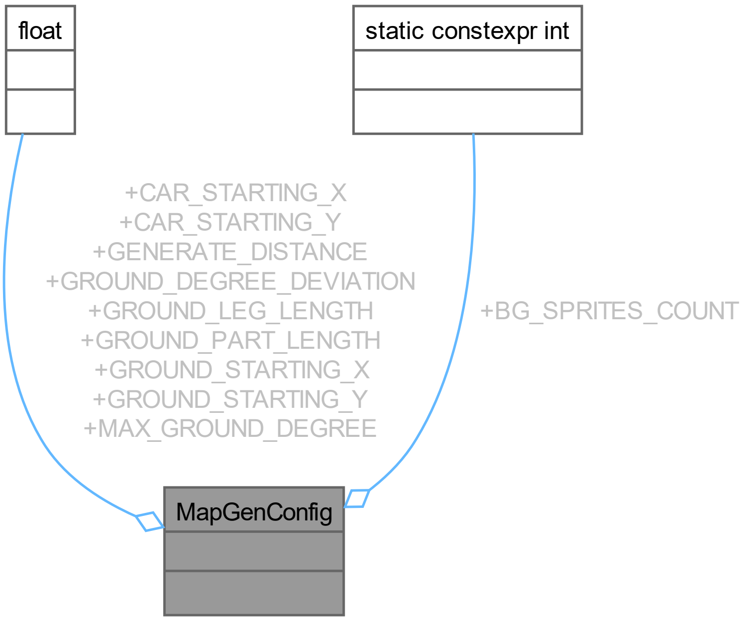
The documentation for this class was generated from the following file:

- config/EvolutionaryAlgorithmConfig.h

3.9 MapGenConfig Class Reference

```
#include <MapGenConfig.h>
```

Collaboration diagram for MapGenConfig:



Static Public Attributes

- static constexpr float GENERATE_DISTANCE = 666.0
- static constexpr float GROUND_STARTING_X = 0.0
- static constexpr float GROUND_STARTING_Y = 360.0
- static constexpr float GROUND_LEG_LENGTH = 4.0
- static constexpr float GROUND_PART_LENGTH = 1.5
- static constexpr int BG_SPRITES_COUNT = 5
- static constexpr float CAR_STARTING_X = 250.0
- static constexpr float CAR_STARTING_Y = 650.0
- static constexpr float GROUND_DEGREE_DEVIATION = 12.0f
- static constexpr float MAX_GROUND_DEGREE = 50.0f

3.9.1 Detailed Description

Definition at line 12 of file MapGenConfig.h.

3.9.2 Member Data Documentation

3.9.2.1 BG_SPRITES_COUNT

```
constexpr int MapGenConfig::BG_SPRITES_COUNT = 5  [static], [constexpr]
```

Definition at line 19 of file MapGenConfig.h.

3.9.2.2 CAR_STARTING_X

```
constexpr float MapGenConfig::CAR_STARTING_X = 250.0  [static], [constexpr]
```

Definition at line 21 of file MapGenConfig.h.

3.9.2.3 CAR_STARTING_Y

```
constexpr float MapGenConfig::CAR_STARTING_Y = 650.0  [static], [constexpr]
```

Definition at line 22 of file MapGenConfig.h.

3.9.2.4 GENERATE_DISTANCE

```
constexpr float MapGenConfig::GENERATE_DISTANCE = 666.0  [static], [constexpr]
```

Definition at line 14 of file MapGenConfig.h.

3.9.2.5 GROUND_DEGREE_DEVIATION

```
constexpr float MapGenConfig::GROUND_DEGREE_DEVIATION = 12.0f  [static], [constexpr]
```

Definition at line 25 of file MapGenConfig.h.

3.9.2.6 GROUND_LEG_LENGTH

```
constexpr float MapGenConfig::GROUND_LEG_LENGTH = 4.0  [static], [constexpr]
```

Definition at line 17 of file MapGenConfig.h.

3.9.2.7 GROUND_PART_LENGTH

```
constexpr float MapGenConfig::GROUND_PART_LENGTH = 1.5  [static], [constexpr]
```

Definition at line 18 of file MapGenConfig.h.

3.9.2.8 GROUND_STARTING_X

```
constexpr float MapGenConfig::GROUND_STARTING_X = 0.0  [static], [constexpr]
```

Definition at line 15 of file MapGenConfig.h.

3.9.2.9 GROUND_STARTING_Y

```
constexpr float MapGenConfig::GROUND_STARTING_Y = 360.0  [static], [constexpr]
```

Definition at line 16 of file MapGenConfig.h.

3.9.2.10 MAX_GROUND_DEGREE

```
constexpr float MapGenConfig::MAX_GROUND_DEGREE = 50.0f  [static], [constexpr]
```

Definition at line 26 of file MapGenConfig.h.

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

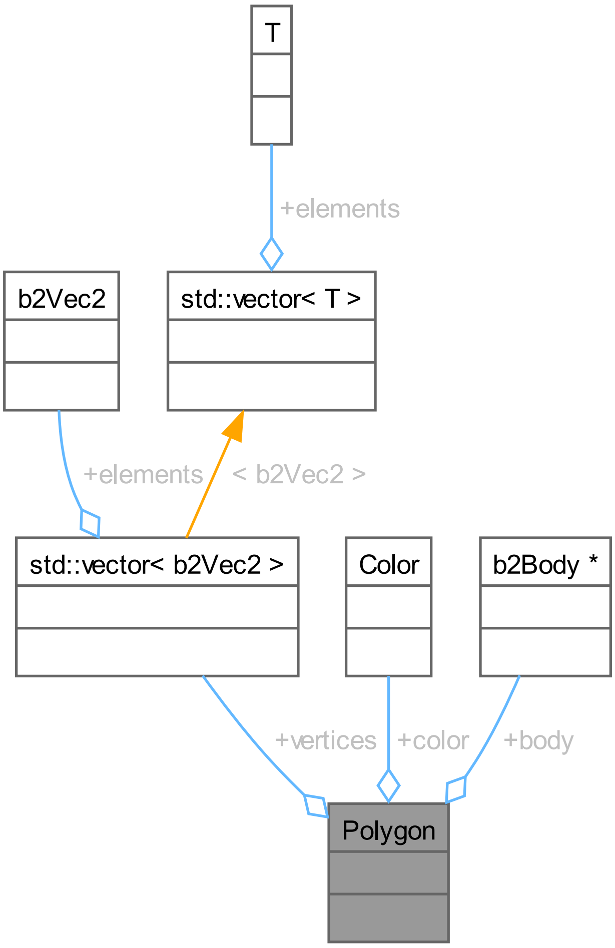
- config/MapGenConfig.h

3.10 Polygon Struct Reference

Struct representing a polygon.

```
#include <Shape.h>
```

Collaboration diagram for Polygon:



Public Attributes

- std::vector< b2Vec2 > vertices
- sf::Color color
- b2Body ∗ body

3.10.1 Detailed Description

Struct representing a polygon.

Definition at line 42 of file Shape.h.

3.10.2 Member Data Documentation

3.10.2.1 body

```
b2Body* Polygon::body
```

Definition at line 45 of file Shape.h.

3.10.2.2 color

```
sf::Color Polygon::color
```

Definition at line 44 of file Shape.h.

3.10.2.3 vertices

```
std::vector<b2Vec2> Polygon::vertices
```

Definition at line 43 of file Shape.h.

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

- src/Shape.h

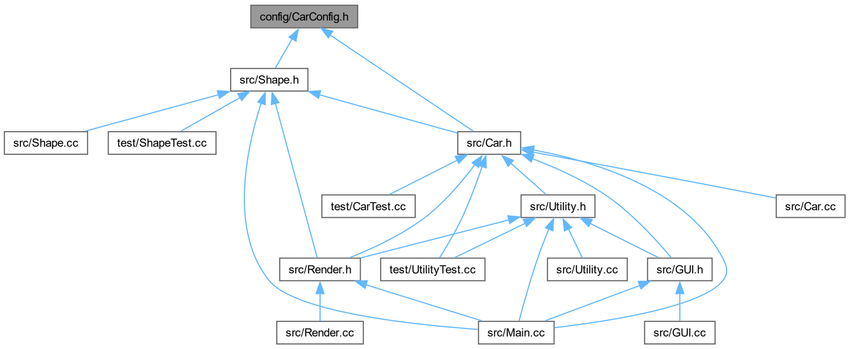
Chapter 4

File Documentation

4.1 config/CarConfig.h File Reference

This file contains all the constant values for the car class.

This graph shows which files directly or indirectly include this file:



Classes

- class CarConfig

4.1.1 Detailed Description

This file contains all the constant values for the car class.

Author

Mateusz Krakowski

Date

2023-06-06

Definition in file CarConfig.h.

4.2 CarConfig.h

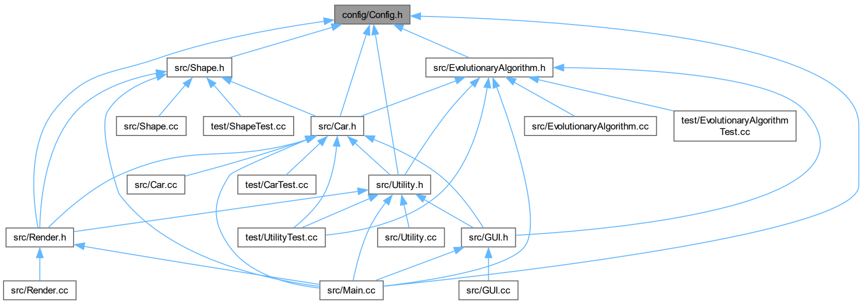
Go to the documentation of this file.

```
#ifndef CARCONFIG_H
#define CARCONFIG_H

class CarConfig {
   public:

       // Car speed is dependent on the car’s torque
    static constexpr float CAR_TORQUE = 2000.0f;


    static constexpr float MAX_JOINT_LENGTH = 0.01f;
    // Number of vertices in a car’s body polygon
    static constexpr int CAR_VERTICES = 8;

};

#endif  // CARCONFIG_H
```

4.3 config/Config.h File Reference

This file contains all the constant values used in the program.

This graph shows which files directly or indirectly include this file:



Classes

- class Config

4.3.1 Detailed Description

This file contains all the constant values used in the program.

Authors

Jakub Marcowski, Mateusz Krakowski

Date

2023-06-06

Definition in file Config.h.

4.4 Config.h

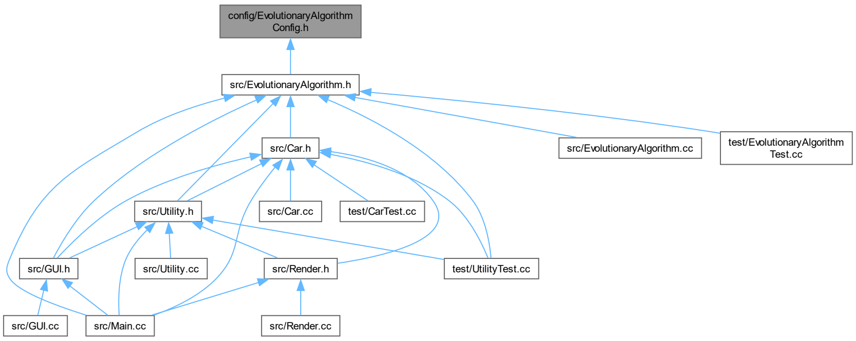
Go to the documentation of this file.

```
#ifndef CONFIG_H
#define CONFIG_H

class Config {
   public:
    static constexpr int WINDOW_WIDTH = 1280;
    static constexpr int WINDOW_HEIGHT = 720;

    // 60 for real time, 120 for fast forward - anything else is undefined behaviour
    static constexpr int MAX_FPS = 60;

    static constexpr int GROUND_PARTS_RENDERED = 32;

    // Exporting to file
    static constexpr bool SAVE_TO_FILE = true;
    static constexpr char SAVE_FILE_NAME[] = "evoRacerOutput.json";

    static constexpr int GENERATION_TIME =
        3000;  // in frames, about 60 frames per second => 50 seconds
    // Pixels per meter. Box2D uses metric units, so we need PPM for conversion purposes
    static constexpr float PPM = 30.0F;

    // SFML uses degrees for angles while Box2D uses radians
    static constexpr float DEG_PER_RAD = 57.2957795F;

    // Draw debug geometry
    static constexpr bool DEBUG = true;

    // Physics
    static constexpr double GRAVITATIONAL_ACCELERATION = -9.81f;
    static constexpr float FRICTION = 0.3f;
    static constexpr int VELOCITY_ARRAY_SIZE = 1000;

    static constexpr double PI = 3.14159265358979323846;
    static constexpr int BACK_WHEEL_POS = 1;
    static constexpr int FRONT_WHEEL_POS = 3;
    static constexpr int CATEGORY_BITS = 2;
    static constexpr int MASK_BITS = 1;
};

#endif  // CONFIG_H
```

4.5 config/EvolutionaryAlgorithmConfig.h File Reference

This file contains all the constant values used in the evolutionary algorithm.

This graph shows which files directly or indirectly include this file:



Classes

- class EvolutionaryAlgorithmConfig

4.5.1 Detailed Description

This file contains all the constant values used in the evolutionary algorithm.

Author

Mateusz Krakowski

Date

2023-06-06

Definition in file EvolutionaryAlgorithmConfig.h.

4.6 EvolutionaryAlgorithmConfig.h

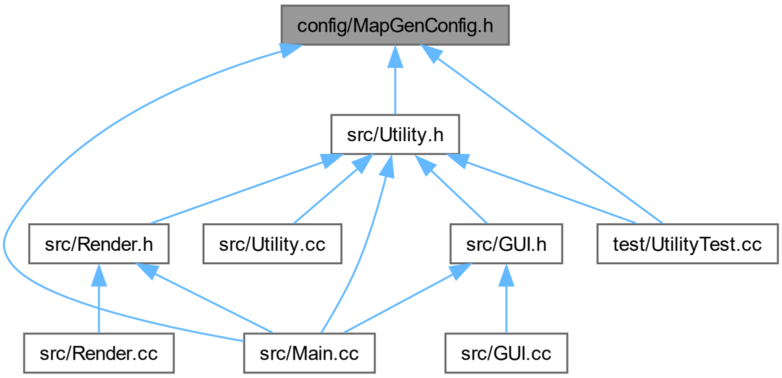
Go to the documentation of this file.

```

#ifndef EVOLUTIONARY_ALGORITHM_CONFIG_H
#define EVOLUTIONARY_ALGORITHM_CONFIG_H

class EvolutionaryAlgorithmConfig {
   public:
    // Evolutionary algorithm parameters
    static constexpr int POPULATION_SIZE = 15;

    // Boundaries for the chromosomes
    static constexpr float MIN_BODY_LENGTH = 1.0f;
    static constexpr float MAX_BODY_LENGTH = 5.0f;

    static constexpr float MIN_BODY_DENSITY = 10.0f;
    static constexpr float MAX_BODY_DENSITY = 1000.0f;

    static constexpr float MIN_WHEEL_RADIUS = 2.0f;
    static constexpr float MAX_WHEEL_RADIUS = 40.0f;

    static constexpr float MIN_WHEEL_DENSITY = 10.0f;
    static constexpr float MAX_WHEEL_DENSITY = 1000.0f;

    // Population initialization hyper parameters
    static constexpr float INITIAL_BODY_LENGTH_MEAN = 3.0f;
    static constexpr float INITIAL_BODY_LENGTH_VARIANCE = 1.0f;

    static constexpr float INITIAL_BODY_DENSITY_MEAN = 100.0f;
    static constexpr float INITIAL_BODY_DENSITY_VARIANCE = 100.0f;

    static constexpr float INITIAL_WHEEL_RADIUS_MEAN = 25.0f;
    static constexpr float INITIAL_WHEEL_RADIUS_VARIANCE = 10.0f;

    static constexpr float INITIAL_WHEEL_DENSITY_MEAN = 100.0f;
    static constexpr float INITIAL_WHEEL_DENSITY_VARIANCE = 100.0f;

    // Mutation hyper parameters

    static constexpr float MUTATION_RATE_BODY_LENGTHS = 0.1f;
    static constexpr float MUTATION_FACTOR_BODY_LENGTHS = 0.5f;

    static constexpr float MUTATION_RATE_BODY_DENSITY = 0.2f;
    static constexpr float MUTATION_FACTOR_BODY_DENSITY = 20.0f;

    static constexpr float MUTATION_RATE_WHEEL_RADIUS = 0.3f;
    static constexpr float MUTATION_FACTOR_WHEEL_RADIUS = 2.0f;

    static constexpr float MUTATION_RATE_WHEEL_DENSITY = 0.1f;
    static constexpr float MUTATION_FACTOR_WHEEL_DENSITY = 20.0f;

    // Selection hyper parameters
    static constexpr int TOURNAMENT_SIZE = 3;  // Has to be equal to or lesser than POPULATION_SIZE
};

#endif  // EVOLUTIONARY_ALGORITHM_CONFIG_H
```

4.7 config/MapGenConfig.h File Reference

This file contains all the constant values used in the map generation algorithm.

This graph shows which files directly or indirectly include this file:



Classes

- class MapGenConfig

4.7.1 Detailed Description

This file contains all the constant values used in the map generation algorithm.

Author

Mateusz Krakowski

Date

2023-06-06

Definition in file MapGenConfig.h.

4.8 MapGenConfig.h

Go to the documentation of this file.

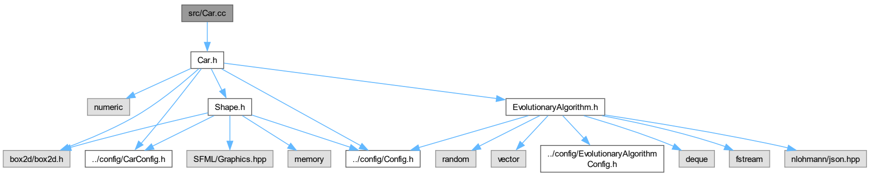
```

#ifndef MAPGENCONFIG_H
#define MAPGENCONFIG_H

class MapGenConfig {
   public:
    static constexpr float GENERATE_DISTANCE = 666.0;
    static constexpr float GROUND_STARTING_X = 0.0;
    static constexpr float GROUND_STARTING_Y = 360.0;
    static constexpr float GROUND_LEG_LENGTH = 4.0;
    static constexpr float GROUND_PART_LENGTH = 1.5;
    static constexpr int BG_SPRITES_COUNT = 5;

    static constexpr float CAR_STARTING_X = 250.0;
    static constexpr float CAR_STARTING_Y = 650.0;

    // Change the mapgen behaviour here
    static constexpr float GROUND_DEGREE_DEVIATION = 12.0f;
    static constexpr float MAX_GROUND_DEGREE = 50.0f;


};

#endif  // MAPGENCONFIG_H
```

4.9 src/Car.cc File Reference

Creates a car with a polygon (car's body) and two circles (front and back wheels).

`#include ¨Car.h¨`

Include dependency graph for Car.cc:



Functions

- std::vector< b2Vec2 > createVertices (std::vector< float > lengths)

  Create a vector of points for a polygon.

4.9.1 Detailed Description

Creates a car with a polygon (car's body) and two circles (front and back wheels).

Authors

Jakub Marcowski, Mateusz Krakowski

Date

2023-06-06

Definition in file Car.cc.

4.9.2 Function Documentation

4.9.2.1 createVertices()

```
std::vector< b2Vec2 > createVertices (
            std::vector< float > lengths )
```

Create a vector of points for a polygon.

Parameters

lengths	Vector of lengths from the center of the polygon to the vertices.

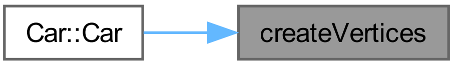
Returns

std::vector<b2Vec2> Vector of vertices.

Definition at line 95 of file Car.cc.

```
00095                                                        {
00096     std::vector<b2Vec2> vertices;
00097
00098     std::vector<float> angles;
00099     angles.reserve(lengths.size());
00100     for (int i = 0; i < lengths.size(); i++) {
00101         angles.push_back(360.0f / lengths.size());
00102     }
00103     // so that the wheels are set properly (that is - parallel to the ground)
00104     float angle = ((180.0f + (angles.back() / 2)) / 180.0f) * Config::PI;
00105
00106     for (int i = 0; i < lengths.size(); i++) {
00107         vertices.emplace_back(lengths[i] * cos(angle), lengths[i] * sin(angle));
00108         angle += (angles[i] / 180.0f) * Config::PI;
00109     }
00110     return vertices;
00111 }
```

Here is the caller graph for this function:



4.10 Car.cc

Go to the documentation of this file.

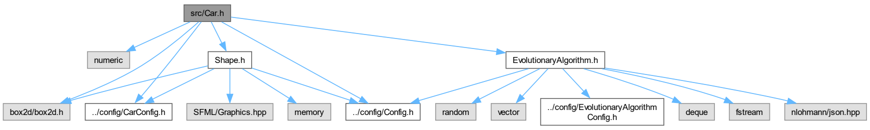
```
00001
00010 #include "Car.h"
00011
00012 Car::Car(const b2WorldPtr& world, float x, float y, const Chromosome& chromosome,
00013          sf::Color bodyColor, sf::Color wheelColor) {
00014     // Create a polygon (octagon)
00015
00016     auto vertices = createVertices(chromosome.bodyLengths);
00017
00018     body_ =
        createPolygon(world, x, y, vertices, chromosome.bodyDensity, Config::FRICTION, bodyColor);

    // Create a circle
    frontWheel_ = createCircle(world, x, y, chromosome.wheelRadius.first,
                               chromosome.wheelDensity.first, Config::FRICTION, wheelColor);

    // Create another circle
    backWheel_ = createCircle(world, x, y, chromosome.wheelRadius.second,
                              chromosome.wheelDensity.second, Config::FRICTION, wheelColor);

    b2DistanceJointDef jointDef2;
    jointDef2.bodyA = body_.body;
    jointDef2.bodyB = frontWheel_.body;
    jointDef2.localAnchorA = vertices[Config::BACK_WHEEL_POS];
    jointDef2.localAnchorB = b2Vec2(0.0f, 0.0f);
    jointDef2.maxLength = CarConfig::MAX_JOINT_LENGTH;
    jointDef2.collideConnected = false;
    world->CreateJoint(&jointDef2);

    jointDef2.bodyA = body_.body;
    jointDef2.bodyB = backWheel_.body;
    jointDef2.localAnchorA = vertices[Config::FRONT_WHEEL_POS];
    jointDef2.localAnchorB = b2Vec2(0.0f, 0.0f);
    jointDef2.maxLength = CarConfig::MAX_JOINT_LENGTH;
    jointDef2.collideConnected = false;
    world->CreateJoint(&jointDef2);

    // Make cars pass through each-other
    // by setting collision filtering
    b2Filter filter;
    filter.categoryBits = Config::CATEGORY_BITS;
    filter.maskBits = Config::MASK_BITS;
    this->setCollisionFilter(filter);

    std::vector<float> v_axis(Config::VELOCITY_ARRAY_SIZE);
    std::vector<float> v_values(Config::VELOCITY_ARRAY_SIZE);

    std::iota(std::begin(v_axis), std::end(v_axis), 1);

    velX_ = v_axis;
    velY_ = v_values;

    posX_ = v_axis;
    posY_ = v_values;
}

Polygon* Car::getBody() { return &body_; }

Circle* Car::getFrontWheel() { return &frontWheel_; }

Circle* Car::getBackWheel() { return &backWheel_; }

float Car::getPosX() const { return body_.body->GetPosition().x; }

float Car::getPosY() const { return body_.body->GetPosition().y; }

std::vector<float>* Car::getVelX() { return &velX_; }

std::vector<float>* Car::getVelY() { return &velY_; }

std::vector<float>* Car::getPosXVec() { return &posX_; }

std::vector<float>* Car::getPosYVec() { return &posY_; }

sf::Color Car::getBodyColor() const { return body_.color; }

b2Vec2 Car::getVelocityVec() const { return body_.body->GetLinearVelocity(); }

float Car::getVelocity() const { return body_.body->GetLinearVelocity().Length(); }

void Car::setCollisionFilter(b2Filter filter) const {
    body_.body->GetFixtureList()->SetFilterData(filter);
    frontWheel_.body->GetFixtureList()->SetFilterData(filter);
    backWheel_.body->GetFixtureList()->SetFilterData(filter);
}

std::vector<b2Vec2> createVertices(std::vector<float> lengths) {
    std::vector<b2Vec2> vertices;

    std::vector<float> angles;
    angles.reserve(lengths.size());
    for (int i = 0; i < lengths.size(); i++) {
        angles.push_back(360.0f / lengths.size());
    }
    // so that the wheels are set properly (that is - parallel to the ground)
    float angle = ((180.0f + (angles.back() / 2)) / 180.0f) * Config::PI;

    for (int i = 0; i < lengths.size(); i++) {
        vertices.emplace_back(lengths[i] * cos(angle), lengths[i] * sin(angle));
        angle += (angles[i] / 180.0f) * Config::PI;
    }
    return vertices;
}
```

4.11 src/Car.h File Reference

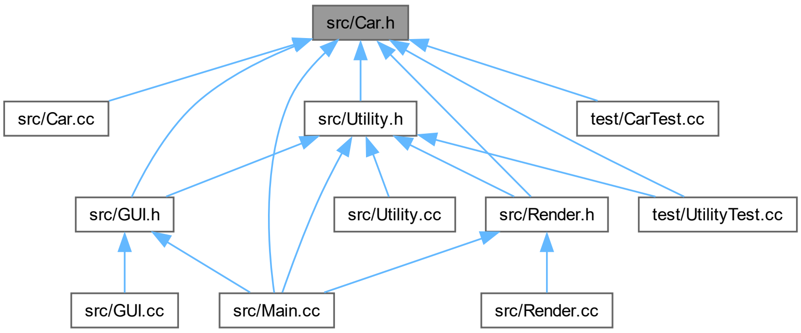
Header file for the Car class.

```
#include <numeric>
#include ¨box2d/box2d.h¨
#include ¨../config/Config.h¨
#include ¨../config/CarConfig.h¨
#include ¨Shape.h¨
#include ¨EvolutionaryAlgorithm.h¨
```

Include dependency graph for Car.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Car

  Class representing a car.

Typedefs

- typedef std::shared_ptr< b2World > b2WorldPtr

Functions

- std::vector< b2Vec2 > createVertices (std::vector< float > lengths)

  Create a vector of points for a polygon.

4.11.1 Detailed Description

Header file for the Car class.

Authors

Jakub Marcowski, Mateusz Krakowski

Date

2023-06-06

Definition in file Car.h.

4.11.2 Typedef Documentation

4.11.2.1 b2WorldPtr

```
typedef std::shared_ptr<b2World> b2WorldPtr
```

Definition at line 21 of file Car.h.

4.11.3 Function Documentation

4.11.3.1 createVertices()

```
std::vector< b2Vec2 > createVertices (
            std::vector< float > lengths )
```

Create a vector of points for a polygon.

Parameters

lengths	Vector of lengths from the center of the polygon to the vertices.

Returns

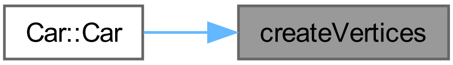
std::vector<b2Vec2> Vector of vertices.

Definition at line 95 of file Car.cc.

```
00095                                                          {
00096     std::vector<b2Vec2> vertices;
```

```

    std::vector<float> angles;
    angles.reserve(lengths.size());
    for (int i = 0; i < lengths.size(); i++) {
        angles.push_back(360.0f / lengths.size());
    }
    // so that the wheels are set properly (that is - parallel to the ground)
    float angle = ((180.0f + (angles.back() / 2)) / 180.0f) * Config::PI;

    for (int i = 0; i < lengths.size(); i++) {
        vertices.emplace_back(lengths[i] * cos(angle), lengths[i] * sin(angle));
        angle += (angles[i] / 180.0f) * Config::PI;
    }
    return vertices;
}
```

Here is the caller graph for this function:



4.12 Car.h

Go to the documentation of this file.

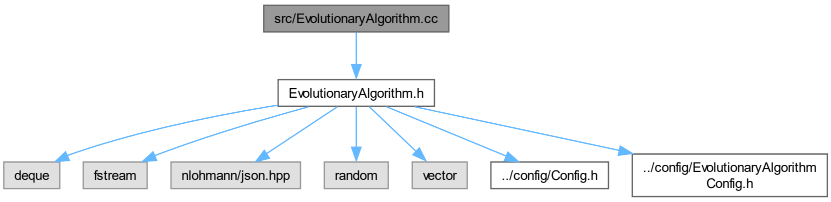
```

#ifndef CAR_H
#define CAR_H

#include <numeric>

#include "box2d/box2d.h"

#include "../config/Config.h"
#include "../config/CarConfig.h"
#include "Shape.h"
#include "EvolutionaryAlgorithm.h"

typedef std::shared_ptr<b2World> b2WorldPtr;

class Car {
   private:
    b2WorldPtr world_;
    Polygon body_;
    Circle frontWheel_;
    Circle backWheel_;
    std::vector<float> velX_;
    std::vector<float> velY_;
    std::vector<float> posX_;
    std::vector<float> posY_;

   public:
    Car(const b2WorldPtr& world, float x, float y, const Chromosome& chromosome,
        sf::Color bodyColor, sf::Color wheelColor);

    Polygon* getBody();

    Circle* getFrontWheel();

    Circle* getBackWheel();

    float getPosX() const;

    float getPosY() const;

    std::vector<float>* getVelX();

    std::vector<float>* getVelY();

    std::vector<float>* getPosXVec();

    std::vector<float>* getPosYVec();

    sf::Color getBodyColor() const;

    b2Vec2 getVelocityVec() const;

    float getVelocity() const;

    void setCollisionFilter(b2Filter filter) const;
};

std::vector<b2Vec2> createVertices(std::vector<float> lengths);

#endif
```

4.13 src/EvolutionaryAlgorithm.cc File Reference

Implementation file for EvolutionaryAlgorithm class, Algorithm used for evolving the cars.

`#include ¨EvolutionaryAlgorithm.h¨`

Include dependency graph for EvolutionaryAlgorithm.cc:



4.13.1 Detailed Description

Implementation file for EvolutionaryAlgorithm class, Algorithm used for evolving the cars.

Author

Mateusz Krakowski

Date

2023-06-06

Definition in file EvolutionaryAlgorithm.cc.

4.14 EvolutionaryAlgorithm.cc

Go to the documentation of this file.

```

#include "EvolutionaryAlgorithm.h"

EvolutionaryAlgorithm::EvolutionaryAlgorithm(int populationSize, bool saveToFile) {
    populationSize_ = populationSize;
    generation_ = 0;
    initializePopulation();
    saveToFile_ = saveToFile;
}

void EvolutionaryAlgorithm::initializePopulation() {
    std::random_device rd;
    std::mt19937 gen(rd());
    std::normal_distribution<float> dist(0.0, 1.0);
    // add variation and mean
    for (int i = 0; i < populationSize_; ++i) {
        Chromosome chrom;
        // TODO: change the 8
        for (int p = 0; p < 8; ++p) {
            float length = dist(gen) * EvolutionaryAlgorithmConfig::INITIAL_BODY_LENGTH_VARIANCE +
                           EvolutionaryAlgorithmConfig::INITIAL_BODY_LENGTH_MEAN;
            length = std::max(length, EvolutionaryAlgorithmConfig::MIN_BODY_LENGTH);
            length = std::min(length, EvolutionaryAlgorithmConfig::MAX_BODY_LENGTH);
            chrom.bodyLengths.push_back(length);
        }

        // Initialize bodyDensity

        chrom.bodyDensity = dist(gen) * EvolutionaryAlgorithmConfig::INITIAL_BODY_DENSITY_VARIANCE +
                            EvolutionaryAlgorithmConfig::INITIAL_BODY_DENSITY_MEAN;

        chrom.bodyDensity =
            std::max(chrom.bodyDensity, EvolutionaryAlgorithmConfig::MIN_BODY_DENSITY);
        chrom.bodyDensity =
            std::min(chrom.bodyDensity, EvolutionaryAlgorithmConfig::MAX_BODY_DENSITY);

        // initialize wheelRadius

        chrom.wheelRadius.first =
            dist(gen) * EvolutionaryAlgorithmConfig::INITIAL_WHEEL_RADIUS_VARIANCE +
            EvolutionaryAlgorithmConfig::INITIAL_WHEEL_RADIUS_MEAN;
        chrom.wheelRadius.first =
            std::max(chrom.wheelRadius.first, EvolutionaryAlgorithmConfig::MIN_WHEEL_RADIUS);
        chrom.wheelRadius.first =

            chrom.wheelRadius.second +=
            dist(gen) * EvolutionaryAlgorithmConfig::INITIAL_WHEEL_RADIUS_VARIANCE +
            EvolutionaryAlgorithmConfig::INITIAL_WHEEL_RADIUS_MEAN;
        chrom.wheelRadius.second =
            std::max(chrom.wheelRadius.second, EvolutionaryAlgorithmConfig::MIN_WHEEL_RADIUS);
        chrom.wheelRadius.second =
            std::min(chrom.wheelRadius.second, EvolutionaryAlgorithmConfig::MAX_WHEEL_RADIUS);

        // Initialize wheelDensity

        chrom.wheelDensity.first +=
            dist(gen) * EvolutionaryAlgorithmConfig::INITIAL_WHEEL_DENSITY_VARIANCE +
            EvolutionaryAlgorithmConfig::INITIAL_WHEEL_DENSITY_MEAN;
        chrom.wheelDensity.first =
            std::max(chrom.wheelDensity.first, EvolutionaryAlgorithmConfig::MIN_WHEEL_DENSITY);
        chrom.wheelDensity.first =
            std::min(chrom.wheelDensity.first, EvolutionaryAlgorithmConfig::MAX_WHEEL_DENSITY);

        chrom.wheelDensity.second +=
            dist(gen) * EvolutionaryAlgorithmConfig::INITIAL_WHEEL_DENSITY_VARIANCE +
            EvolutionaryAlgorithmConfig::INITIAL_WHEEL_DENSITY_MEAN;
        chrom.wheelDensity.second =
            std::max(chrom.wheelDensity.second, EvolutionaryAlgorithmConfig::MIN_WHEEL_DENSITY);
        chrom.wheelDensity.second =
            std::min(chrom.wheelDensity.second, EvolutionaryAlgorithmConfig::MAX_WHEEL_DENSITY);

        population_.push_back(chrom);
    }
}

void EvolutionaryAlgorithm::mutate() {
    std::random_device rd;
    std::mt19937 gen(rd());
    std::normal_distribution<float> dist(0.0, 1.0);

    for (auto& chrom : population_) {
        // Mutate bodyLengths
        for (auto& length : chrom.bodyLengths) {
            if (dist(gen) < EvolutionaryAlgorithmConfig::MUTATION_RATE_BODY_LENGTHS) {
                length += dist(gen) * EvolutionaryAlgorithmConfig::MUTATION_FACTOR_BODY_LENGTHS;
                length = std::max(length, EvolutionaryAlgorithmConfig::MIN_BODY_LENGTH);
                length = std::min(length, EvolutionaryAlgorithmConfig::MAX_BODY_LENGTH);
            }
        }

        // Mutate bodyDensity
        if (dist(gen) < EvolutionaryAlgorithmConfig::MUTATION_RATE_BODY_DENSITY) {
            chrom.bodyDensity +=
                dist(gen) * EvolutionaryAlgorithmConfig::MUTATION_FACTOR_BODY_DENSITY;

            chrom.bodyDensity =
                std::max(chrom.bodyDensity, EvolutionaryAlgorithmConfig::MIN_BODY_DENSITY);
            chrom.bodyDensity =
                std::min(chrom.bodyDensity, EvolutionaryAlgorithmConfig::MAX_BODY_DENSITY);
        }

        // Mutate wheelRadius
        if (dist(gen) < EvolutionaryAlgorithmConfig::MUTATION_RATE_WHEEL_RADIUS) {
            chrom.wheelRadius.first +=
                dist(gen) * EvolutionaryAlgorithmConfig::MUTATION_FACTOR_WHEEL_RADIUS;
            chrom.wheelRadius.first =
                std::max(chrom.wheelRadius.first, EvolutionaryAlgorithmConfig::MIN_WHEEL_RADIUS);
            chrom.wheelRadius.first =
                std::min(chrom.wheelRadius.first, EvolutionaryAlgorithmConfig::MAX_WHEEL_RADIUS);
        }
        if (dist(gen) < EvolutionaryAlgorithmConfig::MUTATION_RATE_WHEEL_RADIUS) {
            chrom.wheelRadius.second +=
                dist(gen) * EvolutionaryAlgorithmConfig::MUTATION_FACTOR_WHEEL_RADIUS;
            chrom.wheelRadius.second =
                std::max(chrom.wheelRadius.second, EvolutionaryAlgorithmConfig::MIN_WHEEL_RADIUS);
            chrom.wheelRadius.second =
                std::min(chrom.wheelRadius.second, EvolutionaryAlgorithmConfig::MAX_WHEEL_RADIUS);
        }

        // Mutate wheelDensity
        if (dist(gen) < EvolutionaryAlgorithmConfig::MUTATION_RATE_WHEEL_DENSITY) {
            chrom.wheelDensity.first +=
                dist(gen) * EvolutionaryAlgorithmConfig::MUTATION_FACTOR_WHEEL_DENSITY;
            chrom.wheelDensity.first =
                std::max(chrom.wheelDensity.first, EvolutionaryAlgorithmConfig::MIN_WHEEL_DENSITY);
            chrom.wheelDensity.first =
                std::min(chrom.wheelDensity.first, EvolutionaryAlgorithmConfig::MAX_WHEEL_DENSITY);
        }
        if (dist(gen) < EvolutionaryAlgorithmConfig::MUTATION_RATE_WHEEL_DENSITY) {
            chrom.wheelDensity.second +=
                dist(gen) * EvolutionaryAlgorithmConfig::MUTATION_FACTOR_WHEEL_DENSITY;
            chrom.wheelDensity.second =
                std::max(chrom.wheelDensity.second, EvolutionaryAlgorithmConfig::MIN_WHEEL_DENSITY);
            chrom.wheelDensity.second =
                std::min(chrom.wheelDensity.second, EvolutionaryAlgorithmConfig::MAX_WHEEL_DENSITY);
        }
    }
}
Chromosome EvolutionaryAlgorithm::tournament() {
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_int_distribution<> uniform_dist(0, populationSize_ - 1);

    std::vector<Chromosome> candidates;

    candidates.reserve(EvolutionaryAlgorithmConfig::TOURNAMENT_SIZE);
    for (int i = 0; i < EvolutionaryAlgorithmConfig::TOURNAMENT_SIZE; ++i) {
        candidates.push_back(population_[uniform_dist(gen)]);
    }

    Chromosome tournament_winner = candidates[0];

    for (int i = 1; i < EvolutionaryAlgorithmConfig::TOURNAMENT_SIZE; ++i) {
        if (candidates[i].fitness > tournament_winner.fitness) {
            tournament_winner = candidates[i];
        }
    }

    return tournament_winner;
}
void EvolutionaryAlgorithm::tournamentSelection() {
    std::vector<Chromosome> tournament_winners;

    tournament_winners.reserve(populationSize_);
    for (int i = 0; i < populationSize_; ++i) {
        tournament_winners.push_back(tournament());
    }

    population_ = tournament_winners;
```

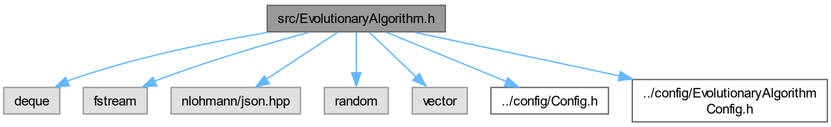
```
}

void EvolutionaryAlgorithm::nextGeneration() {
    if (saveToFile_) {
        exportPopulation();
    }
    tournamentSelection();
    mutate();
    ++generation_;
}

int EvolutionaryAlgorithm::exportPopulation() {
    nlohmann::json jsonData;
    jsonData["generation"] = generation_;

    std::deque<nlohmann::json> populationData;

    for (const auto& chromosome : population_) {
        nlohmann::json chromosomeJson;
        chromosomeJson["bodyLengths"] = chromosome.bodyLengths;
        chromosomeJson["bodyDensity"] = chromosome.bodyDensity;
        chromosomeJson["wheelRadius"] = {chromosome.wheelRadius.first,
                                         chromosome.wheelRadius.second};
        chromosomeJson["wheelDensity"] = {chromosome.wheelDensity.first,
                                          chromosome.wheelDensity.second};
        chromosomeJson["fitness"] = chromosome.fitness;
        populationData.push_front(chromosomeJson);
    }

    jsonData["population"] = populationData;

    std::string jsonString = jsonData.dump(4);

    std::ofstream outputFile(Config::SAVE_FILE_NAME, std::ios::app);
    if (!outputFile.is_open()) {
        return 1;
    }
    outputFile « jsonString;
    outputFile.close();

    return 0;
}
```

4.15 src/EvolutionaryAlgorithm.h File Reference

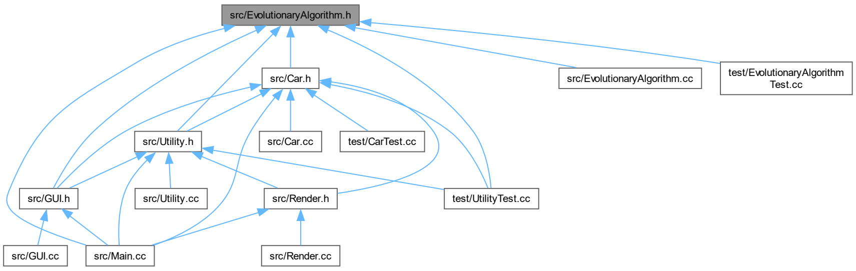
Header file for EvolutionaryAlgorithm class.

```
#include <deque>
#include <fstream>
#include <nlohmann/json.hpp>
#include <random>
#include <vector>
#include ¨../config/Config.h¨
#include ¨../config/EvolutionaryAlgorithmConfig.h¨
```

Include dependency graph for EvolutionaryAlgorithm.h:


src/EvolutionaryAlgorithm.h
deque
fstream
nlohmann/json.hpp
random
vector
../config/Config.h
../config/EvolutionaryAlgorithm Config.h

This graph shows which files directly or indirectly include this file:



Classes

- struct Chromosome
- class EvolutionaryAlgorithm

4.15.1 Detailed Description

Header file for EvolutionaryAlgorithm class.

Author

Mateusz Krakowski

Date

2023-06-06

Definition in file EvolutionaryAlgorithm.h.

4.16 EvolutionaryAlgorithm.h

Go to the documentation of this file.

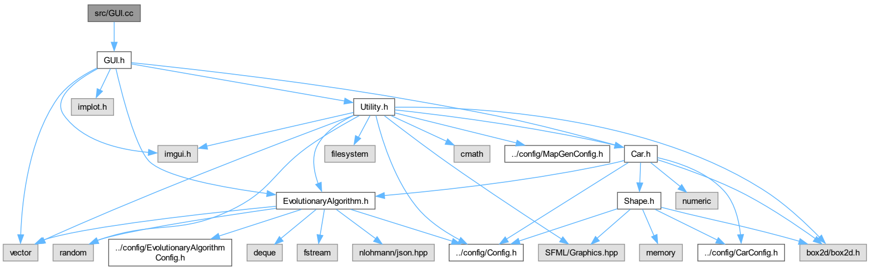
```

#ifndef GENOME_H
#define GENOME_H

#include <deque>
#include <fstream>
#include <nlohmann/json.hpp>
#include <random>
#include <vector>

#include "../config/Config.h"
#include "../config/EvolutionaryAlgorithmConfig.h"

struct Chromosome {
    std::vector<float> bodyLengths;
    float bodyDensity;
    std::pair<float, float> wheelRadius;
    std::pair<float, float> wheelDensity;
    float fitness;
};

class EvolutionaryAlgorithm {
   private:
    unsigned long int generation_;
    unsigned long int populationSize_;
    std::vector<Chromosome> population_;
    bool saveToFile_ = false;
    Chromosome tournament();
    void initializePopulation();

   public:
    explicit EvolutionaryAlgorithm(int populationSize, bool saveToFile = false);
    std::vector<Chromosome> getPopulation() { return population_; }
    void mutate();
    void tournamentSelection();
    void nextGeneration();
    void setFitness(int index, float fitness) { population_[index].fitness = fitness; }
    int getGeneration() const { return generation_; }
    int getPopulationSize() const { return populationSize_; }
    int exportPopulation();
};

#endif
```

4.17 src/GUI.cc File Reference

File containing GUI functions.

`#include "GUI.h"`

Include dependency graph for GUI.cc:



Functions

- void renderVelocityPlot (std::vector< Car > &cars, bool paused)

  Renders velocity plot.
- void renderPositionPlot (std::vector< Car > &cars, bool paused)

  Renders position plot.
- void printEAInfo (EvolutionaryAlgorithm &ea)

  Renders Evolutionary Algorithm's inner state.

4.17.1 Detailed Description

File containing GUI functions.

Author

Jakub Marcowski

Date

2023-06-06

Definition in file GUI.cc.

4.17.2 Function Documentation

4.17.2.1 printEAInfo()

```
void printEAInfo (
            EvolutionaryAlgorithm & ea )
```

Renders Evolutionary Algorithm's inner state.

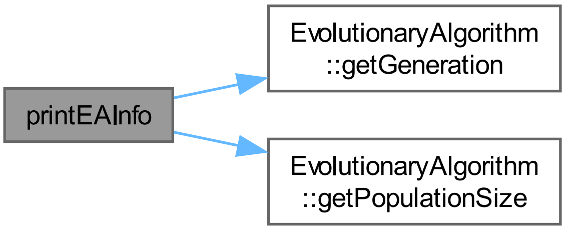
Parameters

ea	Evolutionary Algorithm object.

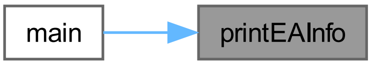
Definition at line 67 of file GUI.cc.

```
00067                                       {
00068     ImGui::Begin("EA Info");
00069     ImGui::Text("Generation: %d", ea.getGeneration());
00070     ImGui::Text("Population size: %d", ea.getPopulationSize());
00071     ImGui::End();
00072 }
```

Here is the call graph for this function:


printEAInfo
EvolutionaryAlgorithm ::getGeneration
EvolutionaryAlgorithm ::getPopulationSize

Here is the caller graph for this function:



4.17.2.2 renderPositionPlot()

```
void renderPositionPlot (
            std::vector< Car > & cars,
            bool paused )
```

Renders position plot.

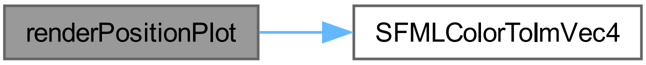
Parameters

cars	Vector of cars.
paused	Whether or not simulation is paused.

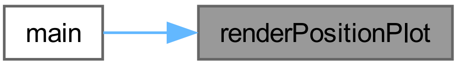
Definition at line 38 of file GUI.cc.

```
00038                                                                  {
00039     ImGui::Begin("Cars' Position");
00040     ImPlot::SetNextAxesToFit();
00041     if (ImPlot::BeginPlot("Position")) {
00042         ImPlot::SetupLegend(ImPlotLocation_West, ImPlotLegendFlags_Outside);
00043         for (int i = 0; i < cars.size(); ++i) {
00044             char i_str[11];
00045             sprintf(i_str, "%d", i);
00046 
00047             if (!paused) {
00048                 cars[i].getPosXVec()->push_back(cars[i].getPosXVec()->back() + 1);
00049                 cars[i].getPosYVec()->push_back(cars[i].getBody()->body->GetPosition().x);
00050             }
00051             std::vector<float> v_axis_crop =
00052                 std::vector<float>(cars[i].getPosXVec()->end() - Config::VELOCITY_ARRAY_SIZE,
00053                                    cars[i].getPosXVec()->end());
00054             std::vector<float> v_values_crop =
00055                 std::vector<float>(cars[i].getPosYVec()->end() - Config::VELOCITY_ARRAY_SIZE,
00056                                    cars[i].getPosYVec()->end());
00057             ImPlot::PushStyleColor(ImPlotCol_Line, SFMLColorToImVec4(cars[i].getBodyColor()));
00058             ImPlot::PlotLine(i_str, &(v_axis_crop[0]), &(v_values_crop[0]),
00059                              Config::VELOCITY_ARRAY_SIZE);
00060             ImPlot::PopStyleColor();
00061         }
00062         ImPlot::EndPlot();
00063     }
00064     ImGui::End();
00065 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.17.2.3 renderVelocityPlot()

```
void renderVelocityPlot (
            std::vector< Car > & cars,
            bool paused )
```

Renders velocity plot.

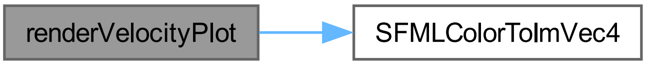
Parameters

cars	Vector of cars.
paused	Whether or not simulation is paused.

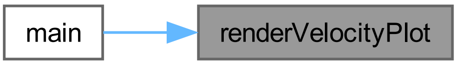
Definition at line 11 of file GUI.cc.

```
00011                                                             {
00012     ImGui::Begin("Cars’ Velocity");
00013     ImPlot::SetNextAxesToFit();
00014     if (ImPlot::BeginPlot("Velocity")) {
00015         ImPlot::SetupLegend(ImPlotLocation_West, ImPlotLegendFlags_Outside);
00016         for (int i = 0; i < cars.size(); ++i) {
00017             char i_str[11];  // 10 digits + null
00018             sprintf(i_str, "%d", i);
00019 
00020             if (!paused) {
00021                 cars[i].getVelX()->push_back(cars[i].getVelX()->back() + 1);
00022                 cars[i].getVelY()->push_back(cars[i].getVelocity());
00023             }
00024             std::vector<float> v_axis_crop = std::vector<float>(
00025                 cars[i].getVelX()->end() - Config::VELOCITY_ARRAY_SIZE, cars[i].getVelX()->end());
00026             std::vector<float> v_values_crop = std::vector<float>(
00027                 cars[i].getVelY()->end() - Config::VELOCITY_ARRAY_SIZE, cars[i].getVelY()->end());
00028             ImPlot::PushStyleColor(ImPlotCol_Line, SFMLColorToImVec4(cars[i].getBodyColor()));
00029             ImPlot::PlotLine(i_str, &(v_axis_crop[0]), &(v_values_crop[0]),
00030                              Config::VELOCITY_ARRAY_SIZE);
00031             ImPlot::PopStyleColor();
00032         }
00033         ImPlot::EndPlot();
00034     }
00035     ImGui::End();
00036 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.18 GUI.cc

Go to the documentation of this file.

```
00001 
00009 #include "GUI.h"
```

```
void renderVelocityPlot(std::vector<Car>& cars, bool paused) {
    ImGui::Begin("Cars' Velocity");
    ImPlot::SetNextAxesToFit();
    if (ImPlot::BeginPlot("Velocity")) {
        ImPlot::SetupLegend(ImPlotLocation_West, ImPlotLegendFlags_Outside);
        for (int i = 0; i < cars.size(); ++i) {
            char i_str[11];  // 10 digits + null
            sprintf(i_str, "%d", i);

            if (!paused) {
                cars[i].getVelX()->push_back(cars[i].getVelX()->back() + 1);
                cars[i].getVelY()->push_back(cars[i].getVelocity());
            }
            std::vector<float> v_axis_crop = std::vector<float>(
                cars[i].getVelX()->end() - Config::VELOCITY_ARRAY_SIZE, cars[i].getVelX()->end());
            std::vector<float> v_values_crop = std::vector<float>(
                cars[i].getVelY()->end() - Config::VELOCITY_ARRAY_SIZE, cars[i].getVelY()->end());
            ImPlot::PushStyleColor(ImPlotCol_Line, SFMLColorToImVec4(cars[i].getBodyColor()));
            ImPlot::PlotLine(i_str, &(v_axis_crop[0]), &(v_values_crop[0]),
                             Config::VELOCITY_ARRAY_SIZE);
            ImPlot::PopStyleColor();
        }
        ImPlot::EndPlot();
    }
    ImGui::End();
}

void renderPositionPlot(std::vector<Car>& cars, bool paused) {
    ImGui::Begin("Cars' Position");
    ImPlot::SetNextAxesToFit();
    if (ImPlot::BeginPlot("Position")) {
        ImPlot::SetupLegend(ImPlotLocation_West, ImPlotLegendFlags_Outside);
        for (int i = 0; i < cars.size(); ++i) {
            char i_str[11];
            sprintf(i_str, "%d", i);

            if (!paused) {
                cars[i].getPosXVec()->push_back(cars[i].getPosXVec()->back() + 1);
                cars[i].getPosYVec()->push_back(cars[i].getBody()->body->GetPosition().x);
            }
            std::vector<float> v_axis_crop =
                std::vector<float>(cars[i].getPosXVec()->end() - Config::VELOCITY_ARRAY_SIZE,
                                   cars[i].getPosXVec()->end());
            std::vector<float> v_values_crop =
                std::vector<float>(cars[i].getPosYVec()->end() - Config::VELOCITY_ARRAY_SIZE,
                                   cars[i].getPosYVec()->end());
            ImPlot::PushStyleColor(ImPlotCol_Line, SFMLColorToImVec4(cars[i].getBodyColor()));
            ImPlot::PlotLine(i_str, &(v_axis_crop[0]), &(v_values_crop[0]),
                             Config::VELOCITY_ARRAY_SIZE);
            ImPlot::PopStyleColor();
        }
        ImPlot::EndPlot();
    }
    ImGui::End();
}

void printEAInfo(EvolutionaryAlgorithm& ea) {
    ImGui::Begin("EA Info");
    ImGui::Text("Generation: %d", ea.getGeneration());
    ImGui::Text("Population size: %d", ea.getPopulationSize());
    ImGui::End();
}
```

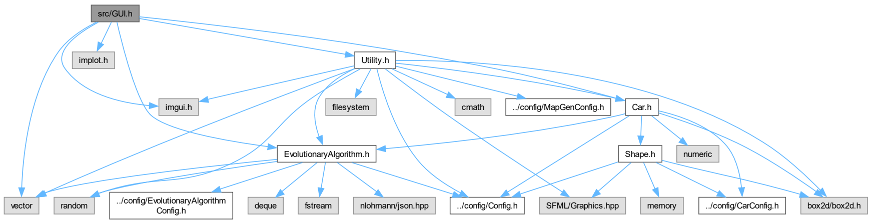
4.19 src/GUI.h File Reference

Header for a file containing GUI functions.

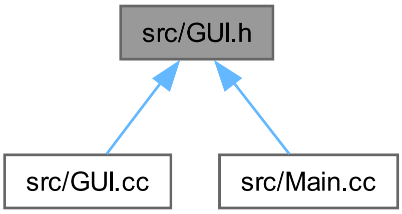
```
#include <vector>
#include ¨imgui.h¨
#include ¨implot.h¨
#include ¨Car.h¨
#include ¨EvolutionaryAlgorithm.h¨
```

```
#include ¨Utility.h¨
```

Include dependency graph for GUI.h:



This graph shows which files directly or indirectly include this file:



Functions

- void renderVelocityPlot (std::vector< Car > &cars, bool paused)

  Renders velocity plot.
- void renderPositionPlot (std::vector< Car > &cars, bool paused)

  Renders position plot.
- void printEAInfo (EvolutionaryAlgorithm &ea)

  Renders Evolutionary Algorithm's inner state.

4.19.1 Detailed Description

Header for a file containing GUI functions.

Author

Jakub Marcowski

Date

2023-06-06

Definition in file GUI.h.

4.19.2 Function Documentation

4.19.2.1 printEAInfo()

```
void printEAInfo (
            EvolutionaryAlgorithm & ea )
```

Renders Evolutionary Algorithm's inner state.

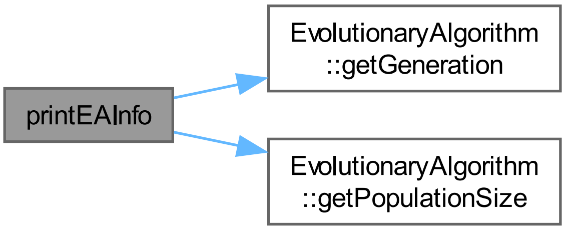
Parameters

ea	Evolutionary Algorithm object.

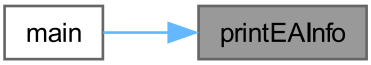
Definition at line 67 of file GUI.cc.

```
00067                                              {
00068     ImGui::Begin("EA Info");
00069     ImGui::Text("Generation: %d", ea.getGeneration());
00070     ImGui::Text("Population size: %d", ea.getPopulationSize());
00071     ImGui::End();
00072 }
```

Here is the call graph for this function:


EvolutionaryAlgorithm ::getGeneration
printEAInfo
EvolutionaryAlgorithm ::getPopulationSize

Here is the caller graph for this function:



4.19.2.2 renderPositionPlot()

```
void renderPositionPlot (
            std::vector< Car > & cars,
            bool paused )
```

Renders position plot.

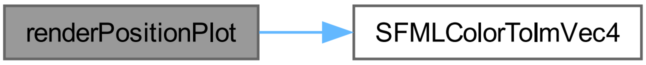
Parameters

cars	Vector of cars.
paused	Whether or not simulation is paused.

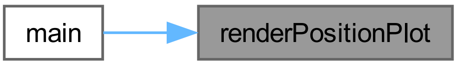
Definition at line 38 of file GUI.cc.

```
00038                                                             {
00039     ImGui::Begin("Cars’ Position");
00040     ImPlot::SetNextAxesToFit();
00041     if (ImPlot::BeginPlot("Position")) {
00042         ImPlot::SetupLegend(ImPlotLocation_West, ImPlotLegendFlags_Outside);
00043         for (int i = 0; i < cars.size(); ++i) {
00044             char i_str[11];
00045             sprintf(i_str, "%d", i);
00046 
00047             if (!paused) {
00048                 cars[i].getPosXVec()->push_back(cars[i].getPosXVec()->back() + 1);
00049                 cars[i].getPosYVec()->push_back(cars[i].getBody()->body->GetPosition().x);
00050             }
00051             std::vector<float> v_axis_crop =
00052                 std::vector<float>(cars[i].getPosXVec()->end() - Config::VELOCITY_ARRAY_SIZE,
00053                                    cars[i].getPosXVec()->end());
00054             std::vector<float> v_values_crop =
00055                 std::vector<float>(cars[i].getPosYVec()->end() - Config::VELOCITY_ARRAY_SIZE,
00056                                    cars[i].getPosYVec()->end());
00057             ImPlot::PushStyleColor(ImPlotCol_Line, SFMLColorToImVec4(cars[i].getBodyColor()));
00058             ImPlot::PlotLine(i_str, &(v_axis_crop[0]), &(v_values_crop[0]),
00059                              Config::VELOCITY_ARRAY_SIZE);
00060             ImPlot::PopStyleColor();
00061         }
00062         ImPlot::EndPlot();
00063     }
00064     ImGui::End();
00065 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.19.2.3 renderVelocityPlot()

```
void renderVelocityPlot (
            std::vector< Car > & cars,
            bool paused )
```

Renders velocity plot.

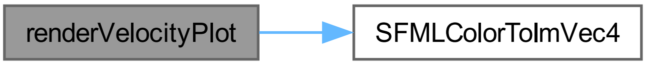
Parameters

cars	Vector of cars.
paused	Whether or not simulation is paused.

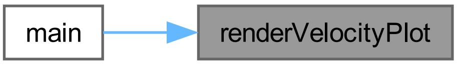
Definition at line 11 of file GUI.cc.

```
00011                                                             {
00012     ImGui::Begin("Cars' Velocity");
00013     ImPlot::SetNextAxesToFit();
00014     if (ImPlot::BeginPlot("Velocity")) {
00015         ImPlot::SetupLegend(ImPlotLocation_West, ImPlotLegendFlags_Outside);
00016         for (int i = 0; i < cars.size(); ++i) {
00017             char i_str[11];  // 10 digits + null
00018             sprintf(i_str, "%d", i);
00019 
00020             if (!paused) {
00021                 cars[i].getVelX()->push_back(cars[i].getVelX()->back() + 1);
00022                 cars[i].getVelY()->push_back(cars[i].getVelocity());
00023             }
00024             std::vector<float> v_axis_crop = std::vector<float>(
00025                 cars[i].getVelX()->end() - Config::VELOCITY_ARRAY_SIZE, cars[i].getVelX()->end());
00026             std::vector<float> v_values_crop = std::vector<float>(
00027                 cars[i].getVelY()->end() - Config::VELOCITY_ARRAY_SIZE, cars[i].getVelY()->end());
00028             ImPlot::PushStyleColor(ImPlotCol_Line, SFMLColorToImVec4(cars[i].getBodyColor()));
00029             ImPlot::PlotLine(i_str, &(v_axis_crop[0]), &(v_values_crop[0]),
00030                              Config::VELOCITY_ARRAY_SIZE);
00031             ImPlot::PopStyleColor();
00032         }
00033         ImPlot::EndPlot();
00034     }
00035     ImGui::End();
00036 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.20 GUI.h

Go to the documentation of this file.

```
00001 
00009 #ifndef GUI_H
```

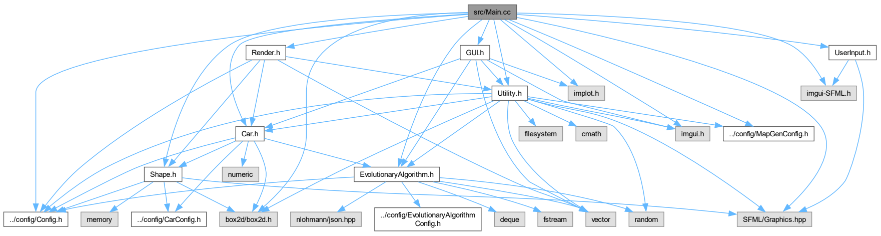
```
#define GUI_H

#include <vector>

#include "imgui.h"
#include "implot.h"

#include "Car.h"
#include "EvolutionaryAlgorithm.h"
#include "Utility.h"

void renderVelocityPlot(std::vector<Car>& cars, bool paused);

void renderPositionPlot(std::vector<Car>& cars, bool paused);

void printEAInfo(EvolutionaryAlgorithm& ea);

#endif  // GUI_H
```

4.21 src/Main.cc File Reference

Main file for the project, contains the main loop.

```
#include ¨box2d/box2d.h¨
#include ¨imgui.h¨
#include ¨imgui-SFML.h¨
#include ¨implot.h¨
#include ¨SFML/Graphics.hpp¨
#include ¨../config/Config.h¨
#include ¨../config/MapGenConfig.h¨
#include ¨Car.h¨
#include ¨EvolutionaryAlgorithm.h¨
#include ¨GUI.h¨
#include ¨Render.h¨
#include ¨Shape.h¨
#include ¨UserInput.h¨
#include ¨Utility.h¨
```

Include dependency graph for Main.cc:



Typedefs

- typedef std::shared_ptr< b2World > b2WorldPtr

Functions

- int main ()

Variables

- b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION))

4.21.1 Detailed Description

Main file for the project, contains the main loop.

Authors

Jakub Marcowski, Mateusz Krakowski

Date

2023-06-06

Definition in file Main.cc.

4.21.2 Typedef Documentation

4.21.2.1 b2WorldPtr

```
typedef std::shared_ptr<b2World> b2WorldPtr
```

Definition at line 25 of file Main.cc.

4.21.3 Function Documentation

4.21.3.1 main()

```
int main ( )
```

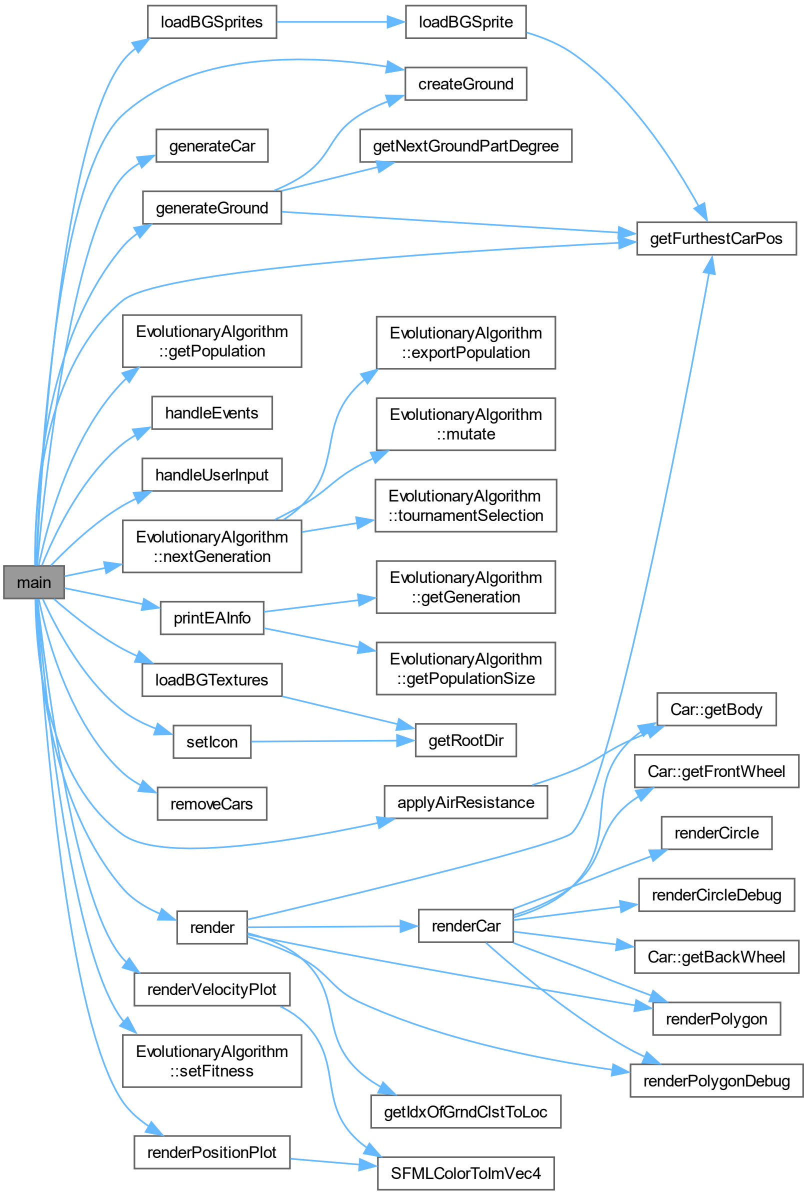
PROGRAM LOOP

Definition at line 30 of file Main.cc.

```
00030        {
00031     sf::ContextSettings settings;
00032     settings.antialiasingLevel = 8;
00033
00034     // Setup SFML window
00035     sf::RenderWindow w(sf::VideoMode(Config::WINDOW_WIDTH, Config::WINDOW_HEIGHT), "EvoRacer",
00036                        sf::Style::Default, settings);
00037     w.setFramerateLimit(Config::MAX_FPS);
00038
00039     // Initialize ImGui and all its friends
00040     ImGui::SFML::Init(w);
00041     ImPlot::CreateContext();
00042
00043     // Change imgui.ini location
00044     ImGui::GetIO().IniFilename = "./imgui.ini";
00045
00046     // Containers to hold objects we create
00047     std::vector<Polygon> groundVector;
00048     std::vector<Car> cars;
00049
00050     // Generate ground
00051     std::vector<b2Vec2> groundVertices = {b2Vec2(0, 0), b2Vec2(MapGenConfig::GROUND_PART_LENGTH, 0),
00052                                           b2Vec2(0, -MapGenConfig::GROUND_LEG_LENGTH)};
```

```
    Polygon ground =
        createGround(world, MapGenConfig::GROUND_STARTING_X, MapGenConfig::GROUND_STARTING_Y,
                     groundVertices, sf::Color(18, 36, 35));
    groundVector.push_back(ground);

    EvolutionaryAlgorithm ea(EvolutionaryAlgorithmConfig::POPULATION_SIZE, Config::SAVE_TO_FILE);

    for (const Chromosome& chromosome : ea.getPopulation()) {
        cars.push_back(generateCar(world, chromosome));
    }

    bool paused = false;     // Should we pause the simulation?
    bool pauseCheck = true;  // Should we check if the user wants to flip `paused'?
    bool nextGen = false;    // Should we generate the next generation?
    bool nextGCheck = true;  // Should we check if the user wants to flip `nextGen'?
    bool focus = true;       // Is the window in focus? (used to prevent input when not in focus)
    int timer = 0;

    // Set window icon
    setIcon(w);

    auto textures = loadBGTextures();
    auto sprites = loadBGSprites(textures, cars);

    sf::Clock deltaClock;
    while (w.isOpen()) {
        // Update the world, standard arguments
        if (!paused) {
            world->Step(1 / 60.0f, 6, 3);
            ++timer;
            if (timer >= Config::GENERATION_TIME) {
                nextGen = true;
            }
        }

        if (nextGen) {
            nextGen = false;
            for (int i = 0; i < cars.size(); ++i) {
                ea.setFitness(i, cars[i].getPosX());
            }
            ea.nextGeneration();
            removeCars(world, &cars);
            for (const Chromosome& chromosome : ea.getPopulation()) {
                cars.push_back(generateCar(world, chromosome));
            }
            timer = 0;
        }

        // Render everything
        render(w, sprites, groundVector, cars);

        ImGui::SFML::Update(w, deltaClock.restart());

        ImGui::PushStyleColor(ImGuiCol_WindowBg, ImVec4(0.071f, 0.141f, 0.137f, 0.5f));

        ImGui::SetNextWindowSize(ImVec2(340, 340), ImGuiCond_FirstUseEver);
        ImGui::SetNextWindowPos(ImVec2(10, 10), ImGuiCond_FirstUseEver);
        renderVelocityPlot(cars, paused);

        ImGui::SetNextWindowSize(ImVec2(340, 340), ImGuiCond_FirstUseEver);
        ImGui::SetNextWindowPos(ImVec2(930, 10), ImGuiCond_FirstUseEver);
        renderPositionPlot(cars, paused);

        ImGui::SetNextWindowSize(ImVec2(175, 75), ImGuiCond_FirstUseEver);
        ImGui::SetNextWindowPos(ImVec2(543, 10), ImGuiCond_FirstUseEver);
        printEAInfo(ea);

        ImGui::PopStyleColor();

        generateGround(world, &groundVector, cars);

        ImGui::SFML::Render(w);

        w.display();

        // Attach camera to the car's body
        sf::View cameraView =
            sf::View(sf::Vector2f(getFurthestCarPos(cars).x * Config::PPM,
                                  Config::WINDOW_HEIGHT - getFurthestCarPos(cars).y * Config::PPM),
                     sf::Vector2f(Config::WINDOW_WIDTH, Config::WINDOW_HEIGHT));
        w.setView(cameraView);

        // If the camera moves, shift backgrounds accordingly to create a parallax effect
        for (int i = 0; i < 5; ++i) {
            sprites[i].setPosition(
                cameraView.getCenter().x * (1.0 - 0.2 * i) - Config::WINDOW_WIDTH * (1.4 - 0.1 * i),
                cameraView.getCenter().y - Config::WINDOW_HEIGHT / 2.0);
        }

        if (!paused) {
            for (auto& car : cars) {
                car.getFrontWheel()->body->ApplyTorque(-CarConfig::CAR_TORQUE, false);
                car.getBackWheel()->body->ApplyTorque(-CarConfig::CAR_TORQUE, false);
                applyAirResistance(car);
            }
        }

        // Display FPS in window title
        w.setTitle("EvoRacer, FPS: " + std::to_string((int)ImGui::GetIO().Framerate));

        handleEvents(w, pauseCheck, nextGCheck, focus);
        handleUserInput(w, paused, pauseCheck, nextGen, nextGCheck, focus);
    }

    ImPlot::DestroyContext();
    ImGui::SFML::Shutdown();

    return 0;
}
```

Here is the call graph for this function:



4.21.4 Variable Documentation

4.21.4.1 world

```
b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION))
```

Definition at line 28 of file Main.cc.

4.22 Main.cc

Go to the documentation of this file.

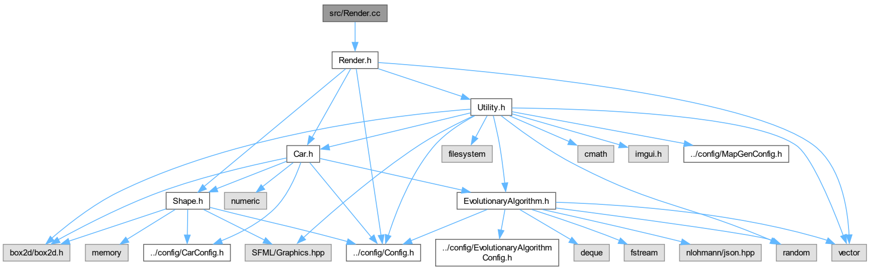
```cpp

#include "box2d/box2d.h"
#include "imgui.h"
#include "imgui-SFML.h"
#include "implot.h"
#include "SFML/Graphics.hpp"

#include "../config/Config.h"
#include "../config/MapGenConfig.h"
#include "Car.h"
#include "EvolutionaryAlgorithm.h"
#include "GUI.h"
#include "Render.h"
#include "Shape.h"
#include "UserInput.h"
#include "Utility.h"

typedef std::shared_ptr<b2World> b2WorldPtr;

// initialize the world as a shared pointer
b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));

int main() {
    sf::ContextSettings settings;
    settings.antialiasingLevel = 8;

    // Setup SFML window
    sf::RenderWindow w(sf::VideoMode(Config::WINDOW_WIDTH, Config::WINDOW_HEIGHT), "EvoRacer",
                       sf::Style::Default, settings);
    w.setFramerateLimit(Config::MAX_FPS);

    // Initialize ImGui and all its friends
    ImGui::SFML::Init(w);
    ImPlot::CreateContext();

    // Change imgui.ini location
    ImGui::GetIO().IniFilename = "./imgui.ini";

    // Containers to hold objects we create
    std::vector<Polygon> groundVector;
    std::vector<Car> cars;

    // Generate ground
    std::vector<b2Vec2> groundVertices = {b2Vec2(0, 0), b2Vec2(MapGenConfig::GROUND_PART_LENGTH, 0),
                                          b2Vec2(0, -MapGenConfig::GROUND_LEG_LENGTH)};
    Polygon ground =
        createGround(world, MapGenConfig::GROUND_STARTING_X, MapGenConfig::GROUND_STARTING_Y,
                     groundVertices, sf::Color(18, 36, 35));
    groundVector.push_back(ground);

    EvolutionaryAlgorithm ea(EvolutionaryAlgorithmConfig::POPULATION_SIZE, Config::SAVE_TO_FILE);

    for (const Chromosome& chromosome : ea.getPopulation()) {
        cars.push_back(generateCar(world, chromosome));
    }

    bool paused = false;     // Should we pause the simulation?
    bool pauseCheck = true;  // Should we check if the user wants to flip `paused'?
    bool nextGen = false;    // Should we generate the next generation?
    bool nextGCheck = true;  // Should we check if the user wants to flip `nextGen'?
    bool focus = true;       // Is the window in focus? (used to prevent input when not in focus)
    int timer = 0;

    // Set window icon
    setIcon(w);

    auto textures = loadBGTextures();
    auto sprites = loadBGSprites(textures, cars);

    sf::Clock deltaClock;
    while (w.isOpen()) {
        // Update the world, standard arguments
        if (!paused) {
            world->Step(1 / 60.0f, 6, 3);
            ++timer;
            if (timer >= Config::GENERATION_TIME) {
                nextGen = true;
            }
        }

        if (nextGen) {
            nextGen = false;
            for (int i = 0; i < cars.size(); ++i) {
                ea.setFitness(i, cars[i].getPosX());
            }
            ea.nextGeneration();
            removeCars(world, &cars);
            for (const Chromosome& chromosome : ea.getPopulation()) {
                cars.push_back(generateCar(world, chromosome));
            }
            timer = 0;
        }

        // Render everything
        render(w, sprites, groundVector, cars);

        ImGui::SFML::Update(w, deltaClock.restart());

        ImGui::PushStyleColor(ImGuiCol_WindowBg, ImVec4(0.071f, 0.141f, 0.137f, 0.5f));

        ImGui::SetNextWindowSize(ImVec2(340, 340), ImGuiCond_FirstUseEver);
        ImGui::SetNextWindowPos(ImVec2(10, 10), ImGuiCond_FirstUseEver);
        renderVelocityPlot(cars, paused);

        ImGui::SetNextWindowSize(ImVec2(340, 340), ImGuiCond_FirstUseEver);
        ImGui::SetNextWindowPos(ImVec2(930, 10), ImGuiCond_FirstUseEver);
        renderPositionPlot(cars, paused);

        ImGui::SetNextWindowSize(ImVec2(175, 75), ImGuiCond_FirstUseEver);
        ImGui::SetNextWindowPos(ImVec2(543, 10), ImGuiCond_FirstUseEver);
        printEAInfo(ea);

        ImGui::PopStyleColor();

        generateGround(world, &groundVector, cars);

        ImGui::SFML::Render(w);

        w.display();

        // Attach camera to the car’s body
        sf::View cameraView =
            sf::View(sf::Vector2f(getFurthestCarPos(cars).x * Config::PPM,
                                  Config::WINDOW_HEIGHT - getFurthestCarPos(cars).y * Config::PPM),
                     sf::Vector2f(Config::WINDOW_WIDTH, Config::WINDOW_HEIGHT));
        w.setView(cameraView);

        // If the camera moves, shift backgrounds accordingly to create a parallax effect
        for (int i = 0; i < 5; ++i) {
            sprites[i].setPosition(
                cameraView.getCenter().x * (1.0 - 0.2 * i) - Config::WINDOW_WIDTH * (1.4 - 0.1 * i),
                cameraView.getCenter().y - Config::WINDOW_HEIGHT / 2.0);
        }

        if (!paused) {
            for (auto& car : cars) {
                car.getFrontWheel()->body->ApplyTorque(-CarConfig::CAR_TORQUE, false);
                car.getBackWheel()->body->ApplyTorque(-CarConfig::CAR_TORQUE, false);
                applyAirResistance(car);
            }
        }

        // Display FPS in window title
        w.setTitle("EvoRacer, FPS: " + std::to_string((int)ImGui::GetIO().Framerate));

        handleEvents(w, pauseCheck, nextGCheck, focus);
        handleUserInput(w, paused, pauseCheck, nextGen, nextGCheck, focus);
    }

    ImPlot::DestroyContext();
    ImGui::SFML::Shutdown();

    return 0;
}
```

4.23 src/Render.cc File Reference

This file contains the render function, which is responsible for rendering all the shapes in the world.

`#include ¨Render.h¨`

Include dependency graph for Render.cc:



Functions

- void renderCircle (sf::RenderWindow &w, Circle ∗circle)

  Function for rendering a circle.
- void renderCircleDebug (sf::RenderWindow &w, Circle ∗circle)

  Function for rendering a circle's debug information.
- void renderPolygon (sf::RenderWindow &w, Polygon ∗polygon)

  Function for rendering a polygon.
- void renderPolygonDebug (sf::RenderWindow &w, Polygon ∗polygon)

  Function for rendering a polygon's debug information.
- void renderCar (sf::RenderWindow &w, Car car)

  Function for rendering a car.
- void render (sf::RenderWindow &w, const std::vector< sf::Sprite > &BGs, std::vector< Polygon > &groundVector, std::vector< Car > &cars)

  Main render function.

4.23.1 Detailed Description

This file contains the render function, which is responsible for rendering all the shapes in the world.

Authors

Jakub Marcowski, Mateusz Krakowski

Date

2023-06-06

Definition in file Render.cc.

4.23.2 Function Documentation

4.23.2.1 render()

```
void render (
            sf::RenderWindow & w,
            const std::vector< sf::Sprite > & BGs,
            std::vector< Polygon > & ground,
            std::vector< Car > & cars )
```

Main render function.

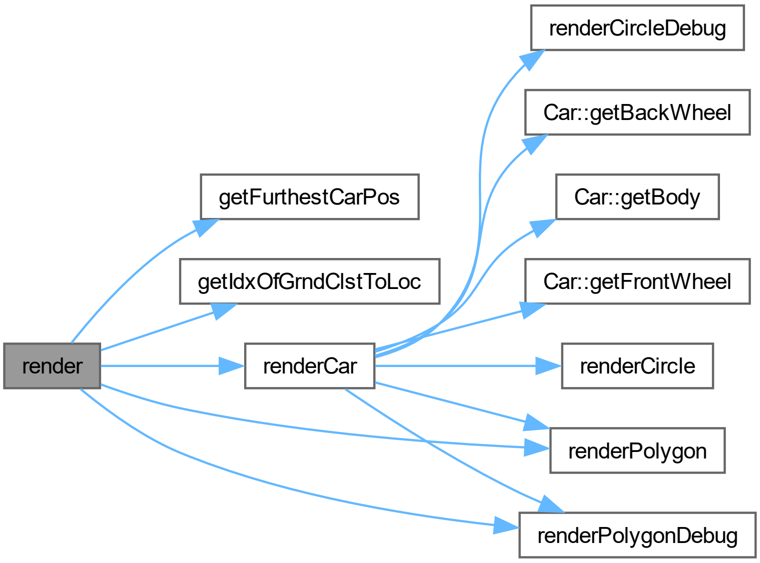
Parameters

w	SFML's RenderWindow.
BGs	Vector of background sprites.
ground	Vector of ground polygons.
cars	Vector of cars.

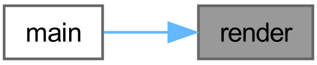
Definition at line 99 of file Render.cc.

```
                                                                    {
    w.clear();
    for (const sf::Sprite &BG : BGs) {
        w.draw(BG);
    }

    int groundBeginIndex = 0;
    int centerIndex = getIdxOfGrndClstToLoc(groundVector, getFurthestCarPos(cars).x);
    int groundEndIndex = groundVector.size();

    if (centerIndex - Config::GROUND_PARTS_RENDERED / 2 > 0) {
        groundBeginIndex = centerIndex - Config::GROUND_PARTS_RENDERED / 2;
    }
    if (centerIndex + Config::GROUND_PARTS_RENDERED / 2 < groundEndIndex) {
        groundEndIndex = centerIndex + Config::GROUND_PARTS_RENDERED / 2;
    }
    std::vector<Polygon> groundSlice(groundVector.begin() + groundBeginIndex,
                                     groundVector.begin() + groundEndIndex);

    for (Polygon ground : groundSlice) {
        renderPolygon(w, &ground);
        if (Config::DEBUG) {
            renderPolygonDebug(w, &ground);
        }
    }

    // new cars should be rendered behind the old ones
    for (int i = cars.size() - 1; i >= 0; --i) {
        renderCar(w, cars[i]);
    }
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.23.2.2 renderCar()

```
void renderCar (
             sf::RenderWindow & w,
             Car car )
```

Function for rendering a car.

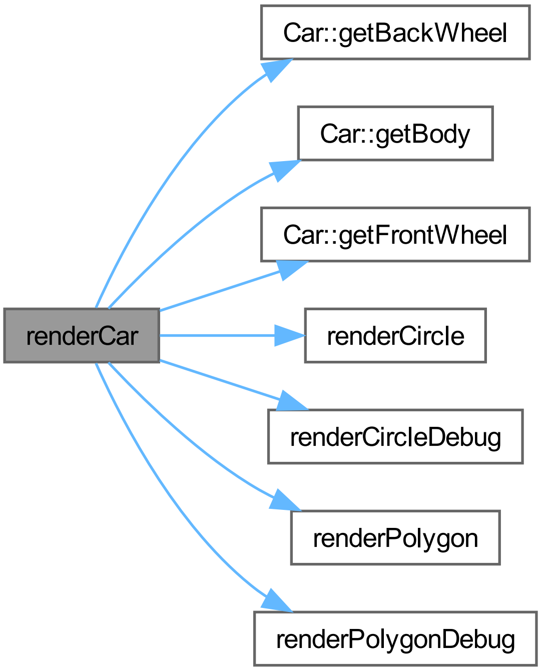
Parameters

w	SFML's RenderWindow.
car	Car to be rendered.

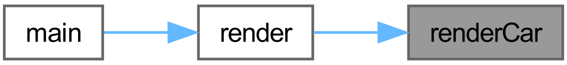
Definition at line 88 of file Render.cc.

```
00088                                              {
00089     renderPolygon(w, car.getBody());
00090     renderCircle(w, car.getFrontWheel());
00091     renderCircle(w, car.getBackWheel());
00092     if (Config::DEBUG) {
00093         renderPolygonDebug(w, car.getBody());
00094         renderCircleDebug(w, car.getFrontWheel());
00095         renderCircleDebug(w, car.getBackWheel());
00096     }
00097 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.23.2.3 renderCircle()

```
void renderCircle (
             sf::RenderWindow & w,
             Circle * circle )
```

Function for rendering a circle.

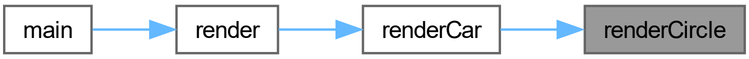
Parameters

w	SFML's RenderWindow.
circle	Pointer to the circle.

Definition at line 12 of file Render.cc.

```
00012                                                       {
00013     sf::CircleShape circ;
00014 
00015     circ.setPosition(circle->body->GetPosition().x * Config::PPM,
00016                      Config::WINDOW_HEIGHT - (circle->body->GetPosition().y * Config::PPM));
00017 
00018     circ.setOrigin(circle->radius, circle->radius);
00019 
00020     circ.setRadius(circle->radius);
00021 
00022     circ.setRotation(-1 * circle->body->GetAngle() * Config::DEG_PER_RAD);
00023 
00024     circ.setFillColor(circle->color);
00025     w.draw(circ);
00026 }
```

Here is the caller graph for this function:



4.23.2.4 renderCircleDebug()

```
void renderCircleDebug (
               sf::RenderWindow & w,
               Circle * circle )
```

Function for rendering a circle's debug information.

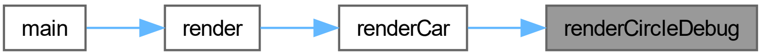
Parameters

w	SFML's RenderWindow.
circle	Pointer to the circle.

Definition at line 28 of file Render.cc.

```
00028                                                            {
00029     // Draw a line from the circle’s center to its edge
00030     // (account for rotation if the body has non-zero torque)
00031     sf::Vertex line[] = {
00032         sf::Vertex(
00033             sf::Vector2f(circle->body->GetPosition().x * Config::PPM,
00034                          Config::WINDOW_HEIGHT - (circle->body->GetPosition().y * Config::PPM))),
00035         sf::Vertex(sf::Vector2f(
00036             circle->body->GetPosition().x * Config::PPM +
00037                 circle->radius * cos(circle->body->GetAngle()),
00038             Config::WINDOW_HEIGHT - (circle->body->GetPosition().y * Config::PPM +
00039                                      circle->radius * sin(circle->body->GetAngle()))))};
00040     w.draw(line, 2, sf::Lines);
00041 }
```

Here is the caller graph for this function:



4.23.2.5 renderPolygon()

```
void renderPolygon (
            sf::RenderWindow & w,
            Polygon * polygon )
```

Function for rendering a polygon.

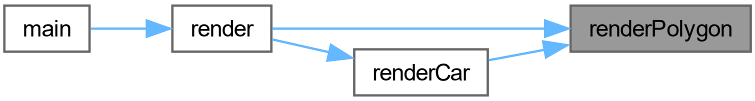
Parameters

w	SFML's RenderWindow.
polygon	Pointer to the polygon.

Definition at line 43 of file Render.cc.

```
00043                                                    {
00044     sf::ConvexShape convex;
00045 
00046     convex.setPosition(polygon->body->GetPosition().x * Config::PPM,
00047                        Config::WINDOW_HEIGHT - (polygon->body->GetPosition().y * Config::PPM));
00048 
00049     convex.setOrigin(0, 0);
00050 
00051     convex.setPointCount(polygon->vertices.size());
00052     for (int i = 0; i < polygon->vertices.size(); ++i) {
00053         convex.setPoint(i, sf::Vector2f(polygon->vertices[i].x * Config::PPM,
00054                                         polygon->vertices[i].y * Config::PPM));
00055     }
00056 
00057     convex.setRotation(-1 * polygon->body->GetAngle() * Config::DEG_PER_RAD);
00058 
00059     // Flip the polygon along the X axis
00060     convex.scale(1, -1);
00061 
00062     convex.setFillColor(polygon->color);
00063     w.draw(convex);
00064 }
```

Here is the caller graph for this function:



4.23.2.6 renderPolygonDebug()

```
void renderPolygonDebug (
              sf::RenderWindow & w,
              Polygon * polygon )
```

Function for rendering a polygon's debug information.

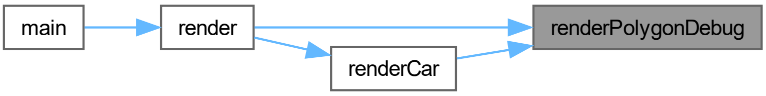
Parameters

w	SFML's RenderWindow.
polygon	Pointer to the polygon.

Definition at line 66 of file Render.cc.

```
00066                                                                   {
00067     // Draw the polygon's center
00068     sf::CircleShape circ;
00069     circ.setRadius(5);
00070     circ.setOrigin(5, 5);
00071     circ.setPosition(polygon->body->GetPosition().x * Config::PPM,
00072                      Config::WINDOW_HEIGHT - (polygon->body->GetPosition().y * Config::PPM));
00073     circ.setFillColor(sf::Color::Blue);
00074     w.draw(circ);
00075 
00076     // Draw the polygon's vertices
00077     for (int i = 0; i < polygon->vertices.size(); ++i) {
00078         circ.setRadius(2);
00079         circ.setOrigin(2, 2);
00080         circ.setPosition(polygon->body->GetWorldPoint(polygon->vertices[i]).x * Config::PPM,
00081                          Config::WINDOW_HEIGHT -
00082                              (polygon->body->GetWorldPoint(polygon->vertices[i]).y * Config::PPM));
00083         circ.setFillColor(sf::Color::White);
00084         w.draw(circ);
00085     }
00086 }
```

Here is the caller graph for this function:



4.24 Render.cc

Go to the documentation of this file.

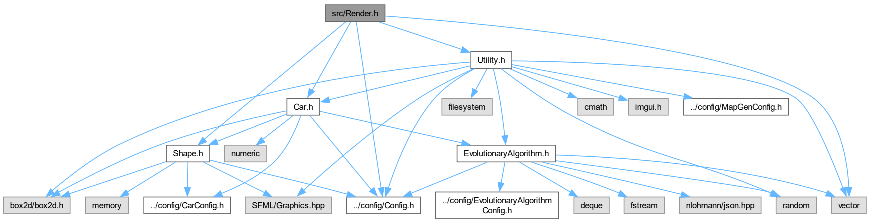
```

#include "Render.h"

void renderCircle(sf::RenderWindow &w, Circle *circle) {
    sf::CircleShape circ;

    circ.setPosition(circle->body->GetPosition().x * Config::PPM,
                     Config::WINDOW_HEIGHT - (circle->body->GetPosition().y * Config::PPM));

    circ.setOrigin(circle->radius, circle->radius);

    circ.setRadius(circle->radius);

    circ.setRotation(-1 * circle->body->GetAngle() * Config::DEG_PER_RAD);
    circ.setFillColor(circle->color);
    w.draw(circ);
}

void renderCircleDebug(sf::RenderWindow &w, Circle *circle) {
    // Draw a line from the circle's center to its edge
    // (account for rotation if the body has non-zero torque)
    sf::Vertex line[] = {
        sf::Vertex(
            sf::Vector2f(circle->body->GetPosition().x * Config::PPM,
                         Config::WINDOW_HEIGHT - (circle->body->GetPosition().y * Config::PPM))),
        sf::Vertex(sf::Vector2f(
            circle->body->GetPosition().x * Config::PPM +
                circle->radius * cos(circle->body->GetAngle()),
            Config::WINDOW_HEIGHT - (circle->body->GetPosition().y * Config::PPM +
                                     circle->radius * sin(circle->body->GetAngle()))))};
    w.draw(line, 2, sf::Lines);
}

void renderPolygon(sf::RenderWindow &w, Polygon *polygon) {
    sf::ConvexShape convex;

    convex.setPosition(polygon->body->GetPosition().x * Config::PPM,
                       Config::WINDOW_HEIGHT - (polygon->body->GetPosition().y * Config::PPM));

    convex.setOrigin(0, 0);

    convex.setPointCount(polygon->vertices.size());
    for (int i = 0; i < polygon->vertices.size(); ++i) {
        convex.setPoint(i, sf::Vector2f(polygon->vertices[i].x * Config::PPM,
                                        polygon->vertices[i].y * Config::PPM));
    }

    convex.setRotation(-1 * polygon->body->GetAngle() * Config::DEG_PER_RAD);

    // Flip the polygon along the X axis
    convex.scale(1, -1);

    convex.setFillColor(polygon->color);
    w.draw(convex);
}

void renderPolygonDebug(sf::RenderWindow &w, Polygon *polygon) {
    // Draw the polygon's center
    sf::CircleShape circ;
    circ.setRadius(5);
    circ.setOrigin(5, 5);
    circ.setPosition(polygon->body->GetPosition().x * Config::PPM,
                     Config::WINDOW_HEIGHT - (polygon->body->GetPosition().y * Config::PPM));
    circ.setFillColor(sf::Color::Blue);
    w.draw(circ);

    // Draw the polygon's vertices
    for (int i = 0; i < polygon->vertices.size(); ++i) {
        circ.setRadius(2);
        circ.setOrigin(2, 2);
        circ.setPosition(polygon->body->GetWorldPoint(polygon->vertices[i]).x * Config::PPM,
                         Config::WINDOW_HEIGHT -
                             (polygon->body->GetWorldPoint(polygon->vertices[i]).y * Config::PPM));
        circ.setFillColor(sf::Color::White);
        w.draw(circ);
    }
}

void renderCar(sf::RenderWindow &w, Car car) {
    renderPolygon(w, car.getBody());
    renderCircle(w, car.getFrontWheel());
    renderCircle(w, car.getBackWheel());
    if (Config::DEBUG) {
        renderPolygonDebug(w, car.getBody());
        renderCircleDebug(w, car.getFrontWheel());
        renderCircleDebug(w, car.getBackWheel());
    }
}

void render(sf::RenderWindow &w, const std::vector<sf::Sprite> &BGs,
            std::vector<Polygon> &groundVector, std::vector<Car> &cars) {
    w.clear();
    for (const sf::Sprite &BG : BGs) {
        w.draw(BG);
    }

    int groundBeginIndex = 0;
    int centerIndex = getIdxOfGrndClstToLoc(groundVector, getFurthestCarPos(cars).x);
    int groundEndIndex = groundVector.size();

    if (centerIndex - Config::GROUND_PARTS_RENDERED / 2 > 0) {
        groundBeginIndex = centerIndex - Config::GROUND_PARTS_RENDERED / 2;
    }
    if (centerIndex + Config::GROUND_PARTS_RENDERED / 2 < groundEndIndex) {
        groundEndIndex = centerIndex + Config::GROUND_PARTS_RENDERED / 2;
    }
    std::vector<Polygon> groundSlice(groundVector.begin() + groundBeginIndex,
                                     groundVector.begin() + groundEndIndex);

    for (Polygon ground : groundSlice) {
        renderPolygon(w, &ground);
        if (Config::DEBUG) {
            renderPolygonDebug(w, &ground);
        }
    }

    // new cars should be rendered behind the old ones
    for (int i = cars.size() - 1; i >= 0; --i) {
        renderCar(w, cars[i]);
    }
}
```

4.25 src/Render.h File Reference

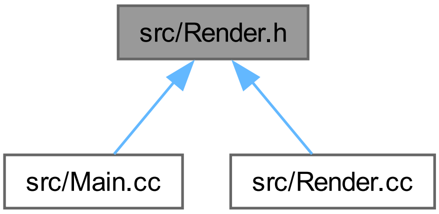
Header file for render function.

```
#include <vector>
#include ¨../config/Config.h¨
#include ¨Car.h¨
#include ¨Shape.h¨
#include ¨Utility.h¨
```

Include dependency graph for Render.h:



This graph shows which files directly or indirectly include this file:



Functions

- void renderCircle (sf::RenderWindow &w, Circle ∗circle)

  Function for rendering a circle.
- void renderCircleDebug (sf::RenderWindow &w, Circle ∗circle)

  Function for rendering a circle's debug information.
- void renderPolygon (sf::RenderWindow &w, Polygon ∗polygon)

  Function for rendering a polygon.
- void renderPolygonDebug (sf::RenderWindow &w, Polygon ∗polygon)

  Function for rendering a polygon's debug information.
- void renderCar (sf::RenderWindow &w, Car car)

  Function for rendering a car.
- void render (sf::RenderWindow &w, const std::vector< sf::Sprite > &BGs, std::vector< Polygon > &ground, std::vector< Car > &cars)

  Main render function.

4.25.1 Detailed Description

Header file for render function.

Authors

Jakub Marcowski, Mateusz Krakowski

Date

2023-06-06

Definition in file Render.h.

4.25.2 Function Documentation

4.25.2.1 render()

```
void render (
            sf::RenderWindow & w,
            const std::vector< sf::Sprite > & BGs,
            std::vector< Polygon > & ground,
            std::vector< Car > & cars )
```

Main render function.

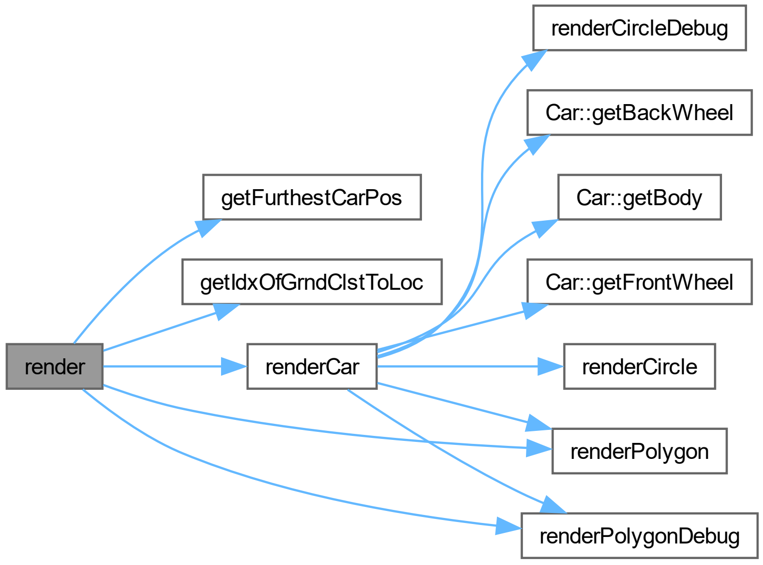
Parameters

w	SFML's RenderWindow.
BGs	Vector of background sprites.
ground	Vector of ground polygons.
cars	Vector of cars.

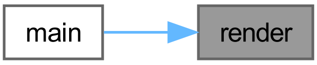
Definition at line 99 of file Render.cc.

```
                                                                      {
    w.clear();
    for (const sf::Sprite &BG : BGs) {
        w.draw(BG);
    }

    int groundBeginIndex = 0;
    int centerIndex = getIdxOfGrndClstToLoc(groundVector, getFurthestCarPos(cars).x);
    int groundEndIndex = groundVector.size();

    if (centerIndex - Config::GROUND_PARTS_RENDERED / 2 > 0) {
        groundBeginIndex = centerIndex - Config::GROUND_PARTS_RENDERED / 2;
    }
    if (centerIndex + Config::GROUND_PARTS_RENDERED / 2 < groundEndIndex) {
        groundEndIndex = centerIndex + Config::GROUND_PARTS_RENDERED / 2;
    }
    std::vector<Polygon> groundSlice(groundVector.begin() + groundBeginIndex,
                                     groundVector.begin() + groundEndIndex);

    for (Polygon ground : groundSlice) {
        renderPolygon(w, &ground);
        if (Config::DEBUG) {
            renderPolygonDebug(w, &ground);
        }
    }

    // new cars should be rendered behind the old ones
    for (int i = cars.size() - 1; i >= 0; --i) {
        renderCar(w, cars[i]);
    }
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.25.2.2 renderCar()

```
void renderCar (
            sf::RenderWindow & w,
            Car car )
```

Function for rendering a car.

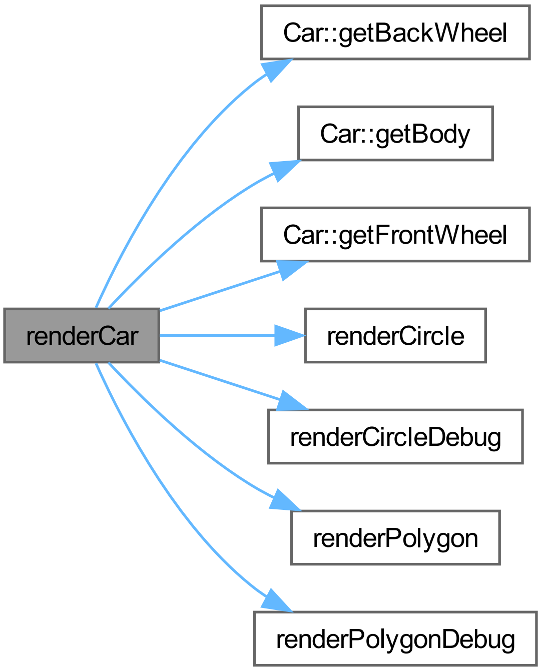
Parameters

w	SFML's RenderWindow.
car	Car to be rendered.

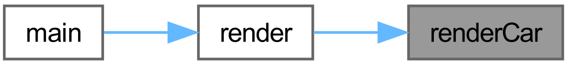
Definition at line 88 of file Render.cc.

```
                                              {
    renderPolygon(w, car.getBody());
    renderCircle(w, car.getFrontWheel());
    renderCircle(w, car.getBackWheel());
    if (Config::DEBUG) {
        renderPolygonDebug(w, car.getBody());
        renderCircleDebug(w, car.getFrontWheel());
        renderCircleDebug(w, car.getBackWheel());
    }
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.25.2.3 renderCircle()

```
void renderCircle (
            sf::RenderWindow & w,
            Circle * circle )
```

Function for rendering a circle.

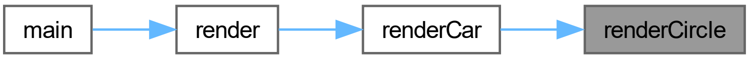
Parameters

w	SFML's RenderWindow.
circle	Pointer to the circle.

Definition at line 12 of file Render.cc.

```
                                                     {
    sf::CircleShape circ;

    circ.setPosition(circle->body->GetPosition().x * Config::PPM,
                     Config::WINDOW_HEIGHT - (circle->body->GetPosition().y * Config::PPM));

    circ.setOrigin(circle->radius, circle->radius);

    circ.setRadius(circle->radius);

    circ.setRotation(-1 * circle->body->GetAngle() * Config::DEG_PER_RAD);

    circ.setFillColor(circle->color);
    w.draw(circ);
}
```

Here is the caller graph for this function:



4.25.2.4 renderCircleDebug()

```
void renderCircleDebug (
            sf::RenderWindow & w,
            Circle * circle )
```

Function for rendering a circle's debug information.

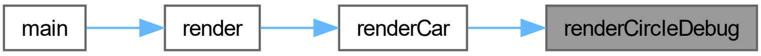
Parameters

w	SFML's RenderWindow.
circle	Pointer to the circle.

Definition at line 28 of file Render.cc.

```
                                                          {
    // Draw a line from the circle's center to its edge
    // (account for rotation if the body has non-zero torque)
    sf::Vertex line[] = {
        sf::Vertex(
            sf::Vector2f(circle->body->GetPosition().x * Config::PPM,
                         Config::WINDOW_HEIGHT - (circle->body->GetPosition().y * Config::PPM))),
        sf::Vertex(sf::Vector2f(
            circle->body->GetPosition().x * Config::PPM +
                circle->radius * cos(circle->body->GetAngle()),
            Config::WINDOW_HEIGHT - (circle->body->GetPosition().y * Config::PPM +
                                     circle->radius * sin(circle->body->GetAngle()))))};
    w.draw(line, 2, sf::Lines);
}
```

Here is the caller graph for this function:



4.25.2.5 renderPolygon()

```
void renderPolygon (
            sf::RenderWindow & w,
            Polygon * polygon )
```

Function for rendering a polygon.

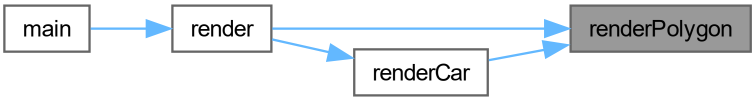
Parameters

w	SFML's RenderWindow.
polygon	Pointer to the polygon.

Definition at line 43 of file Render.cc.

```
00043                                                      {
00044     sf::ConvexShape convex;
00045 
00046     convex.setPosition(polygon->body->GetPosition().x * Config::PPM,
00047                        Config::WINDOW_HEIGHT - (polygon->body->GetPosition().y * Config::PPM));
00048 
00049     convex.setOrigin(0, 0);
00050 
00051     convex.setPointCount(polygon->vertices.size());
00052     for (int i = 0; i < polygon->vertices.size(); ++i) {
00053         convex.setPoint(i, sf::Vector2f(polygon->vertices[i].x * Config::PPM,
00054                                         polygon->vertices[i].y * Config::PPM));
00055     }
00056 
00057     convex.setRotation(-1 * polygon->body->GetAngle() * Config::DEG_PER_RAD);
00058 
00059     // Flip the polygon along the X axis
00060     convex.scale(1, -1);
00061 
00062     convex.setFillColor(polygon->color);
00063     w.draw(convex);
00064 }
```

Here is the caller graph for this function:



4.25.2.6 renderPolygonDebug()

```
void renderPolygonDebug (
              sf::RenderWindow & w,
              Polygon * polygon )
```

Function for rendering a polygon's debug information.

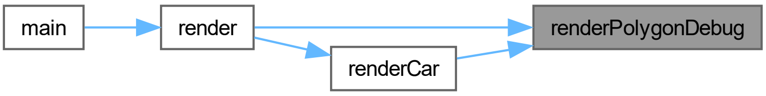
Parameters

w	SFML's RenderWindow.
polygon	Pointer to the polygon.

Definition at line 66 of file Render.cc.

```
00066                                                                  {
00067     // Draw the polygon's center
00068     sf::CircleShape circ;
00069     circ.setRadius(5);
00070     circ.setOrigin(5, 5);
00071     circ.setPosition(polygon->body->GetPosition().x * Config::PPM,
00072                      Config::WINDOW_HEIGHT - (polygon->body->GetPosition().y * Config::PPM));
00073     circ.setFillColor(sf::Color::Blue);
00074     w.draw(circ);
00075 
00076     // Draw the polygon's vertices
00077     for (int i = 0; i < polygon->vertices.size(); ++i) {
00078         circ.setRadius(2);
00079         circ.setOrigin(2, 2);
00080         circ.setPosition(polygon->body->GetWorldPoint(polygon->vertices[i]).x * Config::PPM,
00081                          Config::WINDOW_HEIGHT -
00082                              (polygon->body->GetWorldPoint(polygon->vertices[i]).y * Config::PPM));
00083         circ.setFillColor(sf::Color::White);
00084         w.draw(circ);
00085     }
00086 }
```

Here is the caller graph for this function:



4.26 Render.h

Go to the documentation of this file.

```
00001 
00009 #ifndef RENDER_H
00010 #define RENDER_H
00011 
00012 #include <vector>
00013 
00014 #include "../config/Config.h"
00015 #include "Car.h"
00016 #include "Shape.h"
00017 #include "Utility.h"
00018 
00025 void renderCircle(sf::RenderWindow &w, Circle *circle);
00026 
00033 void renderCircleDebug(sf::RenderWindow &w, Circle *circle);
```

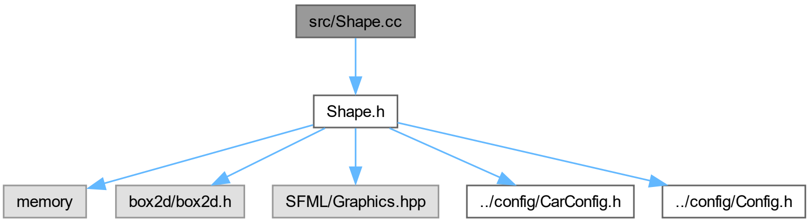
```

void renderPolygon(sf::RenderWindow &w, Polygon *polygon);

void renderPolygonDebug(sf::RenderWindow &w, Polygon *polygon);

void renderCar(sf::RenderWindow &w, Car car);

void render(sf::RenderWindow &w, const std::vector<sf::Sprite> &BGs, std::vector<Polygon> &ground,
            std::vector<Car> &cars);

#endif
```

4.27 src/Shape.cc File Reference

This file contains functions for creating Box2D objects.

```
#include ¨Shape.h¨
```

Include dependency graph for Shape.cc:



Functions

- Box createBox (const b2WorldPtr &world, float x, float y, float width, float height, float density, float friction, sf::Color color)

  Creates a box.
- Polygon createGround (const b2WorldPtr &world, float x, float y, const std::vector< b2Vec2 > &vertices, sf::Color color)

  Create a Ground object.
- Circle createCircle (const b2WorldPtr &world, float x, float y, float radius, float density, float friction, sf::Color color)

  Create a Circle object.
- Polygon createPolygon (const b2WorldPtr &world, float x, float y, std::vector< b2Vec2 > vertices, float density, float friction, sf::Color color)

  Create a Polygon object.

4.27.1 Detailed Description

This file contains functions for creating Box2D objects.

Authors

Jakub Marcowski, Mateusz Krakowski

Date

2023-06-06

Definition in file Shape.cc.

4.27.2 Function Documentation

4.27.2.1 createBox()

```
Box createBox (
            const b2WorldPtr & world,
            float x,
            float y,
            float width,
            float height,
            float density,
            float friction,
            sf::Color color )
```

Creates a box.

Parameters

world	2dWorld.
x	X coordinate of the box.
y	Y coordinate of the box.
width	Width of the box.
height	Height of the box.
density	Density of the box.
friction	Friction of the box.
color	Color of the box.

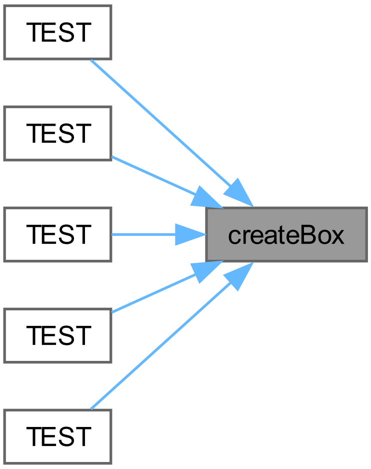
Returns

Box

Definition at line 11 of file Shape.cc.

```
                                            {
    // Argument validation
    if (width <= 0) {
        throw std::invalid_argument("Invalid width parameter");
    } else if (height <= 0.0f) {
        throw std::invalid_argument("Invalid height parameter");
    } else if (density <= 0.0f) {
        throw std::invalid_argument("Invalid density parameter");
    } else if (friction <= 0.0f) {
        throw std::invalid_argument("Invalid friction parameter");
    }
    // Body definition
    b2BodyDef boxBodyDef;
    boxBodyDef.position.Set(x / Config::PPM, y / Config::PPM);
    boxBodyDef.type = b2_dynamicBody;

    // Shape definition
    b2PolygonShape boxShape;
    boxShape.SetAsBox(width / 2 / Config::PPM, height / 2 / Config::PPM);

    // Fixture definition
    b2FixtureDef fixtureDef;
    fixtureDef.density = density;
    fixtureDef.friction = friction;
    fixtureDef.shape = &boxShape;

    // Now we have a body for our Box object
    b2Body* boxBody = world->CreateBody(&boxBodyDef);
    // Lastly, assign the fixture
    boxBody->CreateFixture(&fixtureDef);

    return Box{width, height, color, boxBody};
}
```

Here is the caller graph for this function:



4.27.2.2 createCircle()

```
Circle createCircle (
             const b2WorldPtr & world,
             float x,
             float y,
             float radius,
             float density,
             float friction,
             sf::Color color )
```

Create a Circle object.

Parameters

world	2dWorld
x	X coordinate of the circle.
y	Y coordinate of the circle.
radius	radius of the circle.
density	density of the circle.
friction	friction of the circle.
color	color of the circle.

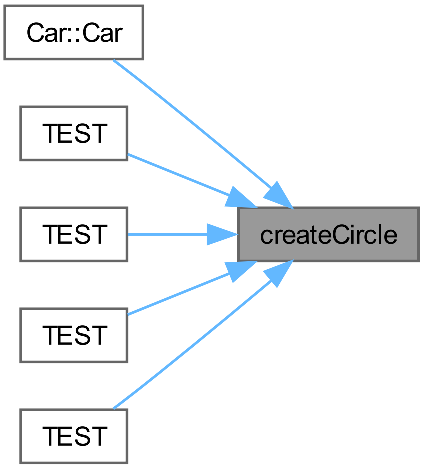
Returns

Circle

Definition at line 65 of file Shape.cc.

```
                                                   {
    // Argument validation
    if (radius <= 0.0f) {
        throw std::invalid_argument("Invalid width parameter");
    } else if (density <= 0.0f) {
        throw std::invalid_argument("Invalid density parameter");
    } else if (friction <= 0.0f) {
        throw std::invalid_argument("Invalid friction parameter");
    }

    b2BodyDef boxBodyDef;
    boxBodyDef.position.Set(x / Config::PPM, y / Config::PPM);
    boxBodyDef.type = b2_dynamicBody;

    b2CircleShape circleShape;
    circleShape.m_radius = radius / Config::PPM;

    b2FixtureDef fixtureDef;
    fixtureDef.density = density;
    fixtureDef.friction = friction;
    fixtureDef.shape = &circleShape;

    b2Body* boxBody = world->CreateBody(&boxBodyDef);

    boxBody->CreateFixture(&fixtureDef);

    return Circle{radius, color, boxBody};
}
```

Here is the caller graph for this function:



4.27.2.3 createGround()

```
Polygon createGround (
              const b2WorldPtr & world,
              float x,
              float y,
              const std::vector< b2Vec2 > & vertices,
              sf::Color color )
```

Create a Ground object.

Parameters

world	2dWorld.
x	X coordinate of the polygon.
y	Y coordinate of the polygon.
vertices	vectors that make up the ground
color	color of the polygon.

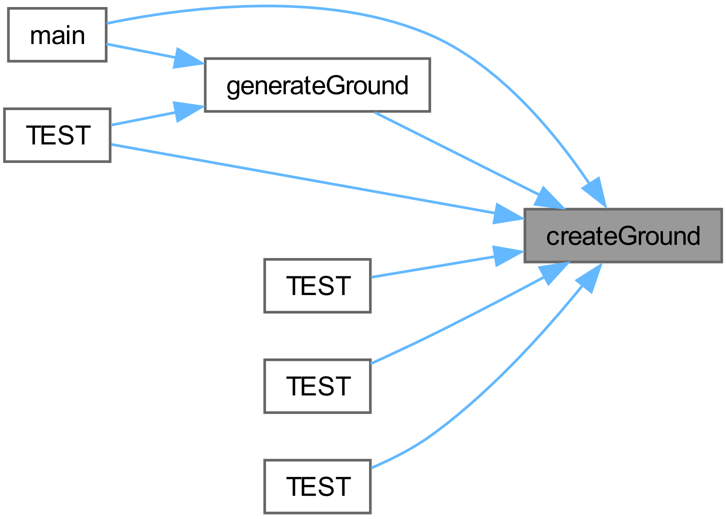
Returns

Polygon

Definition at line 46 of file Shape.cc.

```
00047                                {
00048     // Argument validation
00049     if (vertices.size() < 3) {
00050         throw std::invalid_argument("Invalid number of vertices");
00051     }
00052 
00053     b2BodyDef groundBodyDef;
00054     groundBodyDef.position.Set(x / Config::PPM, y / Config::PPM);
00055 
00056     b2PolygonShape groundPolygon;
00057     groundPolygon.Set(vertices.data(), static_cast<int32>(vertices.size()));
00058 
00059     b2Body* groundBody = world->CreateBody(&groundBodyDef);
00060     groundBody->CreateFixture(&groundPolygon, 0.0f);
00061 
00062     return Polygon{vertices, color, groundBody};
00063 }
```

Here is the caller graph for this function:



4.27.2.4 createPolygon()

```
Polygon createPolygon (
            const b2WorldPtr & world,
            float x,
            float y,
            std::vector< b2Vec2 > vertices,
            float density,
            float friction,
            sf::Color color )
```

Create a Polygon object.

Parameters

world	2dWorld.
x	X coordinate of the polygon.
y	Y coordinate of the polygon.
vertices	vectorst that make up the polygon.
density	density of the polygon.
friction	friction of the polygon.
color	color of the polygon.

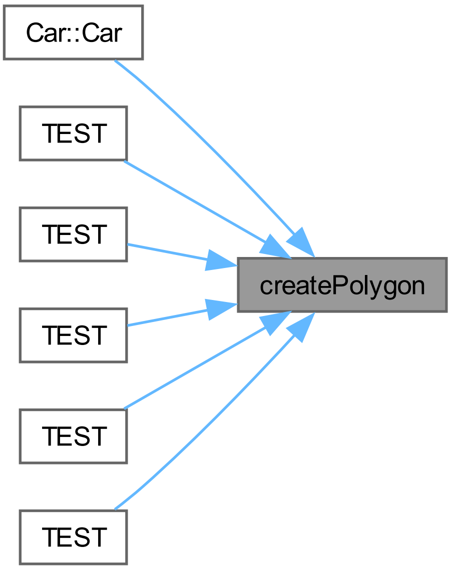
Returns

Polygon

Definition at line 95 of file Shape.cc.

```
                                                                  {
    // Argument validation
    if (vertices.size() < 3 || vertices.size() > CarConfig::CAR_VERTICES) {
        throw std::invalid_argument("Invalid vertices size");
    } else if (density <= 0.0f) {
        throw std::invalid_argument("Invalid density parameter");
    } else if (friction <= 0.0f) {
        throw std::invalid_argument("Invalid friction parameter");
    }
    b2BodyDef boxBodyDef;
    boxBodyDef.position.Set(x / Config::PPM, y / Config::PPM);
    boxBodyDef.type = b2_dynamicBody;

    b2PolygonShape boxShape;
    boxShape.Set(vertices.data(), vertices.size());

    b2FixtureDef fixtureDef;
    fixtureDef.density = density;
    fixtureDef.friction = friction;
    fixtureDef.shape = &boxShape;

    b2Body* boxBody = world->CreateBody(&boxBodyDef);

    boxBody->CreateFixture(&fixtureDef);

    // create a Polygon object with a shared pointer to the b2Body
    return Polygon{vertices, color, boxBody};
}
```

Here is the caller graph for this function:



4.28 Shape.cc

Go to the documentation of this file.

```

#include "Shape.h"

Box createBox(const b2WorldPtr& world, float x, float y, float width, float height, float density,
              float friction, sf::Color color) {
    // Argument validation
    if (width <= 0) {
        throw std::invalid_argument("Invalid width parameter");
    } else if (height <= 0.0f) {
        throw std::invalid_argument("Invalid height parameter");
    } else if (density <= 0.0f) {
        throw std::invalid_argument("Invalid density parameter");
    } else if (friction <= 0.0f) {
        throw std::invalid_argument("Invalid friction parameter");
    }
    // Body definition
    b2BodyDef boxBodyDef;
    boxBodyDef.position.Set(x / Config::PPM, y / Config::PPM);
    boxBodyDef.type = b2_dynamicBody;

    // Shape definition
    b2PolygonShape boxShape;
    boxShape.SetAsBox(width / 2 / Config::PPM, height / 2 / Config::PPM);

    // Fixture definition
    b2FixtureDef fixtureDef;
    fixtureDef.density = density;
    fixtureDef.friction = friction;
    fixtureDef.shape = &boxShape;

    // Now we have a body for our Box object
    b2Body* boxBody = world->CreateBody(&boxBodyDef);
    // Lastly, assign the fixture
    boxBody->CreateFixture(&fixtureDef);

    return Box{width, height, color, boxBody};
}
```

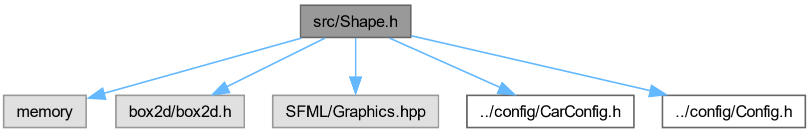
```

Polygon createGround(const b2WorldPtr& world, float x, float y, const std::vector<b2Vec2>& vertices,
                     sf::Color color) {
    // Argument validation
    if (vertices.size() < 3) {
        throw std::invalid_argument("Invalid number of vertices");
    }

    b2BodyDef groundBodyDef;
    groundBodyDef.position.Set(x / Config::PPM, y / Config::PPM);

    b2PolygonShape groundPolygon;
    groundPolygon.Set(vertices.data(), static_cast<int32>(vertices.size()));

    b2Body* groundBody = world->CreateBody(&groundBodyDef);
    groundBody->CreateFixture(&groundPolygon, 0.0f);

    return Polygon{vertices, color, groundBody};
}

Circle createCircle(const b2WorldPtr& world, float x, float y, float radius, float density,
                    float friction, sf::Color color) {
    // Argument validation
    if (radius <= 0.0f) {
        throw std::invalid_argument("Invalid width parameter");
    } else if (density <= 0.0f) {
        throw std::invalid_argument("Invalid density parameter");
    } else if (friction <= 0.0f) {
        throw std::invalid_argument("Invalid friction parameter");
    }

    b2BodyDef boxBodyDef;
    boxBodyDef.position.Set(x / Config::PPM, y / Config::PPM);
    boxBodyDef.type = b2_dynamicBody;

    b2CircleShape circleShape;
    circleShape.m_radius = radius / Config::PPM;

    b2FixtureDef fixtureDef;
    fixtureDef.density = density;
    fixtureDef.friction = friction;
    fixtureDef.shape = &circleShape;

    b2Body* boxBody = world->CreateBody(&boxBodyDef);

    boxBody->CreateFixture(&fixtureDef);

    return Circle{radius, color, boxBody};
}

Polygon createPolygon(const b2WorldPtr& world, float x, float y, std::vector<b2Vec2> vertices,
                      float density, float friction, sf::Color color) {
    // Argument validation
    if (vertices.size() < 3 || vertices.size() > CarConfig::CAR_VERTICES) {
        throw std::invalid_argument("Invalid vertices size");
    } else if (density <= 0.0f) {
        throw std::invalid_argument("Invalid density parameter");
    } else if (friction <= 0.0f) {
        throw std::invalid_argument("Invalid friction parameter");
    }
    b2BodyDef boxBodyDef;
    boxBodyDef.position.Set(x / Config::PPM, y / Config::PPM);
    boxBodyDef.type = b2_dynamicBody;

    b2PolygonShape boxShape;
    boxShape.Set(vertices.data(), vertices.size());

    b2FixtureDef fixtureDef;
    fixtureDef.density = density;
    fixtureDef.friction = friction;
    fixtureDef.shape = &boxShape;

    b2Body* boxBody = world->CreateBody(&boxBodyDef);

    boxBody->CreateFixture(&fixtureDef);

    // create a Polygon object with a shared pointer to the b2Body
    return Polygon{vertices, color, boxBody};
}
```

4.29 src/Shape.h File Reference

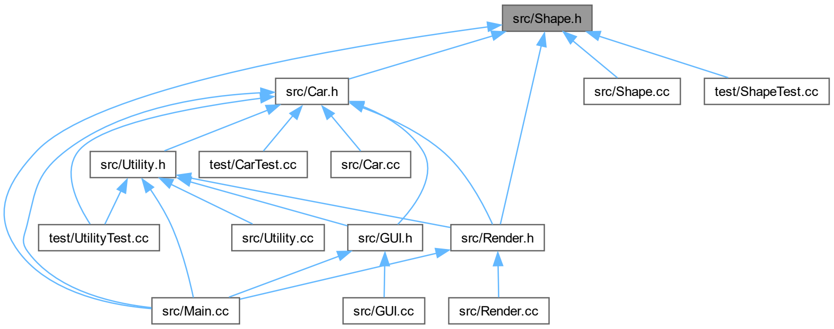
Header file for functions for creating Box2D objects.

```
#include <memory>
#include ¨box2d/box2d.h¨
#include ¨SFML/Graphics.hpp¨
#include ¨../config/CarConfig.h¨
#include ¨../config/Config.h¨
```

Include dependency graph for Shape.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct Box

  Struct representing a box.
- struct Circle

  Struct representing a circle.
- struct Polygon

  Struct representing a polygon.

Typedefs

- typedef std::shared_ptr< b2World > b2WorldPtr

Functions

- Box createBox (const b2WorldPtr &world, float x, float y, float width, float height, float density, float friction, sf::Color color)

  Creates a box.
- Polygon createGround (const b2WorldPtr &world, float x, float y, const std::vector< b2Vec2 > &vertices, sf::Color color)

  Create a Ground object.
- Circle createCircle (const b2WorldPtr &world, float x, float y, float radius, float density, float friction, sf::Color color)

  Create a Circle object.
- Polygon createPolygon (const b2WorldPtr &world, float x, float y, std::vector< b2Vec2 > vertices, float density, float friction, sf::Color color)

  Create a Polygon object.

4.29.1 Detailed Description

Header file for functions for creating Box2D objects.

Authors

Jakub Marcowski, Mateusz Krakowski

Date

2023-06-06

Definition in file Shape.h.

4.29.2 Typedef Documentation

4.29.2.1 b2WorldPtr

```
typedef std::shared_ptr<b2World> b2WorldPtr
```

Definition at line 48 of file Shape.h.

4.29.3 Function Documentation

4.29.3.1 createBox()

```
Box createBox (
             const b2WorldPtr & world,
             float x,
             float y,
             float width,
             float height,
             float density,
             float friction,
             sf::Color color )
```

Creates a box.

Parameters

world	2dWorld.
x	X coordinate of the box.
y	Y coordinate of the box.
width	Width of the box.
height	Height of the box.
density	Density of the box.
friction	Friction of the box.
color	Color of the box.

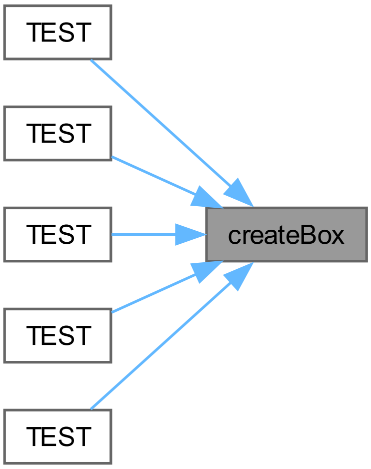
Returns

Box

Definition at line 11 of file Shape.cc.

```
                                                  {
    // Argument validation
    if (width <= 0) {
        throw std::invalid_argument("Invalid width parameter");
    } else if (height <= 0.0f) {
        throw std::invalid_argument("Invalid height parameter");
    } else if (density <= 0.0f) {
        throw std::invalid_argument("Invalid density parameter");
    } else if (friction <= 0.0f) {
        throw std::invalid_argument("Invalid friction parameter");
    }
    // Body definition
    b2BodyDef boxBodyDef;
    boxBodyDef.position.Set(x / Config::PPM, y / Config::PPM);
    boxBodyDef.type = b2_dynamicBody;

    // Shape definition
    b2PolygonShape boxShape;
    boxShape.SetAsBox(width / 2 / Config::PPM, height / 2 / Config::PPM);

    // Fixture definition
    b2FixtureDef fixtureDef;
    fixtureDef.density = density;
    fixtureDef.friction = friction;
    fixtureDef.shape = &boxShape;

    // Now we have a body for our Box object
    b2Body* boxBody = world->CreateBody(&boxBodyDef);
    // Lastly, assign the fixture
    boxBody->CreateFixture(&fixtureDef);

    return Box{width, height, color, boxBody};
}
```

Here is the caller graph for this function:



4.29.3.2 createCircle()

```
Circle createCircle (
             const b2WorldPtr & world,
             float x,
             float y,
             float radius,
             float density,
             float friction,
             sf::Color color )
```

Create a Circle object.

Parameters

world	2dWorld
x	X coordinate of the circle.
y	Y coordinate of the circle.
radius	radius of the circle.
density	density of the circle.
friction	friction of the circle.
color	color of the circle.

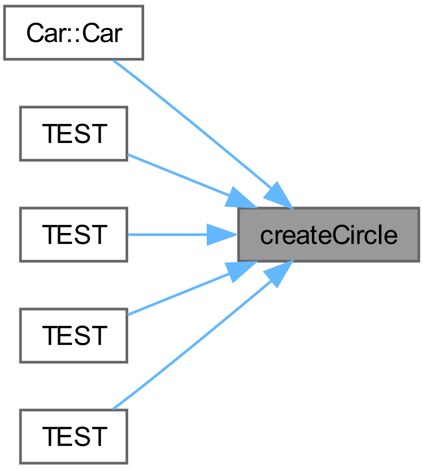
Returns

Circle

Definition at line 65 of file Shape.cc.

```
                                                          {
    // Argument validation
    if (radius <= 0.0f) {
        throw std::invalid_argument("Invalid width parameter");
    } else if (density <= 0.0f) {
        throw std::invalid_argument("Invalid density parameter");
    } else if (friction <= 0.0f) {
        throw std::invalid_argument("Invalid friction parameter");
    }

    b2BodyDef boxBodyDef;
    boxBodyDef.position.Set(x / Config::PPM, y / Config::PPM);
    boxBodyDef.type = b2_dynamicBody;

    b2CircleShape circleShape;
    circleShape.m_radius = radius / Config::PPM;

    b2FixtureDef fixtureDef;
    fixtureDef.density = density;
    fixtureDef.friction = friction;
    fixtureDef.shape = &circleShape;

    b2Body* boxBody = world->CreateBody(&boxBodyDef);

    boxBody->CreateFixture(&fixtureDef);

    return Circle{radius, color, boxBody};
}
```

Here is the caller graph for this function:



4.29.3.3 createGround()

```
Polygon createGround (
            const b2WorldPtr & world,
            float x,
            float y,
            const std::vector< b2Vec2 > & vertices,
            sf::Color color )
```

Create a Ground object.

Parameters

world	2dWorld.
x	X coordinate of the polygon.
y	Y coordinate of the polygon.
vertices	vectors that make up the ground
color	color of the polygon.

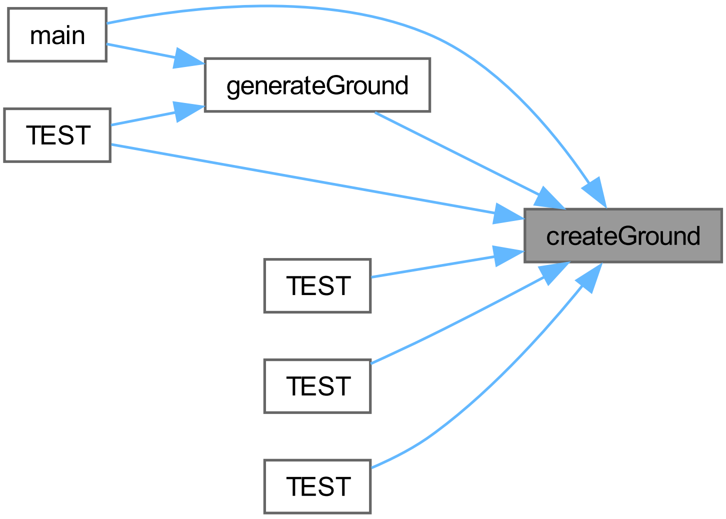
Returns

Polygon

Definition at line 46 of file Shape.cc.

```
                                   {
    // Argument validation
    if (vertices.size() < 3) {
        throw std::invalid_argument("Invalid number of vertices");
    }

    b2BodyDef groundBodyDef;
    groundBodyDef.position.Set(x / Config::PPM, y / Config::PPM);

    b2PolygonShape groundPolygon;
    groundPolygon.Set(vertices.data(), static_cast<int32>(vertices.size()));

    b2Body* groundBody = world->CreateBody(&groundBodyDef);
    groundBody->CreateFixture(&groundPolygon, 0.0f);

    return Polygon{vertices, color, groundBody};
}
```

Here is the caller graph for this function:



4.29.3.4 createPolygon()

```
Polygon createPolygon (
            const b2WorldPtr & world,
            float x,
            float y,
            std::vector< b2Vec2 > vertices,
            float density,
            float friction,
            sf::Color color )
```

Create a Polygon object.

Parameters

world	2dWorld.
x	X coordinate of the polygon.
y	Y coordinate of the polygon.
vertices	vectorst that make up the polygon.
density	density of the polygon.
friction	friction of the polygon.
color	color of the polygon.

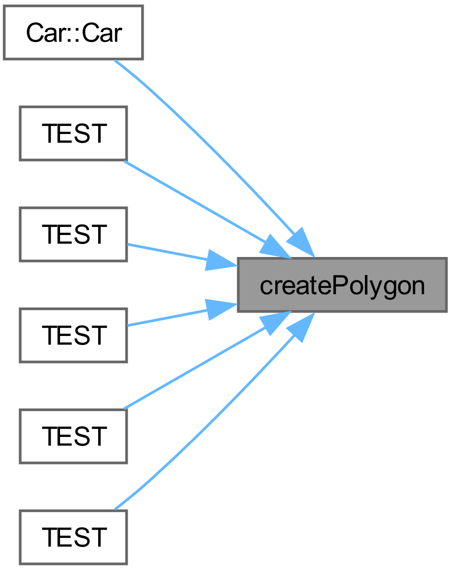
Returns

Polygon

Definition at line 95 of file Shape.cc.

```
                                                                          {
    // Argument validation
    if (vertices.size() < 3 || vertices.size() > CarConfig::CAR_VERTICES) {
        throw std::invalid_argument("Invalid vertices size");
    } else if (density <= 0.0f) {
        throw std::invalid_argument("Invalid density parameter");
    } else if (friction <= 0.0f) {
        throw std::invalid_argument("Invalid friction parameter");
    }
    b2BodyDef boxBodyDef;
    boxBodyDef.position.Set(x / Config::PPM, y / Config::PPM);
    boxBodyDef.type = b2_dynamicBody;

    b2PolygonShape boxShape;
    boxShape.Set(vertices.data(), vertices.size());

    b2FixtureDef fixtureDef;
    fixtureDef.density = density;
    fixtureDef.friction = friction;
    fixtureDef.shape = &boxShape;

    b2Body* boxBody = world->CreateBody(&boxBodyDef);

    boxBody->CreateFixture(&fixtureDef);

    // create a Polygon object with a shared pointer to the b2Body
    return Polygon{vertices, color, boxBody};
}
```

Here is the caller graph for this function:



4.30 Shape.h

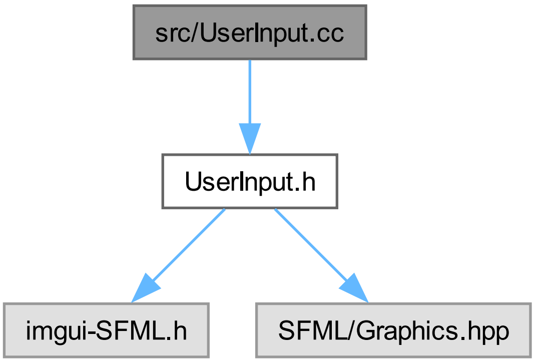
Go to the documentation of this file.

```

#ifndef SHAPE_H
#define SHAPE_H

#include <memory>

#include "box2d/box2d.h"
#include "SFML/Graphics.hpp"

#include "../config/CarConfig.h"
#include "../config/Config.h"

struct Box {
    float width{};
    float height{};
    sf::Color color;
    b2Body* body{};
};

struct Circle {
    float radius{};
    sf::Color color;
    b2Body* body{};
};

struct Polygon {
    std::vector<b2Vec2> vertices;
    sf::Color color;
    b2Body* body;
};

typedef std::shared_ptr<b2World> b2WorldPtr;

Box createBox(const b2WorldPtr& world, float x, float y, float width, float height, float density,
              float friction, sf::Color color);

Polygon createGround(const b2WorldPtr& world, float x, float y, const std::vector<b2Vec2>& vertices,
                     sf::Color color);

Circle createCircle(const b2WorldPtr& world, float x, float y, float radius, float density,
                    float friction, sf::Color color);

Polygon createPolygon(const b2WorldPtr& world, float x, float y, std::vector<b2Vec2> vertices,
                      float density, float friction, sf::Color color);

#endif
```

4.31 src/UserInput.cc File Reference

File containing user input functions.

`#include "UserInput.h"`
Include dependency graph for UserInput.cc:



Functions

- void handleUserInput (sf::RenderWindow &w, bool &paused, bool &pause_check, bool &next_gen, bool &nxt_g_check, bool &focus)

  Function for handling user inputs.
- void handleEvents (sf::RenderWindow &w, bool &pause_check, bool &nxt_g_check, bool &focus)

  Function for handling SFML events.

4.31.1 Detailed Description

File containing user input functions.

Authors

Jakub Marcowski

Date

2023-06-06

Definition in file UserInput.cc.

4.31.2 Function Documentation

4.31.2.1 handleEvents()

```
void handleEvents (
            sf::RenderWindow & w,
            bool & pause_check,
            bool & nxt_g_check,
            bool & focus )
```

Function for handling SFML events.

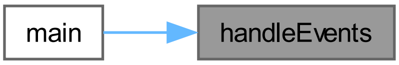
Parameters

w	SFML's RenderWindow.
pause_check	Whether the `paused` was turned on (bool).
nxt_g_check	Whether the `next_gen` was turned on (bool).
focus	Whether the window is in focus (bool).

Definition at line 38 of file UserInput.cc.

```
00038                                                                                      {
00039     // Process events
00040     sf::Event event{};
00041     while (w.pollEvent(event)) {
00042         if (event.type == sf::Event::GainedFocus) {
00043             focus = true;
00044         }
00045         if (event.type == sf::Event::LostFocus) {
00046             focus = false;
00047         }
00048         if (focus) {
00049             ImGui::SFML::ProcessEvent(event);
00050             // Close window : exit
00051             if (event.type == sf::Event::Closed) {
00052                 w.close();
00053             }
00054             if (event.type == sf::Event::KeyReleased) {
00055                 // Allow user to toggle pause again
00056                 if (event.key.code == sf::Keyboard::P || event.key.code == sf::Keyboard::Space) {
00057                     pause_check = true;
00058                 }
00059                 // Allow user to generate the next generation again
00060                 if (event.key.code == sf::Keyboard::N) {
00061                     nxt_g_check = true;
00062                 }
00063             }
00064         }
00065     }
00066 }
```

Here is the caller graph for this function:



4.31.2.2 handleUserInput()

```
void handleUserInput (
            sf::RenderWindow & w,
            bool & paused,
            bool & pause_check,
            bool & next_gen,
            bool & nxt_g_check,
            bool & focus )
```

Function for handling user inputs.

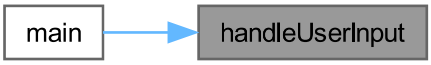
Parameters

w	SFML's RenderWindow.
paused	Whether the simulation is paused (bool).
pause_check	Whether the `paused` was turned on (bool).
next_gen	Whether the program should generate the next generation (bool).
nxt_g_check	Whether the `next_gen` was turned on (bool).
focus	Whether the window is in focus (bool).

Definition at line 11 of file UserInput.cc.

```
                                                       {
    if (focus) {
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Q) ||
            sf::Keyboard::isKeyPressed(sf::Keyboard::Escape)) {
            // Close the window
            w.close();
        }

        if (sf::Keyboard::isKeyPressed(sf::Keyboard::P) ||
            sf::Keyboard::isKeyPressed(sf::Keyboard::Space)) {
            // Pause the simulation
            if (pause_check) {
                paused = !paused;
                pause_check = false;
            }
        }
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::N)) {
            // Generate the next generation
            if (nxt_g_check) {
                next_gen = true;
                nxt_g_check = false;
            }
        }
    }
}
```

Here is the caller graph for this function:



4.32 UserInput.cc

Go to the documentation of this file.

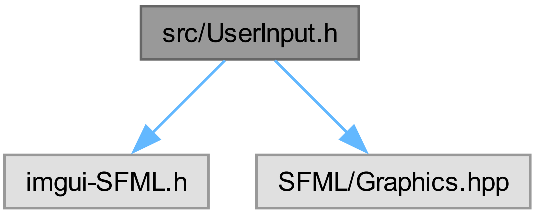
```
#include "UserInput.h"

void handleUserInput(sf::RenderWindow &w, bool &paused, bool &pause_check, bool &next_gen,
                     bool &nxt_g_check, bool &focus) {
    if (focus) {
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Q) ||
            sf::Keyboard::isKeyPressed(sf::Keyboard::Escape)) {
            // Close the window
            w.close();
        }

        if (sf::Keyboard::isKeyPressed(sf::Keyboard::P) ||
            sf::Keyboard::isKeyPressed(sf::Keyboard::Space)) {
            // Pause the simulation
            if (pause_check) {
                paused = !paused;
                pause_check = false;
            }
        }
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::N)) {
            // Generate the next generation
            if (nxt_g_check) {
                next_gen = true;
                nxt_g_check = false;
            }
        }
    }
}

void handleEvents(sf::RenderWindow &w, bool &pause_check, bool &nxt_g_check, bool &focus) {
    // Process events
    sf::Event event{};
    while (w.pollEvent(event)) {
        if (event.type == sf::Event::GainedFocus) {
            focus = true;
        }
        if (event.type == sf::Event::LostFocus) {
            focus = false;
        }
        if (focus) {
            ImGui::SFML::ProcessEvent(event);
            // Close window : exit
            if (event.type == sf::Event::Closed) {
                w.close();
            }
            if (event.type == sf::Event::KeyReleased) {
                // Allow user to toggle pause again
                if (event.key.code == sf::Keyboard::P || event.key.code == sf::Keyboard::Space) {
                    pause_check = true;
                }
                // Allow user to generate the next generation again
                if (event.key.code == sf::Keyboard::N) {
                    nxt_g_check = true;
                }
            }
        }
    }
}
```

4.33 src/UserInput.h File Reference

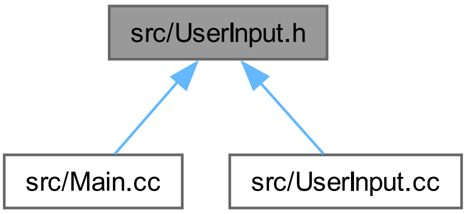
Header for a file containing user input functions.

```
#include ¨imgui-SFML.h¨
#include ¨SFML/Graphics.hpp¨
```

Include dependency graph for UserInput.h:



This graph shows which files directly or indirectly include this file:



Functions

- void handleUserInput (sf::RenderWindow &w, bool &paused, bool &pause_check, bool &next_gen, bool &nxt_g_check, bool &focus)

  Function for handling user inputs.
- void handleEvents (sf::RenderWindow &w, bool &pause_check, bool &nxt_g_check, bool &focus)

  Function for handling SFML events.

4.33.1 Detailed Description

Header for a file containing user input functions.

Authors

Jakub Marcowski

Date

2023-06-06

Definition in file UserInput.h.

4.33.2 Function Documentation

4.33.2.1 handleEvents()

```
void handleEvents (
            sf::RenderWindow & w,
            bool & pause_check,
            bool & nxt_g_check,
            bool & focus )
```

Function for handling SFML events.

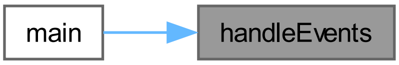
Parameters

w	SFML's RenderWindow.
pause_check	Whether the `paused` was turned on (bool).
nxt_g_check	Whether the `next_gen` was turned on (bool).
focus	Whether the window is in focus (bool).

Definition at line 38 of file UserInput.cc.

```
00038                                                                                            {
00039     // Process events
00040     sf::Event event{};
00041     while (w.pollEvent(event)) {
00042         if (event.type == sf::Event::GainedFocus) {
00043             focus = true;
00044         }
00045         if (event.type == sf::Event::LostFocus) {
00046             focus = false;
00047         }
00048         if (focus) {
00049             ImGui::SFML::ProcessEvent(event);
00050             // Close window : exit
00051             if (event.type == sf::Event::Closed) {
00052                 w.close();
00053             }
00054             if (event.type == sf::Event::KeyReleased) {
00055                 // Allow user to toggle pause again
00056                 if (event.key.code == sf::Keyboard::P || event.key.code == sf::Keyboard::Space) {
00057                     pause_check = true;
00058                 }
00059                 // Allow user to generate the next generation again
00060                 if (event.key.code == sf::Keyboard::N) {
00061                     nxt_g_check = true;
00062                 }
00063             }
00064         }
00065     }
00066 }
```

Here is the caller graph for this function:



4.33.2.2 handleUserInput()

```
void handleUserInput (
              sf::RenderWindow & w,
              bool & paused,
              bool & pause_check,
              bool & next_gen,
              bool & nxt_g_check,
              bool & focus )
```

Function for handling user inputs.

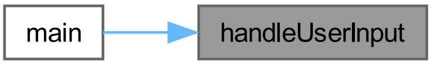
Parameters

w	SFML's RenderWindow.
paused	Whether the simulation is paused (bool).
pause_check	Whether the `paused` was turned on (bool).
next_gen	Whether the program should generate the next generation (bool).
nxt_g_check	Whether the `next_gen` was turned on (bool).
focus	Whether the window is in focus (bool).

Definition at line 11 of file UserInput.cc.

```
                                                     {
    if (focus) {
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Q) ||
            sf::Keyboard::isKeyPressed(sf::Keyboard::Escape)) {
            // Close the window
            w.close();
        }

        if (sf::Keyboard::isKeyPressed(sf::Keyboard::P) ||
            sf::Keyboard::isKeyPressed(sf::Keyboard::Space)) {
            // Pause the simulation
            if (pause_check) {
                paused = !paused;
                pause_check = false;
            }
        }
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::N)) {
            // Generate the next generation
            if (nxt_g_check) {
                next_gen = true;
                nxt_g_check = false;
            }
        }
    }
}
```

Here is the caller graph for this function:



4.34 UserInput.h

Go to the documentation of this file.

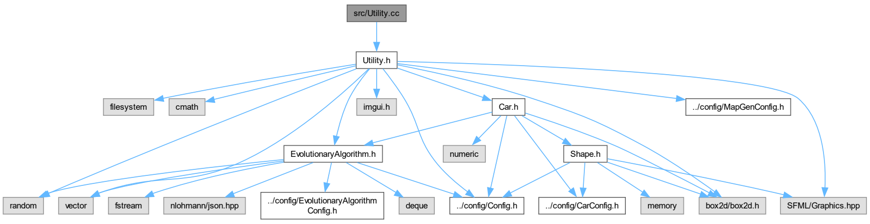
```

#ifndef USER_INPUT_H
#define USER_INPUT_H

#include "imgui-SFML.h"
#include "SFML/Graphics.hpp"

void handleUserInput(sf::RenderWindow &w, bool &paused, bool &pause_check, bool &next_gen,
                     bool &nxt_g_check, bool &focus);

void handleEvents(sf::RenderWindow &w, bool &pause_check, bool &nxt_g_check, bool &focus);

#endif
```

4.35 src/Utility.cc File Reference

File containing utility functions.

`#include ¨Utility.h¨`

Include dependency graph for Utility.cc:



Functions

- void applyAirResistance (Car car)

  Simplified air drag.
- void generateGround (const b2WorldPtr &world, std::vector< Polygon > ∗groundVector, const std::vector< Car > &cars)

  Simplified air drag.
- float getNextGroundPartDegree ()

  Get the angle of the next ground part.
- Car generateCar (const b2WorldPtr &world, const Chromosome &chromosome)
- ImVec4 SFMLColorToImVec4 (sf::Color color)

  Transforms a SFML color into an ImGUI color.
- b2Vec2 getFurthestCarPos (const std::vector< Car > &cars)

  Returns the b2Vec2 position of the car that is the furthest from the starting point.
- int getIdxOfGrndClstToLoc (std::vector< Polygon > ground, float x)

  ¨Get Index Of Ground Closest To Location¨
- void removeCars (const b2WorldPtr &world, std::vector< Car > ∗cars)

  Deletes all cars from the world and the Car vector.
- std::filesystem::path getRootDir ()

  Get the Root Dir object.
- void setIcon (sf::RenderWindow &window)

  Sets the icon of the window.
- std::vector< sf::Texture ∗ > loadBGTextures ()

  Loads the textures for the background and returns them in a vector.

- sf::Sprite loadBGSprite (sf::Texture ∗texture, const std::vector< Car > &cars)

  Loads the sprite for the background.
- std::vector< sf::Sprite > loadBGSprites (std::vector< sf::Texture ∗ > textures, const std::vector< Car > &cars)

  Loads the sprites for the background.

4.35.1 Detailed Description

File containing utility functions.

Authors

Jakub Marcowski, Mateusz Krakowski

Date

2023-06-06

Definition in file Utility.cc.

4.35.2 Function Documentation

4.35.2.1 applyAirResistance()

```
void applyAirResistance (
             Car car )
```

Simplified air drag.

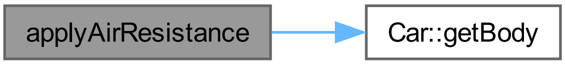
Parameters

car	Car to apply air resistance to.

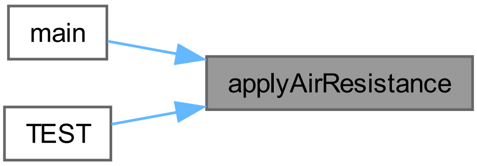
Definition at line 11 of file Utility.cc.

```
00011                              {
00012     // F = V^2 * k
00013     // k  1/2 *   * A * C_d  3.4
00014     //   = 1.293 kg/m^3
00015     // A = ? (let's assume 5 m^2)
00016     // C_d = ? (let's assume 1.05)
00017     //
00018     // F = 3.4 * V^2
00019     car.getBody()->body->ApplyForceToCenter(
00020         b2Vec2(-1.84 * car.getBody()->body->GetLinearVelocity().x *
00021                    abs(car.getBody()->body->GetLinearVelocity().x),
00022                -1.84 * car.getBody()->body->GetLinearVelocity().y *
00023                    abs(car.getBody()->body->GetLinearVelocity().y)),
00024         true);
00025 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



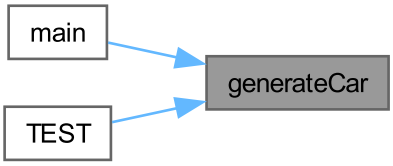
4.35.2.2 generateCar()

```
Car generateCar (
              const b2WorldPtr & world,
              const Chromosome & chromosome )
```

Definition at line 66 of file Utility.cc.

```
                                                                          {
         std::random_device rd;
         std::mt19937 gen(rd());
         std::uniform_int_distribution<> rgb_value(50, 200);

         sf::Color bodyColor = sf::Color(rgb_value(gen), rgb_value(gen), rgb_value(gen));
         sf::Color wheelColor = sf::Color(rgb_value(gen), rgb_value(gen), rgb_value(gen));

         return {world,
                 MapGenConfig::CAR_STARTING_X,
                 MapGenConfig::CAR_STARTING_Y,
                 chromosome,
                 bodyColor,
                 wheelColor};
   }
```

Here is the caller graph for this function:



4.35.2.3 generateGround()

```
void generateGround (
             const b2WorldPtr & world,
             std::vector< Polygon > * groundVector,
             const std::vector< Car > & cars )
```

Simplified air drag.

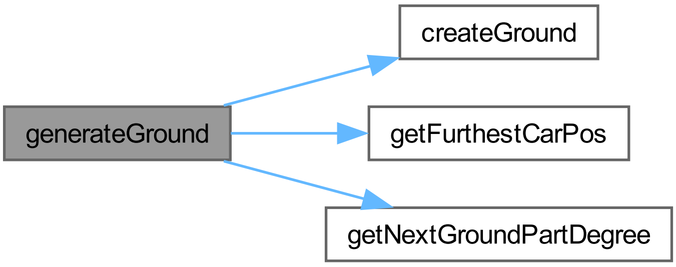
Parameters

world	A shared pointer to the box2d world.
groundVector	A vector of ground polygons.
cars	A vector of cars.

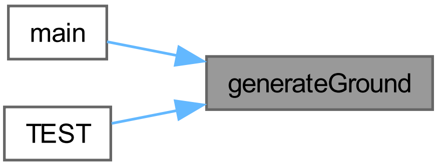
Definition at line 27 of file Utility.cc.

```
00028                                             {
00029     Polygon lastGround = groundVector->back();
00030     if (lastGround.vertices[1].x * Config::PPM <
00031         getFurthestCarPos(cars).x * Config::PPM + MapGenConfig::GENERATE_DISTANCE) {
00032         float degree = getNextGroundPartDegree();
00033         float angle_in_radians = degree * (M_PI / 180.0f);
00034 
00035         float delta_x = MapGenConfig::GROUND_PART_LENGTH * cos(angle_in_radians);
00036         float delta_y = -MapGenConfig::GROUND_PART_LENGTH * sin(angle_in_radians);
00037 
00038         std::vector<b2Vec2> groundVertices = {
00039             b2Vec2(lastGround.vertices[1].x, lastGround.vertices[1].y),
00040             b2Vec2(lastGround.vertices[1].x + delta_x, lastGround.vertices[1].y + delta_y),
00041             b2Vec2(lastGround.vertices[2].x + delta_x, lastGround.vertices[2].y + delta_y)};
00042 
00043         Polygon ground =
00044             createGround(world, MapGenConfig::GROUND_STARTING_X, MapGenConfig::GROUND_STARTING_Y,
00045                          groundVertices, sf::Color(18, 36, 35));
00046 
00047         groundVector->push_back(ground);
00048     }
00049 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.35.2.4 getFurthestCarPos()

```
b2Vec2 getFurthestCarPos (
            const std::vector< Car > & cars )
```

Returns the b2Vec2 position of the car that is the furthest from the starting point.

Parameters

cars	Vector of cars.

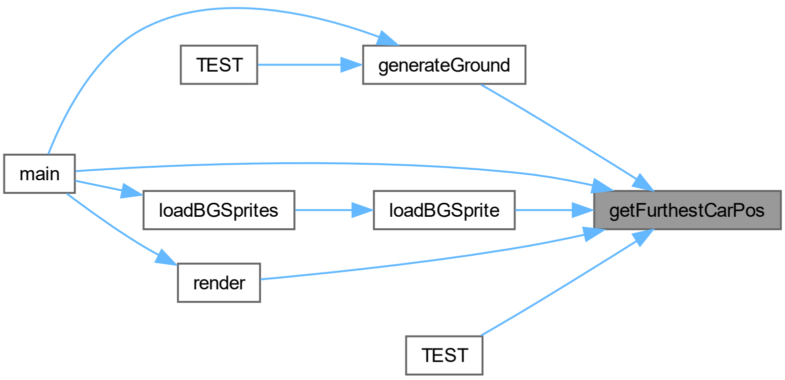
Returns

b2Vec2 Position of the furthest car.

Definition at line 86 of file Utility.cc.

```
00086                                                             {
00087     float furthestCarX = 0;
00088     float furthestCarY = 0;
00089     for (auto car : cars) {
00090         float currentCarX = car.getBody()->body->GetPosition().x;
00091         float currentCarY = car.getBody()->body->GetPosition().y;
00092         if (currentCarX > furthestCarX) {
00093             furthestCarX = currentCarX;
00094             furthestCarY = currentCarY;
00095         }
00096     }
00097     return {furthestCarX, furthestCarY};
00098 }
```

Here is the caller graph for this function:



4.35.2.5 getIdxOfGrndClstToLoc()

```
int getIdxOfGrndClstToLoc (
              std::vector< Polygon > ground,
              float x )
```

¨Get Index Of Ground Closest To Location¨

- returns the index of the ground element that is the closest to the given location.

Parameters

cars	Vector of cars.

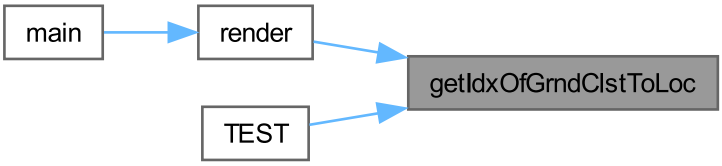
Returns

int Index of the ground element.

Definition at line 100 of file Utility.cc.

```
00100                                                                     {
00101     int index = 0;
00102     for (int i = 0; i < ground.size(); ++i) {
00103         if (ground[i].vertices[0].x - x > 0) {
00104             break;
00105         }
00106         index = i;
00107     }
00108     return index;
00109 }
```

Here is the caller graph for this function:



4.35.2.6 getNextGroundPartDegree()

```
float getNextGroundPartDegree ( )
```

Get the angle of the next ground part.

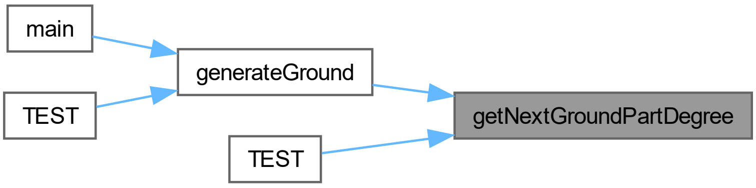
Returns

float

Definition at line 51 of file Utility.cc.

```
00051                              {
00052     std::random_device rd;
00053     std::mt19937 gen(rd());
00054     std::normal_distribution<float> dist(0.0, MapGenConfig::GROUND_DEGREE_DEVIATION);
00055 
00056     float degree = dist(gen);
00057     if (degree > MapGenConfig::MAX_GROUND_DEGREE) {
00058         degree = MapGenConfig::MAX_GROUND_DEGREE;
00059     } else if (degree < -MapGenConfig::MAX_GROUND_DEGREE) {
00060         degree = -MapGenConfig::MAX_GROUND_DEGREE;
00061     }
00062 
00063     return degree;
00064 }
```

Here is the caller graph for this function:



4.35.2.7 getRootDir()

```
std::filesystem::path getRootDir ( )
```

Get the Root Dir object.

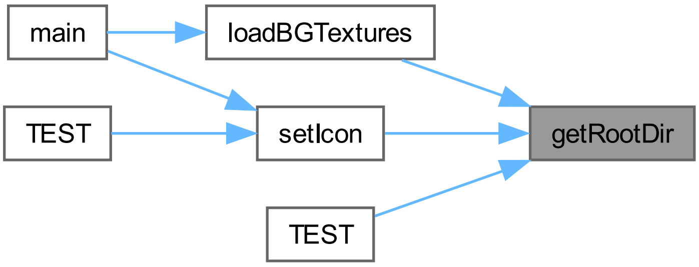
Returns

std::filesystem::path to the root directory

Definition at line 120 of file Utility.cc.

```
00120                              {
00121     std::filesystem::path filePath = std::filesystem::path(__FILE__);
00122     std::filesystem::path dirPath = filePath.parent_path();
00123     return dirPath;
00124 }
```

Here is the caller graph for this function:



4.35.2.8 loadBGSprite()

```
sf::Sprite loadBGSprite (
            sf::Texture * texture,
            const std::vector< Car > & cars )
```

Loads the sprite for the background.

Parameters

texture	
cars	

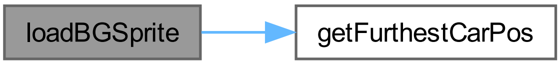
Returns

sf::Sprite

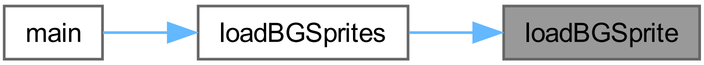
Definition at line 147 of file Utility.cc.

```
                                                                                            {
          sf::Sprite sprite(*texture);
          sprite.setScale(sf::Vector2f(Config::WINDOW_WIDTH / sprite.getLocalBounds().width,
                                       Config::WINDOW_HEIGHT / sprite.getLocalBounds().height));
          sprite.setTextureRect(sf::IntRect(0, 0, 256 * Config::WINDOW_WIDTH, Config::WINDOW_HEIGHT));
          sprite.setPosition(
              sf::Vector2f(getFurthestCarPos(cars).x * Config::PPM, 0.5 * Config::WINDOW_HEIGHT) -
              sf::Vector2f(Config::WINDOW_WIDTH / 2.0f, Config::WINDOW_HEIGHT / 2.0f));
          return sprite;
      }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.35.2.9 loadBGSprites()

```
std::vector< sf::Sprite > loadBGSprites (
            std::vector< sf::Texture * > textures,
            const std::vector< Car > & cars )
```

Loads the sprites for the background.

Parameters

textures	
cars	

Returns

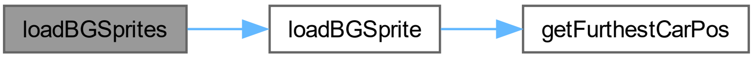
std::vector<sf::Sprite>

Definition at line 158 of file Utility.cc.

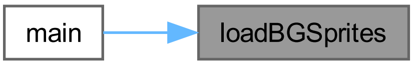
```
                                                                                 {
          std::vector<sf::Sprite> sprites;
```

```
00161     sprites.reserve(MapGenConfig::BG_SPRITES_COUNT);
00162     for (int i = 0; i < MapGenConfig::BG_SPRITES_COUNT; ++i) {
00163         sprites.push_back(loadBGSprite(textures[i], cars));
00164     }
00165     return sprites;
00166 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.35.2.10 loadBGTextures()

```
std::vector< sf::Texture * > loadBGTextures ( )
```

Loads the textures for the background and returns them in a vector.

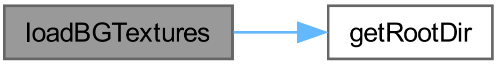
Returns

std::vector<sf::Texture*>

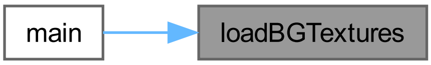
Definition at line 134 of file Utility.cc.

```
00134                                              {
00135     std::vector<sf::Texture*> textures;
00136     for (int i = 0; i < MapGenConfig::BG_SPRITES_COUNT; ++i) {
00137         std::string BGPath =
00138             (getRootDir() / ("../resources/background_img_" + std::to_string(i) + ".png")).string();
00139         auto* texture = new sf::Texture();
00140         texture->loadFromFile(BGPath);
00141         texture->setRepeated(true);
00142         textures.push_back(texture);
00143     }
00144     return textures;
00145 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.35.2.11 removeCars()

```
void removeCars (
            const b2WorldPtr & world,
            std::vector< Car > * cars )
```

Deletes all cars from the world and the Car vector.

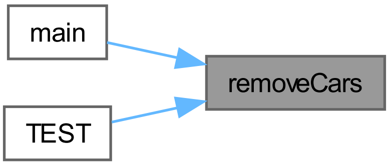
Parameters

world	A shared pointer to the box2d world.
cars	A vector of cars.

Definition at line 111 of file Utility.cc.

```
00111                                                                {
00112     for (auto car : *cars) {
00113         world->DestroyBody(car.getBody()->body);
00114         world->DestroyBody(car.getBackWheel()->body);
00115         world->DestroyBody(car.getFrontWheel()->body);
00116     }
00117     cars->clear();
00118 }
```

Here is the caller graph for this function:



4.35.2.12 setIcon()

```
void setIcon (
            sf::RenderWindow & window )
```

Sets the icon of the window.

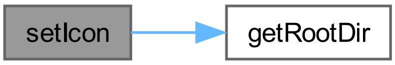
Parameters

window	

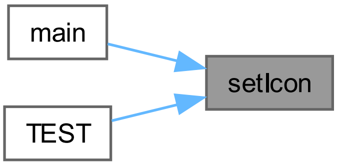
Definition at line 126 of file Utility.cc.

```
00126                                    {
00127     std::string iconPath = (getRootDir() / "../resources/evoracer_icon.png").string();
00128     auto icon = sf::Image{};
00129     if (icon.loadFromFile(iconPath)) {
00130         window.setIcon(icon.getSize().x, icon.getSize().y, icon.getPixelsPtr());
00131     }
00132 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.35.2.13 SFMLColorToImVec4()

```
ImVec4 SFMLColorToImVec4 (
              sf::Color color )
```

Transforms a SFML color into an ImGUI color.

Parameters

color	SFML color.

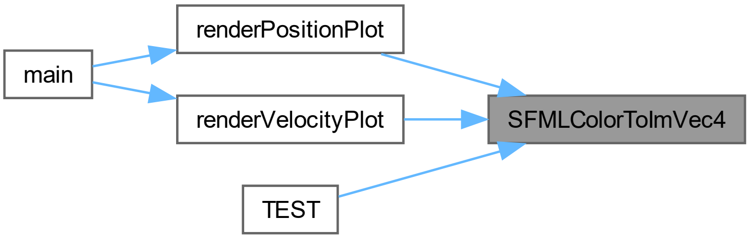
Returns

ImVec4 ImGUI color.

Definition at line 82 of file Utility.cc.

```
00082                                        {
00083     return {color.r / 255.0f, color.g / 255.0f, color.b / 255.0f, color.a / 255.0f};
00084 }
```

Here is the caller graph for this function:



4.36 Utility.cc

Go to the documentation of this file.

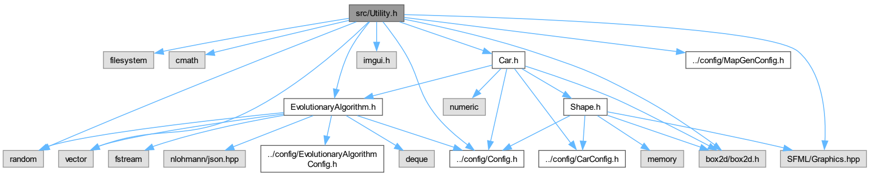
```cpp

#include "Utility.h"

void applyAirResistance(Car car) {
    // F = V^2 * k
    // k  1/2 *  * A * C_d  3.4
    //   = 1.293 kg/m^3
    // A = ? (let’s assume 5 m^2)
    // C_d = ? (let’s assume 1.05)
    //
    // F = 3.4 * V^2
    car.getBody()->body->ApplyForceToCenter(
        b2Vec2(-1.84 * car.getBody()->body->GetLinearVelocity().x *
                   abs(car.getBody()->body->GetLinearVelocity().x),
               -1.84 * car.getBody()->body->GetLinearVelocity().y *
                   abs(car.getBody()->body->GetLinearVelocity().y)),
        true);
}

void generateGround(const b2WorldPtr& world, std::vector<Polygon>* groundVector,
                    const std::vector<Car>& cars) {
    Polygon lastGround = groundVector->back();
    if (lastGround.vertices[1].x * Config::PPM <
        getFurthestCarPos(cars).x * Config::PPM + MapGenConfig::GENERATE_DISTANCE) {
        float degree = getNextGroundPartDegree();
        float angle_in_radians = degree * (M_PI / 180.0f);

        float delta_x = MapGenConfig::GROUND_PART_LENGTH * cos(angle_in_radians);
        float delta_y = -MapGenConfig::GROUND_PART_LENGTH * sin(angle_in_radians);

        std::vector<b2Vec2> groundVertices = {
            b2Vec2(lastGround.vertices[1].x, lastGround.vertices[1].y),
            b2Vec2(lastGround.vertices[1].x + delta_x, lastGround.vertices[1].y + delta_y),
            b2Vec2(lastGround.vertices[2].x + delta_x, lastGround.vertices[2].y + delta_y)};

        Polygon ground =
            createGround(world, MapGenConfig::GROUND_STARTING_X, MapGenConfig::GROUND_STARTING_Y,
                         groundVertices, sf::Color(18, 36, 35));

        groundVector->push_back(ground);
    }
}

float getNextGroundPartDegree() {
    std::random_device rd;
    std::mt19937 gen(rd());
    std::normal_distribution<float> dist(0.0, MapGenConfig::GROUND_DEGREE_DEVIATION);

    float degree = dist(gen);
    if (degree > MapGenConfig::MAX_GROUND_DEGREE) {
        degree = MapGenConfig::MAX_GROUND_DEGREE;
    } else if (degree < -MapGenConfig::MAX_GROUND_DEGREE) {
        degree = -MapGenConfig::MAX_GROUND_DEGREE;
    }

    return degree;
}

Car generateCar(const b2WorldPtr& world, const Chromosome& chromosome) {
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_int_distribution<> rgb_value(50, 200);

    sf::Color bodyColor = sf::Color(rgb_value(gen), rgb_value(gen), rgb_value(gen));
    sf::Color wheelColor = sf::Color(rgb_value(gen), rgb_value(gen), rgb_value(gen));

    return {world,
            MapGenConfig::CAR_STARTING_X,
            MapGenConfig::CAR_STARTING_Y,
            chromosome,
            bodyColor,
            wheelColor};
}

ImVec4 SFMLColorToImVec4(sf::Color color) {
    return {color.r / 255.0f, color.g / 255.0f, color.b / 255.0f, color.a / 255.0f};
}

b2Vec2 getFurthestCarPos(const std::vector<Car>& cars) {
    float furthestCarX = 0;
    float furthestCarY = 0;
    for (auto car : cars) {
        float currentCarX = car.getBody()->body->GetPosition().x;
        float currentCarY = car.getBody()->body->GetPosition().y;
        if (currentCarX > furthestCarX) {
            furthestCarX = currentCarX;
            furthestCarY = currentCarY;
        }
    }
    return {furthestCarX, furthestCarY};
}

int getIdxOfGrndClstToLoc(std::vector<Polygon> ground, float x) {
    int index = 0;
    for (int i = 0; i < ground.size(); ++i) {
        if (ground[i].vertices[0].x - x > 0) {
            break;
        }
        index = i;
    }
    return index;
}

void removeCars(const b2WorldPtr& world, std::vector<Car>* cars) {
    for (auto car : *cars) {
        world->DestroyBody(car.getBody()->body);
        world->DestroyBody(car.getBackWheel()->body);
        world->DestroyBody(car.getFrontWheel()->body);
    }
    cars->clear();
}

std::filesystem::path getRootDir() {
    std::filesystem::path filePath = std::filesystem::path(__FILE__);
    std::filesystem::path dirPath = filePath.parent_path();
    return dirPath;
}

void setIcon(sf::RenderWindow& window) {
    std::string iconPath = (getRootDir() / "../resources/evoracer_icon.png").string();
    auto icon = sf::Image{};
    if (icon.loadFromFile(iconPath)) {
        window.setIcon(icon.getSize().x, icon.getSize().y, icon.getPixelsPtr());
    }
}

std::vector<sf::Texture*> loadBGTextures() {
    std::vector<sf::Texture*> textures;
    for (int i = 0; i < MapGenConfig::BG_SPRITES_COUNT; ++i) {
        std::string BGPath =
            (getRootDir() / ("../resources/background_img_" + std::to_string(i) + ".png")).string();
        auto* texture = new sf::Texture();
        texture->loadFromFile(BGPath);
        texture->setRepeated(true);
        textures.push_back(texture);
    }
    return textures;
}

sf::Sprite loadBGSprite(sf::Texture* texture, const std::vector<Car>& cars) {
    sf::Sprite sprite(*texture);
    sprite.setScale(sf::Vector2f(Config::WINDOW_WIDTH / sprite.getLocalBounds().width,
                                 Config::WINDOW_HEIGHT / sprite.getLocalBounds().height));
    sprite.setTextureRect(sf::IntRect(0, 0, 256 * Config::WINDOW_WIDTH, Config::WINDOW_HEIGHT));
    sprite.setPosition(
        sf::Vector2f(getFurthestCarPos(cars).x * Config::PPM, 0.5 * Config::WINDOW_HEIGHT) -
        sf::Vector2f(Config::WINDOW_WIDTH / 2.0f, Config::WINDOW_HEIGHT / 2.0f));
    return sprite;
}

std::vector<sf::Sprite> loadBGSprites(std::vector<sf::Texture*> textures,
                                      const std::vector<Car>& cars) {
    std::vector<sf::Sprite> sprites;
    sprites.reserve(MapGenConfig::BG_SPRITES_COUNT);
    for (int i = 0; i < MapGenConfig::BG_SPRITES_COUNT; ++i) {
        sprites.push_back(loadBGSprite(textures[i], cars));
    }
    return sprites;
}
```

4.37 src/Utility.h File Reference

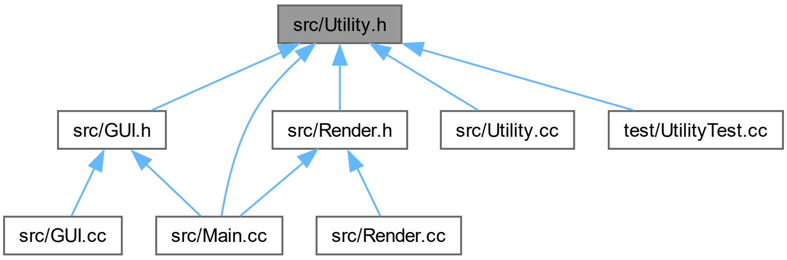
Header for a file containing utility functions.

```
#include <filesystem>
#include <cmath>
#include <random>
#include <vector>
#include ¨box2d/box2d.h¨
#include ¨imgui.h¨
#include ¨SFML/Graphics.hpp¨
#include ¨EvolutionaryAlgorithm.h¨
#include ¨../config/Config.h¨
#include ¨../config/MapGenConfig.h¨
#include ¨Car.h¨
```

Include dependency graph for Utility.h:



This graph shows which files directly or indirectly include this file:



Typedefs

- typedef std::shared_ptr< b2World > b2WorldPtr

Functions

- void applyAirResistance (Car car)

  Simplified air drag.
- void generateGround (const b2WorldPtr &world, std::vector< Polygon > ∗groundVector, const std::vector< Car > &cars)

  Simplified air drag.
- float getNextGroundPartDegree ()

  Get the angle of the next ground part.
- Car generateCar (const b2WorldPtr &world, const Chromosome &chromosome)
- ImVec4 SFMLColorToImVec4 (sf::Color color)

Transforms a SFML color into an ImGUI color.

- b2Vec2 getFurthestCarPos (const std::vector< Car > &cars)

  Returns the b2Vec2 position of the car that is the furthest from the starting point.
- int getIdxOfGrndClstToLoc (std::vector< Polygon > ground, float x)

  ¨Get Index Of Ground Closest To Location¨
- void removeCars (const b2WorldPtr &world, std::vector< Car > ∗cars)

  Deletes all cars from the world and the Car vector.
- std::filesystem::path getRootDir ()

  Get the Root Dir object.
- void setIcon (sf::RenderWindow &window)

  Sets the icon of the window.
- std::vector< sf::Texture ∗ > loadBGTextures ()

  Loads the textures for the background and returns them in a vector.
- sf::Sprite loadBGSprite (sf::Texture ∗texture, const std::vector< Car > &cars)

  Loads the sprite for the background.
- std::vector< sf::Sprite > loadBGSprites (std::vector< sf::Texture ∗ > textures, const std::vector< Car > &cars)

  Loads the sprites for the background.

4.37.1 Detailed Description

Header for a file containing utility functions.

Authors

Jakub Marcowski, Mateusz Krakowski

Date

2023-06-06

Definition in file Utility.h.

4.37.2 Typedef Documentation

4.37.2.1 b2WorldPtr

```
typedef std::shared_ptr<b2World> b2WorldPtr
```

Definition at line 26 of file Utility.h.

4.37.3 Function Documentation

4.37.3.1 applyAirResistance()

```
void applyAirResistance (
            Car car )
```

Simplified air drag.

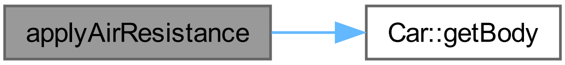
Parameters

car	Car to apply air resistance to.

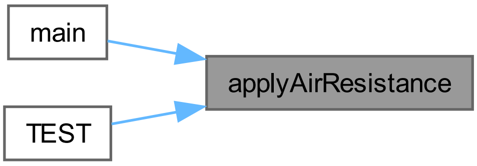
Definition at line 11 of file Utility.cc.

```
00011                                {
00012     // F = V^2 * k
00013     // k  1/2 *  * A * C_d  3.4
00014     //  = 1.293 kg/m^3
00015     // A = ? (let's assume 5 m^2)
00016     // C_d = ? (let's assume 1.05)
00017     //
00018     // F = 3.4 * V^2
00019     car.getBody()->body->ApplyForceToCenter(
00020         b2Vec2(-1.84 * car.getBody()->body->GetLinearVelocity().x *
00021                    abs(car.getBody()->body->GetLinearVelocity().x),
00022                -1.84 * car.getBody()->body->GetLinearVelocity().y *
00023                    abs(car.getBody()->body->GetLinearVelocity().y)),
00024         true);
00025 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.37.3.2 generateCar()

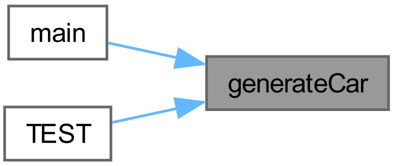
```
Car generateCar (
            const b2WorldPtr & world,
            const Chromosome & chromosome )
```

Definition at line 66 of file Utility.cc.

```
00066                                                               {
00067     std::random_device rd;
00068     std::mt19937 gen(rd());
00069     std::uniform_int_distribution<> rgb_value(50, 200);
```

```

    sf::Color bodyColor = sf::Color(rgb_value(gen), rgb_value(gen), rgb_value(gen));
    sf::Color wheelColor = sf::Color(rgb_value(gen), rgb_value(gen), rgb_value(gen));

    return {world,
            MapGenConfig::CAR_STARTING_X,
            MapGenConfig::CAR_STARTING_Y,
            chromosome,
            bodyColor,
            wheelColor};
}
```

Here is the caller graph for this function:



4.37.3.3 generateGround()

```
void generateGround (
               const b2WorldPtr & world,
               std::vector< Polygon > * groundVector,
               const std::vector< Car > & cars )
```

Simplified air drag.

Parameters

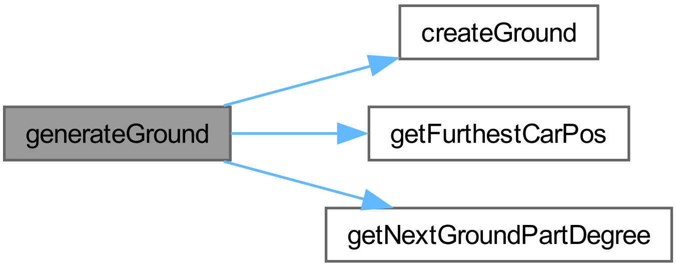
world	A shared pointer to the box2d world.
groundVector	A vector of ground polygons.
cars	A vector of cars.

Definition at line 27 of file Utility.cc.

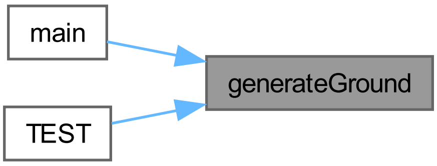
```
                                                {
    Polygon lastGround = groundVector->back();
    if (lastGround.vertices[1].x * Config::PPM <
        getFurthestCarPos(cars).x * Config::PPM + MapGenConfig::GENERATE_DISTANCE) {
        float degree = getNextGroundPartDegree();
        float angle_in_radians = degree * (M_PI / 180.0f);

        float delta_x = MapGenConfig::GROUND_PART_LENGTH * cos(angle_in_radians);
        float delta_y = -MapGenConfig::GROUND_PART_LENGTH * sin(angle_in_radians);

        std::vector<b2Vec2> groundVertices = {
            b2Vec2(lastGround.vertices[1].x, lastGround.vertices[1].y),
            b2Vec2(lastGround.vertices[1].x + delta_x, lastGround.vertices[1].y + delta_y),
            b2Vec2(lastGround.vertices[2].x + delta_x, lastGround.vertices[2].y + delta_y)};

        Polygon ground =
            createGround(world, MapGenConfig::GROUND_STARTING_X, MapGenConfig::GROUND_STARTING_Y,
                         groundVertices, sf::Color(18, 36, 35));

        groundVector->push_back(ground);
```

```
00048     }
00049 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.37.3.4 getFurthestCarPos()

```
b2Vec2 getFurthestCarPos (
            const std::vector< Car > & cars )
```

Returns the b2Vec2 position of the car that is the furthest from the starting point.

Parameters

cars	Vector of cars.

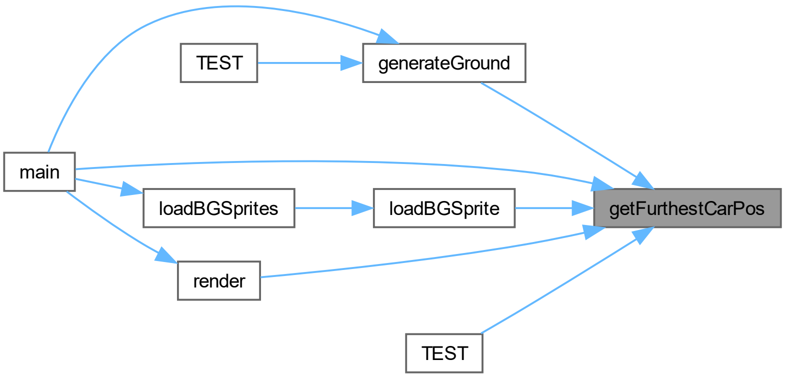
Returns

b2Vec2 Position of the furthest car.

Definition at line 86 of file Utility.cc.

```
                                                            {
          float furthestCarX = 0;
          float furthestCarY = 0;
          for (auto car : cars) {
              float currentCarX = car.getBody()->body->GetPosition().x;
              float currentCarY = car.getBody()->body->GetPosition().y;
              if (currentCarX > furthestCarX) {
                  furthestCarX = currentCarX;
                  furthestCarY = currentCarY;
              }
          }
          return {furthestCarX, furthestCarY};
      }
```

Here is the caller graph for this function:



4.37.3.5 getIdxOfGrndClstToLoc()

```
int getIdxOfGrndClstToLoc (
              std::vector< Polygon > ground,
              float x )
```

¨Get Index Of Ground Closest To Location¨

- returns the index of the ground element that is the closest to the given location.

Parameters

cars	Vector of cars.

Returns

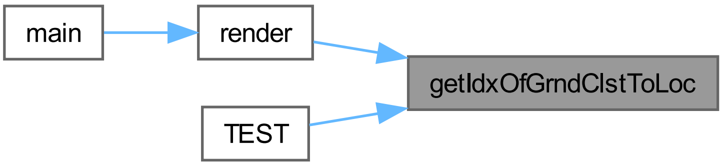
int Index of the ground element.

Definition at line 100 of file Utility.cc.

```
                                                                {
          int index = 0;
          for (int i = 0; i < ground.size(); ++i) {
              if (ground[i].vertices[0].x - x > 0) {
```

```
00104             break;
00105         }
00106         index = i;
00107     }
00108     return index;
00109 }
```

Here is the caller graph for this function:



4.37.3.6 getNextGroundPartDegree()

```
float getNextGroundPartDegree ( )
```

Get the angle of the next ground part.

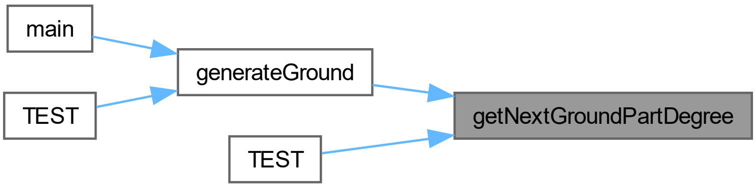
Returns

float

Definition at line 51 of file Utility.cc.

```
00051                                {
00052     std::random_device rd;
00053     std::mt19937 gen(rd());
00054     std::normal_distribution<float> dist(0.0, MapGenConfig::GROUND_DEGREE_DEVIATION);
00055 
00056     float degree = dist(gen);
00057     if (degree > MapGenConfig::MAX_GROUND_DEGREE) {
00058         degree = MapGenConfig::MAX_GROUND_DEGREE;
00059     } else if (degree < -MapGenConfig::MAX_GROUND_DEGREE) {
00060         degree = -MapGenConfig::MAX_GROUND_DEGREE;
00061     }
00062 
00063     return degree;
00064 }
```

Here is the caller graph for this function:



4.37.3.7 getRootDir()

```
std::filesystem::path getRootDir ( )
```

Get the Root Dir object.

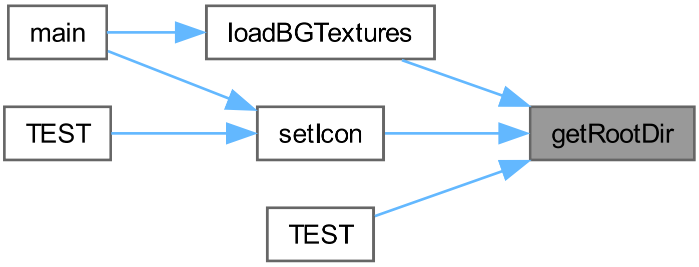
Returns

std::filesystem::path to the root directory

Definition at line 120 of file Utility.cc.

```
00120                              {
00121     std::filesystem::path filePath = std::filesystem::path(__FILE__);
00122     std::filesystem::path dirPath = filePath.parent_path();
00123     return dirPath;
00124 }
```

Here is the caller graph for this function:



4.37.3.8 loadBGSprite()

```
sf::Sprite loadBGSprite (
             sf::Texture * texture,
             const std::vector< Car > & cars )
```

Loads the sprite for the background.

Parameters

texture	
cars	

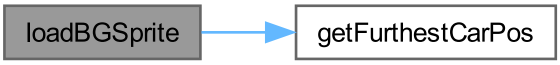
Returns

sf::Sprite

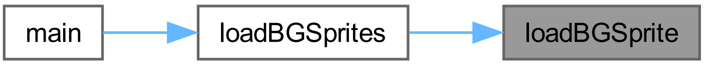
Definition at line 147 of file Utility.cc.

```
                                                                  {
    sf::Sprite sprite(*texture);
    sprite.setScale(sf::Vector2f(Config::WINDOW_WIDTH / sprite.getLocalBounds().width,
                                 Config::WINDOW_HEIGHT / sprite.getLocalBounds().height));
    sprite.setTextureRect(sf::IntRect(0, 0, 256 * Config::WINDOW_WIDTH, Config::WINDOW_HEIGHT));
    sprite.setPosition(
        sf::Vector2f(getFurthestCarPos(cars).x * Config::PPM, 0.5 * Config::WINDOW_HEIGHT) -
        sf::Vector2f(Config::WINDOW_WIDTH / 2.0f, Config::WINDOW_HEIGHT / 2.0f));
    return sprite;
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.37.3.9 loadBGSprites()

```
std::vector< sf::Sprite > loadBGSprites (
            std::vector< sf::Texture * > textures,
            const std::vector< Car > & cars )
```

Loads the sprites for the background.

Parameters

textures	
cars	

Returns

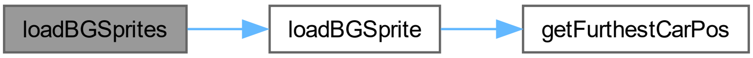
std::vector<sf::Sprite>

Definition at line 158 of file Utility.cc.

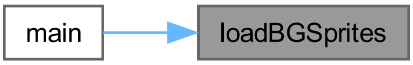
```
                                                          {
    std::vector<sf::Sprite> sprites;
```

```
00161     sprites.reserve(MapGenConfig::BG_SPRITES_COUNT);
00162     for (int i = 0; i < MapGenConfig::BG_SPRITES_COUNT; ++i) {
00163         sprites.push_back(loadBGSprite(textures[i], cars));
00164     }
00165     return sprites;
00166 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.37.3.10 loadBGTextures()

```
std::vector< sf::Texture * > loadBGTextures ( )
```

Loads the textures for the background and returns them in a vector.

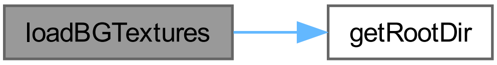
Returns

std::vector<sf::Texture∗>

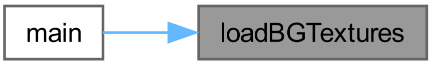
Definition at line 134 of file Utility.cc.

```
00134                                              {
00135     std::vector<sf::Texture*> textures;
00136     for (int i = 0; i < MapGenConfig::BG_SPRITES_COUNT; ++i) {
00137         std::string BGPath =
00138             (getRootDir() / ("../resources/background_img_" + std::to_string(i) + ".png")).string();
00139         auto* texture = new sf::Texture();
00140         texture->loadFromFile(BGPath);
00141         texture->setRepeated(true);
00142         textures.push_back(texture);
00143     }
00144     return textures;
00145 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.37.3.11 removeCars()

```
void removeCars (
             const b2WorldPtr & world,
             std::vector< Car > * cars )
```

Deletes all cars from the world and the Car vector.

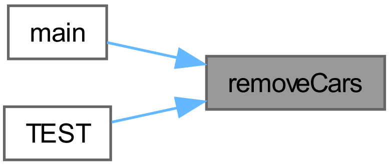
Parameters

world	A shared pointer to the box2d world.
cars	A vector of cars.

Definition at line 111 of file Utility.cc.

```
00111                                                             {
00112     for (auto car : *cars) {
00113         world->DestroyBody(car.getBody()->body);
00114         world->DestroyBody(car.getBackWheel()->body);
00115         world->DestroyBody(car.getFrontWheel()->body);
00116     }
00117     cars->clear();
00118 }
```

Here is the caller graph for this function:



4.37.3.12 setIcon()

```
void setIcon (
             sf::RenderWindow & window )
```

Sets the icon of the window.

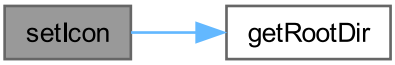
Parameters

window	

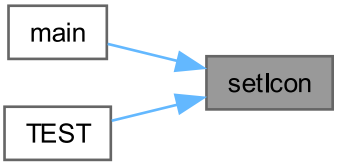
Definition at line 126 of file Utility.cc.

```
00126                                    {
00127     std::string iconPath = (getRootDir() / "../resources/evoracer_icon.png").string();
00128     auto icon = sf::Image{};
00129     if (icon.loadFromFile(iconPath)) {
00130         window.setIcon(icon.getSize().x, icon.getSize().y, icon.getPixelsPtr());
00131     }
00132 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.37.3.13 SFMLColorToImVec4()

```
ImVec4 SFMLColorToImVec4 (
             sf::Color color )
```

Transforms a SFML color into an ImGUI color.

Parameters

color	SFML color.

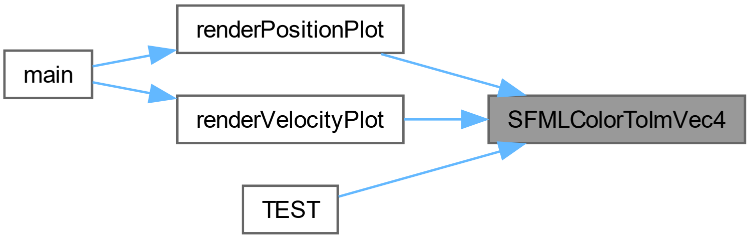
Returns

ImVec4 ImGUI color.

Definition at line 82 of file Utility.cc.

```
00082                                    {
00083     return {color.r / 255.0f, color.g / 255.0f, color.b / 255.0f, color.a / 255.0f};
00084 }
```

Here is the caller graph for this function:



4.38 Utility.h

Go to the documentation of this file.

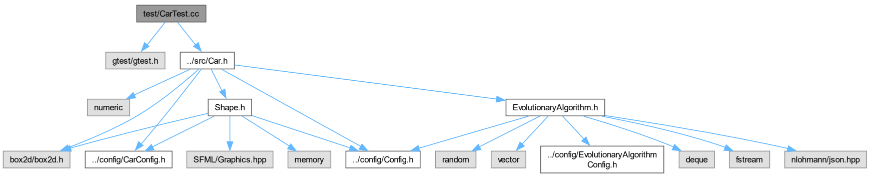
```
#ifndef UTILITY_H
#define UTILITY_H

#include <filesystem>
#include <cmath>
#include <random>
#include <vector>

#include "box2d/box2d.h"
#include "imgui.h"
#include "SFML/Graphics.hpp"
#include "EvolutionaryAlgorithm.h"

#include "../config/Config.h"
#include "../config/MapGenConfig.h"
#include "Car.h"

typedef std::shared_ptr<b2World> b2WorldPtr;

// TODO: think if some of these functions should be moved to other files
// and whether or not some constants should be moved to config

void applyAirResistance(Car car);

void generateGround(const b2WorldPtr& world, std::vector<Polygon>* groundVector,
                    const std::vector<Car>& cars);

float getNextGroundPartDegree();

Car generateCar(const b2WorldPtr& world, const Chromosome& chromosome);

ImVec4 SFMLColorToImVec4(sf::Color color);

b2Vec2 getFurthestCarPos(const std::vector<Car>& cars);

int getIdxOfGrndClstToLoc(std::vector<Polygon> ground, float x);

void removeCars(const b2WorldPtr& world, std::vector<Car>* cars);

std::filesystem::path getRootDir();

void setIcon(sf::RenderWindow& window);

std::vector<sf::Texture*> loadBGTextures();

sf::Sprite loadBGSprite(sf::Texture* texture, const std::vector<Car>& cars);

std::vector<sf::Sprite> loadBGSprites(std::vector<sf::Texture*> textures,
                                      const std::vector<Car>& cars);

#endif  // UTILITY_H
```

4.39 test/CarTest.cc File Reference

This file contains tests for functions from src/Car.h.

```
#include <gtest/gtest.h>
#include ¨../src/Car.h¨
```

Include dependency graph for CarTest.cc:



Functions

- TEST (CreateCarTest, BasicTest)

4.39.1 Detailed Description

This file contains tests for functions from src/Car.h.

Author

Mateusz Krakowski

Date

2023-06-06

Author

Mateusz Krakowski

Date

2023-06-03

Definition in file CarTest.cc.

4.39.2 Function Documentation

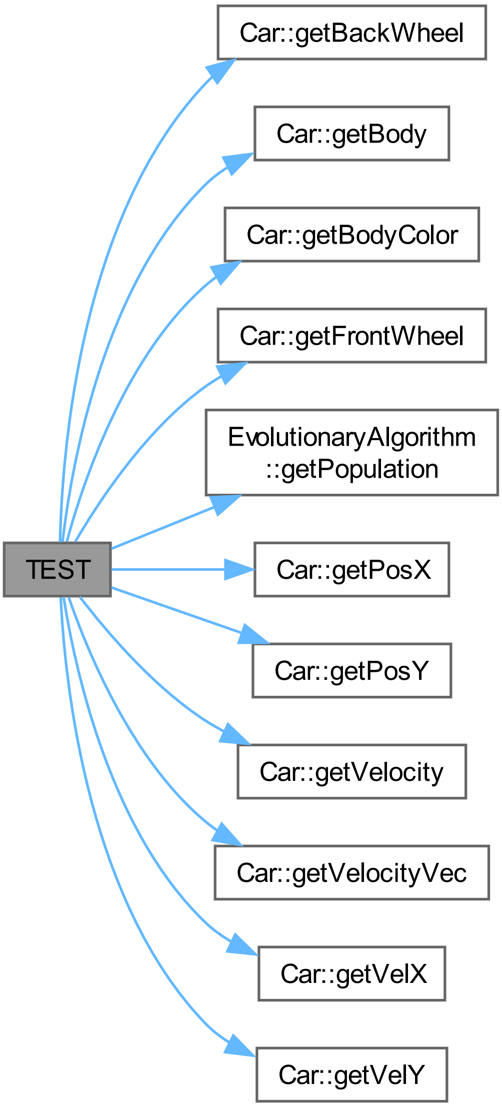
4.39.2.1 TEST()

```
TEST (
            CreateCarTest ,
            BasicTest   )
```

Definition at line 13 of file CarTest.cc.

```
00013                                  {
00014     b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
00015     float x = 0.0f, y = 0.0f;
00016     sf::Color bodyColor = sf::Color::Red;
00017     sf::Color wheelColor = sf::Color::Blue;
00018 
00019     EvolutionaryAlgorithm ea = EvolutionaryAlgorithm(EvolutionaryAlgorithmConfig::POPULATION_SIZE);
00020     Chromosome chromosome = ea.getPopulation()[0];
00021     Car car = Car(world, x, y, chromosome, bodyColor, wheelColor);
00022 
00023     EXPECT_EQ(car.getPosX(), x);
00024     EXPECT_EQ(car.getPosY(), y);
00025     EXPECT_EQ(car.getBodyColor(), bodyColor);
00026     EXPECT_NO_THROW(car.getBody());
00027     EXPECT_NO_THROW(car.getFrontWheel());
00028     EXPECT_NO_THROW(car.getBackWheel());
00029     EXPECT_NO_THROW(car.getVelX());
00030     EXPECT_NO_THROW(car.getVelY());
00031     EXPECT_NO_THROW(car.getVelocityVec());
00032     EXPECT_NO_THROW(car.getVelocity());
00033 }
```

Here is the call graph for this function:



4.40 CarTest.cc

Go to the documentation of this file.

```
00001
00009 #include <gtest/gtest.h>
00010
00011 #include "../src/Car.h"
00012
00013 TEST(CreateCarTest, BasicTest) {
00014     b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
00015     float x = 0.0f, y = 0.0f;
00016     sf::Color bodyColor = sf::Color::Red;
00017     sf::Color wheelColor = sf::Color::Blue;
00018
00019     EvolutionaryAlgorithm ea = EvolutionaryAlgorithm(EvolutionaryAlgorithmConfig::POPULATION_SIZE);
```

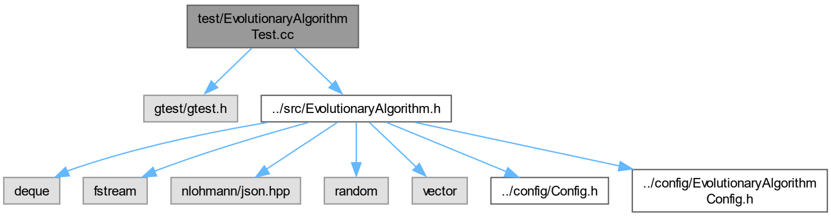
```
    Chromosome chromosome = ea.getPopulation()[0];
    Car car = Car(world, x, y, chromosome, bodyColor, wheelColor);

    EXPECT_EQ(car.getPosX(), x);
    EXPECT_EQ(car.getPosY(), y);
    EXPECT_EQ(car.getBodyColor(), bodyColor);
    EXPECT_NO_THROW(car.getBody());
    EXPECT_NO_THROW(car.getFrontWheel());
    EXPECT_NO_THROW(car.getBackWheel());
    EXPECT_NO_THROW(car.getVelX());
    EXPECT_NO_THROW(car.getVelY());
    EXPECT_NO_THROW(car.getVelocityVec());
    EXPECT_NO_THROW(car.getVelocity());
}
```

4.41 test/EvolutionaryAlgorithmTest.cc File Reference

This file contains tests for functions from src/EvolutionaryAlgorithm.h.

```
#include <gtest/gtest.h>
#include ¨../src/EvolutionaryAlgorithm.h¨
```

Include dependency graph for EvolutionaryAlgorithmTest.cc:


test/EvolutionaryAlgorithm
Test.cc
gtest/gtest.h
../src/EvolutionaryAlgorithm.h
deque
fstream
nlohmann/json.hpp
random
vector
../config/Config.h
../config/EvolutionaryAlgorithm
Config.h

Functions

- TEST (EvolutionaryAlgorithmTest, MutationTest)
- TEST (EvolutionaryAlgorithmTest, TournamentSelectionTest)
- TEST (EvolutionaryAlgorithmTest, NextGenerationTest)
- TEST (EvolutionaryAlgorithmTest, SaveToJsonTest)

4.41.1 Detailed Description

This file contains tests for functions from src/EvolutionaryAlgorithm.h.

Author

Mateusz Krakowski

Date

2023-06-06

Definition in file EvolutionaryAlgorithmTest.cc.

4.41.2 Function Documentation

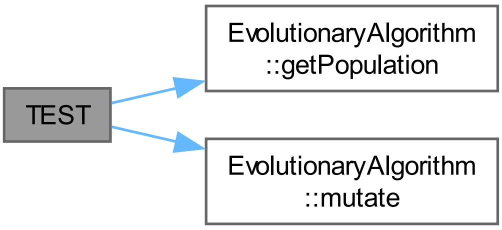
4.41.2.1 TEST() [1/4]

```
TEST (
          EvolutionaryAlgorithmTest ,
          MutationTest  )
```

Definition at line 12 of file EvolutionaryAlgorithmTest.cc.

```
00012                                                {
00013     EvolutionaryAlgorithm evo(10);
00014     // mutate the population 50 times just to test it
00015     for (int i = 0; i < 50; ++i) {
00016         evo.mutate();
00017     }
00018     std::vector<Chromosome> population = evo.getPopulation();
00019 
00020     // assert that all body lengths are within the range
00021     for (auto& chrom : population) {
00022         for (auto& length : chrom.bodyLengths) {
00023             ASSERT_GE(length, EvolutionaryAlgorithmConfig::MIN_BODY_LENGTH);
00024             ASSERT_LE(length, EvolutionaryAlgorithmConfig::MAX_BODY_LENGTH);
00025         }
00026     }
00027 }
```

Here is the call graph for this function:


TEST
EvolutionaryAlgorithm ::getPopulation
EvolutionaryAlgorithm ::mutate

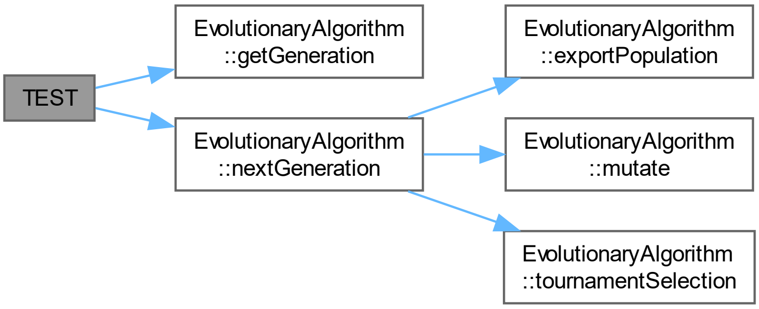
4.41.2.2 TEST() [2/4]

```
TEST (
          EvolutionaryAlgorithmTest ,
          NextGenerationTest  )
```

Definition at line 37 of file EvolutionaryAlgorithmTest.cc.

```
00037                                                      {
00038     EvolutionaryAlgorithm evo(10);
00039     evo.nextGeneration();
00040     for (int i = 0; i < 49; ++i) {
00041         evo.nextGeneration();
00042     }
00043 
00044     ASSERT_EQ(evo.getGeneration(), 50);
00045 }
```

Here is the call graph for this function:


TEST
EvolutionaryAlgorithm ::getGeneration
EvolutionaryAlgorithm ::nextGeneration
EvolutionaryAlgorithm ::exportPopulation
EvolutionaryAlgorithm ::mutate
EvolutionaryAlgorithm ::tournamentSelection

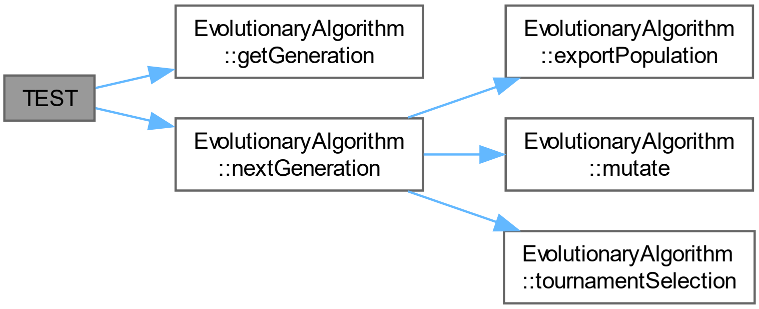
4.41.2.3 TEST() [3/4]

```
TEST (
            EvolutionaryAlgorithmTest ,
            SaveToJsonTest   )
```

Definition at line 47 of file EvolutionaryAlgorithmTest.cc.

```
00047                                              {
00048     EvolutionaryAlgorithm evo(2, true);
00049     evo.nextGeneration();
00050     for (int i = 0; i < 5; ++i) {
00051         evo.nextGeneration();
00052     }
00053 
00054     ASSERT_EQ(evo.getGeneration(), 6);
00055 }
```

Here is the call graph for this function:


TEST
EvolutionaryAlgorithm ::getGeneration
EvolutionaryAlgorithm ::nextGeneration
EvolutionaryAlgorithm ::exportPopulation
EvolutionaryAlgorithm ::mutate
EvolutionaryAlgorithm ::tournamentSelection

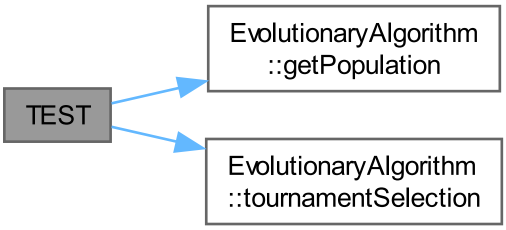
4.41.2.4 TEST() [4/4]

```
TEST (
            EvolutionaryAlgorithmTest ,
            TournamentSelectionTest  )
```

Definition at line 29 of file EvolutionaryAlgorithmTest.cc.

```
00029                                                   {
00030     EvolutionaryAlgorithm evo(10);
00031     evo.tournamentSelection();
00032     std::vector<Chromosome> population = evo.getPopulation();
00033 
00034     ASSERT_EQ(population.size(), 10);
00035 }
```

Here is the call graph for this function:


TEST
EvolutionaryAlgorithm ::getPopulation
EvolutionaryAlgorithm ::tournamentSelection

4.42 EvolutionaryAlgorithmTest.cc

Go to the documentation of this file.

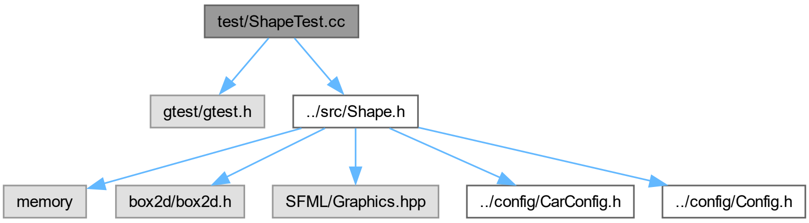
```
00001 
00009 #include <gtest/gtest.h>
00010 #include "../src/EvolutionaryAlgorithm.h"
00011 
00012 TEST(EvolutionaryAlgorithmTest, MutationTest) {
00013     EvolutionaryAlgorithm evo(10);
00014     // mutate the population 50 times just to test it
00015     for (int i = 0; i < 50; ++i) {
00016         evo.mutate();
00017     }
00018     std::vector<Chromosome> population = evo.getPopulation();
00019 
00020     // assert that all body lengths are within the range
00021     for (auto& chrom : population) {
00022         for (auto& length : chrom.bodyLengths) {
00023             ASSERT_GE(length, EvolutionaryAlgorithmConfig::MIN_BODY_LENGTH);
00024             ASSERT_LE(length, EvolutionaryAlgorithmConfig::MAX_BODY_LENGTH);
00025         }
00026     }
00027 }
00028 
00029 TEST(EvolutionaryAlgorithmTest, TournamentSelectionTest) {
00030     EvolutionaryAlgorithm evo(10);
00031     evo.tournamentSelection();
00032     std::vector<Chromosome> population = evo.getPopulation();
00033 
00034     ASSERT_EQ(population.size(), 10);
00035 }
00036 
00037 TEST(EvolutionaryAlgorithmTest, NextGenerationTest) {
00038     EvolutionaryAlgorithm evo(10);
00039     evo.nextGeneration();
00040     for (int i = 0; i < 49; ++i) {
        evo.nextGeneration();
    }

    ASSERT_EQ(evo.getGeneration(), 50);
}

TEST(EvolutionaryAlgorithmTest, SaveToJsonTest) {
    EvolutionaryAlgorithm evo(2, true);
    evo.nextGeneration();
    for (int i = 0; i < 5; ++i) {
        evo.nextGeneration();
    }

    ASSERT_EQ(evo.getGeneration(), 6);
}
```

4.43 test/ShapeTest.cc File Reference

This file contains tests for functions from src/Shape.h.

```
#include <gtest/gtest.h>
#include ¨../src/Shape.h¨
```

Include dependency graph for ShapeTest.cc:



Functions

- TEST (CreateBoxTest, BasicTest)
- TEST (CreateBoxTest, InvalidWidthTest)
- TEST (CreateBoxTest, InvalidHeightTest)
- TEST (CreateBoxTest, InvalidDensityTest)
- TEST (CreateBoxTest, InvalidFrictionTest)
- TEST (CreateGroundTest, BasicTest)
- TEST (CreateGroundTest, InvalidVerticesTest)
- TEST (CreateCircleTest, BasicTest)
- TEST (CreateCircleTest, InvalidRadiusTest)
- TEST (CreateCircleTest, InvalidDensityTest)
- TEST (CreateCircleTest, InvalidFrictionTest)
- TEST (CreatePolygonTest, BasicTest)
- TEST (CreatePolygonTest, EmptyVerticesTest)
- TEST (CreatePolygonTest, TooMuchVerticesTest)
- TEST (CreatePolygonTest, InvalidDensityTest)
- TEST (CreatePolygonTest, InvalidFrictionTest)

4.43.1 Detailed Description

This file contains tests for functions from src/Shape.h.

Author

Mateusz Krakowski

Date

2023-06-06

Definition in file ShapeTest.cc.

4.43.2 Function Documentation

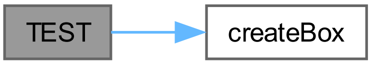
4.43.2.1 TEST() [1/16]

```
TEST (
            CreateBoxTest ,
            BasicTest   )
```

Definition at line 12 of file ShapeTest.cc.

```
00012                               {
00013     b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
00014     float x = 0.0f, y = 0.0f, width = 10.0f, height = 20.0f, density = 1.0f, friction = 0.5f;
00015     sf::Color color = sf::Color::Red;
00016 
00017     Box box = createBox(world, x, y, width, height, density, friction, color);
00018 
00019     EXPECT_EQ(box.body->GetPosition().x, x / Config::PPM);
00020     EXPECT_EQ(box.body->GetPosition().y, y / Config::PPM);
00021     EXPECT_EQ(box.width, width);
00022     EXPECT_EQ(box.height, height);
00023     EXPECT_EQ(box.color, color);
00024     EXPECT_EQ(box.body->GetFixtureList()->GetDensity(), density);
00025     EXPECT_EQ(box.body->GetFixtureList()->GetFriction(), friction);
00026 }
```

Here is the call graph for this function:



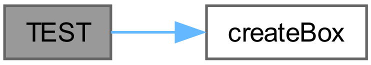
4.43.2.2 TEST() [2/16]

```
TEST (
            CreateBoxTest ,
            InvalidDensityTest )
```

Definition at line 48 of file ShapeTest.cc.

```
00048                                       {
00049     // Test with invalid input (negative width and height)
00050     b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
00051     float x = 1.0f, y = 1.0f, width = 10.0f, height = 1.0f, density = -1.0f, friction = 0.5f;
00052     sf::Color color = sf::Color::Red;
00053
00054     ASSERT_THROW(createBox(world, x, y, width, height, density, friction, color),
00055                  std::invalid_argument);
00056 }
```

Here is the call graph for this function:



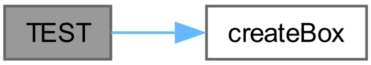
4.43.2.3 TEST() [3/16]

```
TEST (
            CreateBoxTest ,
            InvalidFrictionTest )
```

Definition at line 58 of file ShapeTest.cc.

```
00058                                        {
00059     // Test with invalid input (negative width and height)
00060     b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
00061     float x = 1.0f, y = 1.0f, width = 10.0f, height = 5.0f, density = 2.0f, friction = 0.0f;
00062     sf::Color color = sf::Color::Red;
00063
00064     ASSERT_THROW(createBox(world, x, y, width, height, density, friction, color),
00065                  std::invalid_argument);
00066 }
```

Here is the call graph for this function:



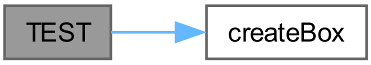
4.43.2.4 TEST() [4/16]

```
TEST (
            CreateBoxTest ,
            InvalidHeightTest  )
```

Definition at line 38 of file ShapeTest.cc.

```
                                    {
    // Test with invalid input (negative width and height)
    b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
    float x = 0.0f, y = 0.0f, width = 10.0f, height = 0.0f, density = 1.0f, friction = 0.5f;
    sf::Color color = sf::Color::Red;

    ASSERT_THROW(createBox(world, x, y, width, height, density, friction, color),
                 std::invalid_argument);
}
```

Here is the call graph for this function:



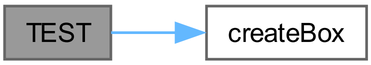
4.43.2.5 TEST() [5/16]

```
TEST (
            CreateBoxTest ,
            InvalidWidthTest  )
```

Definition at line 28 of file ShapeTest.cc.

```
                                   {
    // Test with invalid input (negative width and height)
    b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
    float x = 0.0f, y = 0.0f, width = -10.0f, height = 10.0f, density = 1.0f, friction = 0.5f;
    sf::Color color = sf::Color::Red;

    ASSERT_THROW(createBox(world, x, y, width, height, density, friction, color),
                 std::invalid_argument);
}
```

Here is the call graph for this function:



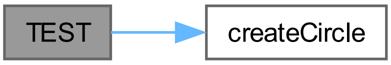
4.43.2.6 TEST() [6/16]

```
TEST (
            CreateCircleTest ,
            BasicTest  )
```

Definition at line 94 of file ShapeTest.cc.

```
00094                                  {
00095     b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
00096     float x = 10.0f, y = 20.0f, radius = 2.0f, density = 1.0f, friction = 0.5f;
00097     sf::Color color(255, 255, 0);
00098 
00099     Circle circle = createCircle(world, x, y, radius, density, friction, color);
00100 
00101     ASSERT_EQ(circle.radius, radius);
00102     ASSERT_EQ(circle.color, color);
00103     ASSERT_EQ(circle.body->GetType(), b2_dynamicBody);
00104     ASSERT_EQ(circle.body->GetPosition(), b2Vec2(x / Config::PPM, y / Config::PPM));
00105     ASSERT_EQ(circle.body->GetFixtureList()->GetDensity(), density);
00106     ASSERT_EQ(circle.body->GetFixtureList()->GetFriction(), friction);
00107 }
```

Here is the call graph for this function:



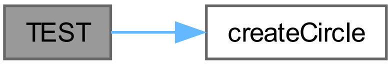
4.43.2.7 TEST() [7/16]

```
TEST (
            CreateCircleTest ,
            InvalidDensityTest  )
```

Definition at line 118 of file ShapeTest.cc.

```
00118                                          {
00119     b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
00120     float x = 10.0f, y = 20.0f, radius = 2.0f, density = -1.0f, friction = 0.5f;
00121     sf::Color color(255, 255, 0);
00122 
00123     ASSERT_THROW(createCircle(world, x, y, radius, density, friction, color),
00124                  std::invalid_argument);
00125 }
```

Here is the call graph for this function:



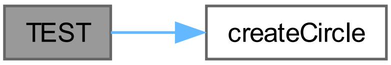
4.43.2.8 TEST() [8/16]

```
TEST (
            CreateCircleTest ,
            InvalidFrictionTest  )
```

Definition at line 127 of file ShapeTest.cc.

```
00127                                         {
00128     b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
00129     float x = 10.0f, y = 20.0f, radius = 2.0f, density = 1.0f, friction = -0.5f;
00130     sf::Color color(255, 255, 0);
00131
00132     ASSERT_THROW(createCircle(world, x, y, radius, density, friction, color),
00133                  std::invalid_argument);
00134 }
```

Here is the call graph for this function:



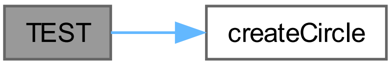
4.43.2.9 TEST() [9/16]

```
TEST (
            CreateCircleTest ,
            InvalidRadiusTest  )
```

Definition at line 109 of file ShapeTest.cc.

```
00109                                       {
00110     b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
00111     float x = 10.0f, y = 20.0f, radius = 0.0f, density = 1.0f, friction = 0.5f;
00112     sf::Color color(255, 255, 0);
00113
00114     ASSERT_THROW(createCircle(world, x, y, radius, density, friction, color),
00115                  std::invalid_argument);
00116 }
```

Here is the call graph for this function:



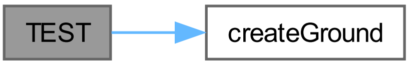
4.43.2.10 TEST() [10/16]

```
TEST (
            CreateGroundTest ,
            BasicTest   )
```

Definition at line 68 of file ShapeTest.cc.

```
00068                                   {
00069     b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
00070     float x = 42.0f, y = 42.0f;
00071     std::vector<b2Vec2> vertices = {b2Vec2(-25.0f, -5.0f), b2Vec2(25.0f, -5.0f),
00072                                     b2Vec2(25.0f, 5.0f), b2Vec2(-25.0f, 5.0f)};
00073     sf::Color color = sf::Color::Blue;
00074
00075     ASSERT_NO_THROW(Polygon ground = createGround(world, x, y, vertices, color));
00076
00077     Polygon ground = createGround(world, x, y, vertices, color);
00078     ASSERT_EQ(ground.vertices, vertices);
00079     ASSERT_EQ(ground.color, color);
00080     ASSERT_EQ(ground.body->GetType(), b2_staticBody);
00081     ASSERT_EQ(ground.body->GetFixtureList()->GetDensity(), 0.0f);
00082 }
```

Here is the call graph for this function:



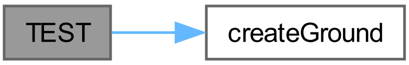
4.43.2.11 TEST() [11/16]

```
TEST (
            CreateGroundTest ,
            InvalidVerticesTest   )
```

Definition at line 84 of file ShapeTest.cc.

```
00084                                             {
00085     b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
00086     float x = 42.0f, y = 42.0f;
00087     std::vector<b2Vec2> invalidVertices;  // Empty vertices
00088
00089     sf::Color color = sf::Color::Blue;
00090
00091     ASSERT_THROW(createGround(world, x, y, invalidVertices, color), std::invalid_argument);
00092 }
```

Here is the call graph for this function:



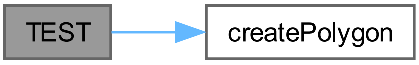
4.43.2.12 TEST() [12/16]

```
TEST (
              CreatePolygonTest ,
              BasicTest  )
```

Definition at line 136 of file ShapeTest.cc.

```
00136                                     {
00137     b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
00138     float x = 0.0f, y = 0.0f, density = 1.0f, friction = 0.5f;
00139     std::vector<b2Vec2> vertices = {b2Vec2(-1.0f, -1.0f), b2Vec2(1.0f, -1.0f), b2Vec2(0.0f, 1.0f)};
00140     sf::Color color = sf::Color::Red;
00141     Polygon polygon = createPolygon(world, x, y, vertices, density, friction, color);
00142 
00143     EXPECT_NE(nullptr, polygon.body);
00144 
00145     ASSERT_EQ(polygon.body->GetPosition(), b2Vec2(x / Config::PPM, y / Config::PPM));
00146 
00147     std::shared_ptr<const b2Fixture> fixture(polygon.body->GetFixtureList(),
00148                                               [](const b2Fixture* f) {});
00149     ASSERT_NE(nullptr, fixture);
00150     std::shared_ptr<const b2PolygonShape> shape(
00151         dynamic_cast<const b2PolygonShape*>(fixture->GetShape()), [](const b2PolygonShape* s) {});
00152     ASSERT_NE(nullptr, shape);
00153 
00154     EXPECT_EQ(density, fixture->GetDensity());
00155     EXPECT_EQ(friction, fixture->GetFriction());
00156 }
```

Here is the call graph for this function:



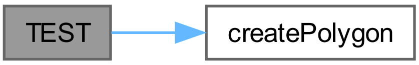
4.43.2.13 TEST() [13/16]

```
TEST (
              CreatePolygonTest ,
              EmptyVerticesTest  )
```

Definition at line 158 of file ShapeTest.cc.

```
00158                                             {
00159     b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
00160     float x = 0.0f, y = 0.0f, density = 1.0f, friction = 0.5f;
00161     std::vector<b2Vec2> vertices = {};
00162     sf::Color color = sf::Color::Red;
00163     ASSERT_THROW(createPolygon(world, x, y, vertices, density, friction, color);
00164                  , std::invalid_argument);
00165 }
```

Here is the call graph for this function:



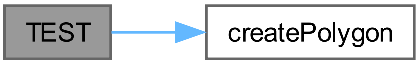
4.43.2.14 TEST() [14/16]

```
TEST (
            CreatePolygonTest ,
            InvalidDensityTest  )
```

Definition at line 178 of file ShapeTest.cc.

```
                                        {
    b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
    float x = 0.0f, y = 0.0f, density = 0.0f, friction = 0.5f;
    std::vector<b2Vec2> vertices = {b2Vec2(-1.0f, -1.0f), b2Vec2(1.0f, -1.0f), b2Vec2(0.0f, 1.0f),
                                    b2Vec2(-1.0f, -1.0f), b2Vec2(1.0f, -1.0f), b2Vec2(0.0f, 1.0f),
                                    b2Vec2(-1.0f, -1.0f), b2Vec2(1.0f, -1.0f)};
    sf::Color color = sf::Color::Red;
    ASSERT_THROW(createPolygon(world, x, y, vertices, density, friction, color);
                 , std::invalid_argument);
}
```

Here is the call graph for this function:



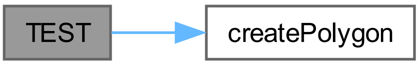
4.43.2.15 TEST() [15/16]

```
TEST (
            CreatePolygonTest ,
            InvalidFrictionTest  )
```

Definition at line 189 of file ShapeTest.cc.

```
                                         {
    b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
    float x = 0.0f, y = 0.0f, density = 1.0f, friction = -0.5f;
    std::vector<b2Vec2> vertices = {b2Vec2(-1.0f, -1.0f), b2Vec2(1.0f, -1.0f), b2Vec2(0.0f, 1.0f),
                                    b2Vec2(-1.0f, -1.0f), b2Vec2(1.0f, -1.0f), b2Vec2(0.0f, 1.0f),
                                    b2Vec2(-1.0f, -1.0f), b2Vec2(1.0f, -1.0f)};
    sf::Color color = sf::Color::Red;
    ASSERT_THROW(createPolygon(world, x, y, vertices, density, friction, color);
                 , std::invalid_argument);
}
```

Here is the call graph for this function:



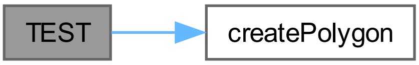
4.43.2.16 TEST() [16/16]

```
TEST (
            CreatePolygonTest ,
            TooMuchVerticesTest  )
```

Definition at line 167 of file ShapeTest.cc.

```
00167                                           {
00168     b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
00169     float x = 0.0f, y = 0.0f, density = 1.0f, friction = 0.5f;
00170     std::vector<b2Vec2> vertices = {b2Vec2(-1.0f, -1.0f), b2Vec2(1.0f, -1.0f), b2Vec2(0.0f, 1.0f),
00171                                     b2Vec2(-1.0f, -1.0f), b2Vec2(1.0f, -1.0f), b2Vec2(0.0f, 1.0f),
00172                                     b2Vec2(-1.0f, -1.0f), b2Vec2(1.0f, -1.0f), b2Vec2(0.0f, 1.0f)};
00173     sf::Color color = sf::Color::Red;
00174     ASSERT_THROW(createPolygon(world, x, y, vertices, density, friction, color);
00175                  , std::invalid_argument);
00176 }
```

Here is the call graph for this function:



4.44 ShapeTest.cc

Go to the documentation of this file.

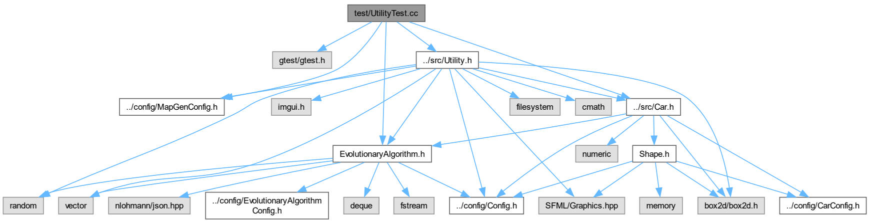
```

#include <gtest/gtest.h>
#include "../src/Shape.h"

TEST(CreateBoxTest, BasicTest) {
    b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
    float x = 0.0f, y = 0.0f, width = 10.0f, height = 20.0f, density = 1.0f, friction = 0.5f;
    sf::Color color = sf::Color::Red;

    Box box = createBox(world, x, y, width, height, density, friction, color);

    EXPECT_EQ(box.body->GetPosition().x, x / Config::PPM);
    EXPECT_EQ(box.body->GetPosition().y, y / Config::PPM);
    EXPECT_EQ(box.width, width);
    EXPECT_EQ(box.height, height);
    EXPECT_EQ(box.color, color);
    EXPECT_EQ(box.body->GetFixtureList()->GetDensity(), density);
    EXPECT_EQ(box.body->GetFixtureList()->GetFriction(), friction);
}

TEST(CreateBoxTest, InvalidWidthTest) {
    // Test with invalid input (negative width and height)
    b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
    float x = 0.0f, y = 0.0f, width = -10.0f, height = 10.0f, density = 1.0f, friction = 0.5f;
    sf::Color color = sf::Color::Red;

    ASSERT_THROW(createBox(world, x, y, width, height, density, friction, color),
                 std::invalid_argument);
}

TEST(CreateBoxTest, InvalidHeightTest) {
    // Test with invalid input (negative width and height)
    b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
    float x = 0.0f, y = 0.0f, width = 10.0f, height = 0.0f, density = 1.0f, friction = 0.5f;
    sf::Color color = sf::Color::Red;

    ASSERT_THROW(createBox(world, x, y, width, height, density, friction, color),
                 std::invalid_argument);
```

```
}

TEST(CreateBoxTest, InvalidDensityTest) {
    // Test with invalid input (negative width and height)
    b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
    float x = 1.0f, y = 1.0f, width = 10.0f, height = 1.0f, density = -1.0f, friction = 0.5f;
    sf::Color color = sf::Color::Red;

    ASSERT_THROW(createBox(world, x, y, width, height, density, friction, color),
                 std::invalid_argument);
}

TEST(CreateBoxTest, InvalidFrictionTest) {
    // Test with invalid input (negative width and height)
    b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
    float x = 1.0f, y = 1.0f, width = 10.0f, height = 5.0f, density = 2.0f, friction = 0.0f;
    sf::Color color = sf::Color::Red;

    ASSERT_THROW(createBox(world, x, y, width, height, density, friction, color),
                 std::invalid_argument);
}

TEST(CreateGroundTest, BasicTest) {
    b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
    float x = 42.0f, y = 42.0f;
    std::vector<b2Vec2> vertices = {b2Vec2(-25.0f, -5.0f), b2Vec2(25.0f, -5.0f),
                                    b2Vec2(25.0f, 5.0f), b2Vec2(-25.0f, 5.0f)};
    sf::Color color = sf::Color::Blue;

    ASSERT_NO_THROW(Polygon ground = createGround(world, x, y, vertices, color));

    Polygon ground = createGround(world, x, y, vertices, color);
    ASSERT_EQ(ground.vertices, vertices);
    ASSERT_EQ(ground.color, color);
    ASSERT_EQ(ground.body->GetType(), b2_staticBody);
    ASSERT_EQ(ground.body->GetFixtureList()->GetDensity(), 0.0f);
}

TEST(CreateGroundTest, InvalidVerticesTest) {
    b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
    float x = 42.0f, y = 42.0f;
    std::vector<b2Vec2> invalidVertices;  // Empty vertices

    sf::Color color = sf::Color::Blue;

    ASSERT_THROW(createGround(world, x, y, invalidVertices, color), std::invalid_argument);
}

TEST(CreateCircleTest, BasicTest) {
    b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
    float x = 10.0f, y = 20.0f, radius = 2.0f, density = 1.0f, friction = 0.5f;
    sf::Color color(255, 255, 0);

    Circle circle = createCircle(world, x, y, radius, density, friction, color);

    ASSERT_EQ(circle.radius, radius);
    ASSERT_EQ(circle.color, color);
    ASSERT_EQ(circle.body->GetType(), b2_dynamicBody);
    ASSERT_EQ(circle.body->GetPosition(), b2Vec2(x / Config::PPM, y / Config::PPM));
    ASSERT_EQ(circle.body->GetFixtureList()->GetDensity(), density);
    ASSERT_EQ(circle.body->GetFixtureList()->GetFriction(), friction);
}

TEST(CreateCircleTest, InvalidRadiusTest) {
    b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
    float x = 10.0f, y = 20.0f, radius = 0.0f, density = 1.0f, friction = 0.5f;
    sf::Color color(255, 255, 0);

    ASSERT_THROW(createCircle(world, x, y, radius, density, friction, color),
                 std::invalid_argument);
}

TEST(CreateCircleTest, InvalidDensityTest) {
    b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
    float x = 10.0f, y = 20.0f, radius = 2.0f, density = -1.0f, friction = 0.5f;
    sf::Color color(255, 255, 0);

    ASSERT_THROW(createCircle(world, x, y, radius, density, friction, color),
                 std::invalid_argument);
}

TEST(CreateCircleTest, InvalidFrictionTest) {
    b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
    float x = 10.0f, y = 20.0f, radius = 2.0f, density = 1.0f, friction = -0.5f;
    sf::Color color(255, 255, 0);

    ASSERT_THROW(createCircle(world, x, y, radius, density, friction, color),
                 std::invalid_argument);
}

TEST(CreatePolygonTest, BasicTest) {
    b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
    float x = 0.0f, y = 0.0f, density = 1.0f, friction = 0.5f;
    std::vector<b2Vec2> vertices = {b2Vec2(-1.0f, -1.0f), b2Vec2(1.0f, -1.0f), b2Vec2(0.0f, 1.0f)};
    sf::Color color = sf::Color::Red;
    Polygon polygon = createPolygon(world, x, y, vertices, density, friction, color);

    EXPECT_NE(nullptr, polygon.body);

    ASSERT_EQ(polygon.body->GetPosition(), b2Vec2(x / Config::PPM, y / Config::PPM));

    std::shared_ptr<const b2Fixture> fixture(polygon.body->GetFixtureList(),
                                             [](const b2Fixture* f) {});
    ASSERT_NE(nullptr, fixture);
    std::shared_ptr<const b2PolygonShape> shape(
        dynamic_cast<const b2PolygonShape*>(fixture->GetShape()), [](const b2PolygonShape* s) {});
    ASSERT_NE(nullptr, shape);

    EXPECT_EQ(density, fixture->GetDensity());
    EXPECT_EQ(friction, fixture->GetFriction());
}

TEST(CreatePolygonTest, EmptyVerticesTest) {
    b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
    float x = 0.0f, y = 0.0f, density = 1.0f, friction = 0.5f;
    std::vector<b2Vec2> vertices = {};
    sf::Color color = sf::Color::Red;
    ASSERT_THROW(createPolygon(world, x, y, vertices, density, friction, color);
                 , std::invalid_argument);
}

TEST(CreatePolygonTest, TooMuchVerticesTest) {
    b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
    float x = 0.0f, y = 0.0f, density = 1.0f, friction = 0.5f;
    std::vector<b2Vec2> vertices = {b2Vec2(-1.0f, -1.0f), b2Vec2(1.0f, -1.0f), b2Vec2(0.0f, 1.0f),
                                    b2Vec2(-1.0f, -1.0f), b2Vec2(1.0f, -1.0f), b2Vec2(0.0f, 1.0f),
                                    b2Vec2(-1.0f, -1.0f), b2Vec2(1.0f, -1.0f), b2Vec2(0.0f, 1.0f)};
    sf::Color color = sf::Color::Red;
    ASSERT_THROW(createPolygon(world, x, y, vertices, density, friction, color);
                 , std::invalid_argument);
}

TEST(CreatePolygonTest, InvalidDensityTest) {
    b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
    float x = 0.0f, y = 0.0f, density = 0.0f, friction = 0.5f;
    std::vector<b2Vec2> vertices = {b2Vec2(-1.0f, -1.0f), b2Vec2(1.0f, -1.0f), b2Vec2(0.0f, 1.0f),
                                    b2Vec2(-1.0f, -1.0f), b2Vec2(1.0f, -1.0f), b2Vec2(0.0f, 1.0f),
                                    b2Vec2(-1.0f, -1.0f), b2Vec2(1.0f, -1.0f)};
    sf::Color color = sf::Color::Red;
    ASSERT_THROW(createPolygon(world, x, y, vertices, density, friction, color);
                 , std::invalid_argument);
}

TEST(CreatePolygonTest, InvalidFrictionTest) {
    b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
    float x = 0.0f, y = 0.0f, density = 1.0f, friction = -0.5f;
    std::vector<b2Vec2> vertices = {b2Vec2(-1.0f, -1.0f), b2Vec2(1.0f, -1.0f), b2Vec2(0.0f, 1.0f),
                                    b2Vec2(-1.0f, -1.0f), b2Vec2(1.0f, -1.0f), b2Vec2(0.0f, 1.0f),
                                    b2Vec2(-1.0f, -1.0f), b2Vec2(1.0f, -1.0f)};
    sf::Color color = sf::Color::Red;
    ASSERT_THROW(createPolygon(world, x, y, vertices, density, friction, color);
                 , std::invalid_argument);
}
```

4.45 test/UtilityTest.cc File Reference

```
#include <gtest/gtest.h>
#include ¨../config/MapGenConfig.h¨
#include ¨../src/Car.h¨
#include ¨../src/EvolutionaryAlgorithm.h¨
#include ¨../src/Utility.h¨
```

Include dependency graph for UtilityTest.cc:



Functions

- TEST (UtilityTest, applyAirResistanceTest)
- TEST (UtilityTest, generateGroundTest)
- TEST (UtilityTest, getNextGroundPartDegreeTest)
- TEST (UtilityTest, generateCarTest)
- TEST (UtilityTest, getFurthestCarPosTest)
- TEST (UtilityTest, SFMLColorToImVec4)
- TEST (UtilityTest, getIdxOfGrndClstToLocTest)
- TEST (UtilityTest, removeCarsTest)
- TEST (UtilityTest, getRootDirTest)
- TEST (UtilityTest, setIconTest)

4.45.1 Function Documentation

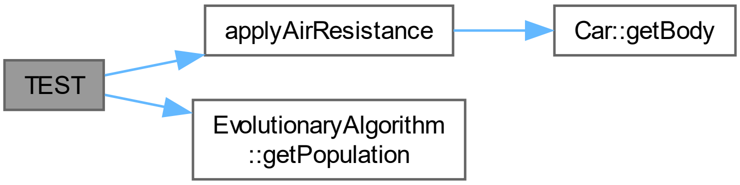
4.45.1.1 TEST() [1/10]

```
TEST (
              UtilityTest ,
              applyAirResistanceTest   )
```

Definition at line 16 of file UtilityTest.cc.

```
00016                                           {
00017     b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
00018     float x = 0.0f, y = 0.0f;
00019     sf::Color bodyColor = sf::Color::Red;
00020     sf::Color wheelColor = sf::Color::Blue;
00021
00022     EvolutionaryAlgorithm ea = EvolutionaryAlgorithm(EvolutionaryAlgorithmConfig::POPULATION_SIZE);
00023     Chromosome chromosome = ea.getPopulation()[0];
00024     Car car = Car(world, x, y, chromosome, bodyColor, wheelColor);
00025
00026     EXPECT_NO_THROW(applyAirResistance(car));
00027 }
```

Here is the call graph for this function:



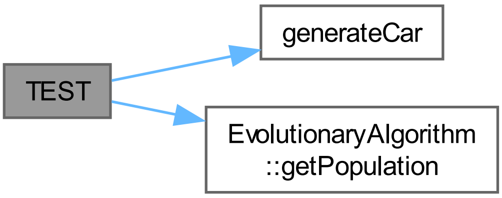
4.45.1.2 TEST() [2/10]

```
TEST (
            UtilityTest ,
            generateCarTest   )
```

Definition at line 60 of file UtilityTest.cc.

```
00060                                  {
00061     b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
00062     float x = 0.0f, y = 0.0f;
00063     sf::Color bodyColor = sf::Color::Red;
00064     sf::Color wheelColor = sf::Color::Blue;
00065 
00066     EvolutionaryAlgorithm ea = EvolutionaryAlgorithm(EvolutionaryAlgorithmConfig::POPULATION_SIZE);
00067     Chromosome chromosome = ea.getPopulation()[0];
00068 
00069     EXPECT_NO_THROW(generateCar(world, chromosome));
00070 }
```

Here is the call graph for this function:



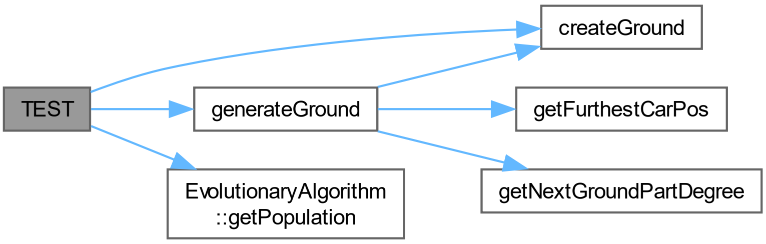
4.45.1.3 TEST() [3/10]

```
TEST (
            UtilityTest ,
            generateGroundTest   )
```

Definition at line 29 of file UtilityTest.cc.

```
00029                                     {
00030     b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
00031     float x = 0.0f, y = 0.0f;
00032     sf::Color bodyColor = sf::Color::Red;
00033     sf::Color wheelColor = sf::Color::Blue;
00034 
00035     EvolutionaryAlgorithm ea = EvolutionaryAlgorithm(EvolutionaryAlgorithmConfig::POPULATION_SIZE);
00036     Chromosome chromosome = ea.getPopulation()[0];
00037     std::vector<Car> cars;
00038 
00039     Car car = Car(world, x, y, chromosome, bodyColor, wheelColor);
00040     cars.push_back(car);
00041 
00042     std::vector<b2Vec2> groundVertecies = {
00043         b2Vec2(MapGenConfig::GROUND_STARTING_X, MapGenConfig::GROUND_STARTING_Y),
00044         b2Vec2(MapGenConfig::GROUND_STARTING_X + MapGenConfig::GROUND_LEG_LENGTH,
00045                MapGenConfig::GROUND_STARTING_Y),
00046         b2Vec2(MapGenConfig::GROUND_STARTING_X + MapGenConfig::GROUND_LEG_LENGTH,
00047                MapGenConfig::GROUND_STARTING_Y + MapGenConfig::GROUND_PART_LENGTH)};
00048 
00049     std::vector<Polygon> groundVector = {createGround(world, MapGenConfig::GROUND_STARTING_X,
00050                                                       MapGenConfig::GROUND_STARTING_Y,
00051                                                       groundVertecies, sf::Color(18, 36, 35))};
00052 
00053     EXPECT_NO_THROW(generateGround(world, &groundVector, cars));
00054
00055     EXPECT_EQ(groundVector.size(), 2);
00056 }
```

Here is the call graph for this function:



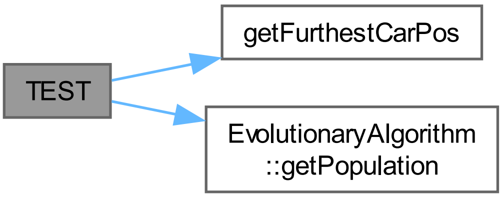
4.45.1.4 TEST() [4/10]

```
TEST (
          UtilityTest ,
          getFurthestCarPosTest  )
```

Definition at line 72 of file UtilityTest.cc.

```
00072                                           {
00073     b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
00074     float x = 0.0f, y = 0.0f;
00075     sf::Color bodyColor = sf::Color::Red;
00076     sf::Color wheelColor = sf::Color::Blue;
00077
00078     EvolutionaryAlgorithm ea = EvolutionaryAlgorithm(EvolutionaryAlgorithmConfig::POPULATION_SIZE);
00079     Chromosome chromosome = ea.getPopulation()[0];
00080     std::vector<Car> cars;
00081
00082     Car car = Car(world, x, y, chromosome, bodyColor, wheelColor);
00083     cars.push_back(car);
00084
00085     EXPECT_NO_THROW(getFurthestCarPos(cars));
00086     EXPECT_EQ(getFurthestCarPos(cars).x, x);
00087     EXPECT_EQ(getFurthestCarPos(cars).y, y);
00088 }
```

Here is the call graph for this function:



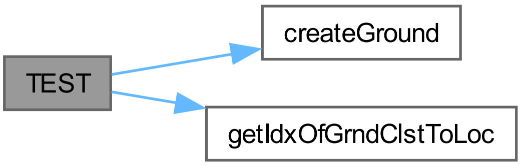
4.45.1.5 TEST() [5/10]

```
TEST (
            UtilityTest ,
            getIdxOfGrndClstToLocTest  )
```

Definition at line 100 of file UtilityTest.cc.

```
00100                                          {
00101     b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
00102     std::vector<b2Vec2> groundVertecies = {
00103         b2Vec2(MapGenConfig::GROUND_STARTING_X, MapGenConfig::GROUND_STARTING_Y),
00104         b2Vec2(MapGenConfig::GROUND_STARTING_X + MapGenConfig::GROUND_LEG_LENGTH,
00105                MapGenConfig::GROUND_STARTING_Y),
00106         b2Vec2(MapGenConfig::GROUND_STARTING_X + MapGenConfig::GROUND_LEG_LENGTH,
00107                MapGenConfig::GROUND_STARTING_Y + MapGenConfig::GROUND_PART_LENGTH)};
00108
00109     Polygon ground =
00110         createGround(world, MapGenConfig::GROUND_STARTING_X, MapGenConfig::GROUND_STARTING_Y,
00111                      groundVertecies, sf::Color(18, 36, 35));
00112
00113     std::vector<Polygon> groundVector;
00114     groundVector.push_back(ground);
00115
00116     EXPECT_NO_THROW(getIdxOfGrndClstToLoc(groundVector, 0.0f));
00117     EXPECT_EQ(getIdxOfGrndClstToLoc(groundVector, 0.0f), 0);
00118 }
```

Here is the call graph for this function:



4.45.1.6 TEST() [6/10]

```
TEST (
            UtilityTest ,
            getNextGroundPartDegreeTest  )
```

Definition at line 58 of file UtilityTest.cc.

```
00058 { EXPECT_NO_THROW(getNextGroundPartDegree()); }
```

Here is the call graph for this function:

TEST
getNextGroundPartDegree

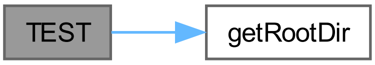
4.45.1.7 TEST() [7/10]

```
TEST (
            UtilityTest ,
            getRootDirTest  )
```

Definition at line 137 of file UtilityTest.cc.

```
00137 { EXPECT_NO_THROW(getRootDir()); }
```

Here is the call graph for this function:



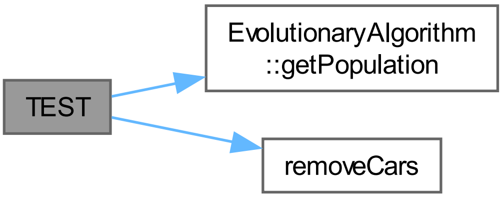
4.45.1.8 TEST() [8/10]

```
TEST (
            UtilityTest ,
            removeCarsTest  )
```

Definition at line 120 of file UtilityTest.cc.

```
00120                                       {
00121     b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
00122     float x = 0.0f, y = 0.0f;
00123     sf::Color bodyColor = sf::Color::Red;
00124     sf::Color wheelColor = sf::Color::Blue;
00125 
00126     EvolutionaryAlgorithm ea = EvolutionaryAlgorithm(EvolutionaryAlgorithmConfig::POPULATION_SIZE);
00127     Chromosome chromosome = ea.getPopulation()[0];
00128     std::vector<Car> cars;
00129 
00130     Car car = Car(world, x, y, chromosome, bodyColor, wheelColor);
00131     cars.push_back(car);
00132 
00133     EXPECT_NO_THROW(removeCars(world, &cars));
00134     EXPECT_EQ(cars.size(), 0);
00135 }
```

Here is the call graph for this function:



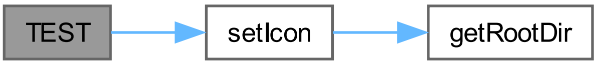
4.45.1.9 TEST() [9/10]

```
TEST (
              UtilityTest ,
              setIconTest  )
```

Definition at line 139 of file UtilityTest.cc.

```
00139                                    {
00140     sf::RenderWindow window(sf::VideoMode(800, 600), "SFML window");
00141     EXPECT_NO_THROW(setIcon(window));
00142 }
```

Here is the call graph for this function:



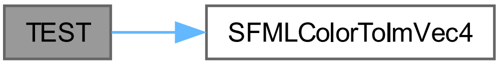
4.45.1.10 TEST() [10/10]

```
TEST (
              UtilityTest ,
              SFMLColorToImVec4  )
```

Definition at line 90 of file UtilityTest.cc.

```
00090                                         {
00091     sf::Color color = sf::Color::Red;
00092     ImVec4 imVec4 = SFMLColorToImVec4(color);
00093 
00094     EXPECT_EQ(imVec4.x, 1.0f);
00095     EXPECT_EQ(imVec4.y, 0.0f);
00096     EXPECT_EQ(imVec4.z, 0.0f);
00097     EXPECT_EQ(imVec4.w, 1.0f);
00098 }
```

Here is the call graph for this function:



4.46 UtilityTest.cc

Go to the documentation of this file.

```

#include <gtest/gtest.h>

#include "../config/MapGenConfig.h"
#include "../src/Car.h"
#include "../src/EvolutionaryAlgorithm.h"
#include "../src/Utility.h"

TEST(UtilityTest, applyAirResistanceTest) {
    b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
    float x = 0.0f, y = 0.0f;
    sf::Color bodyColor = sf::Color::Red;
    sf::Color wheelColor = sf::Color::Blue;

    EvolutionaryAlgorithm ea = EvolutionaryAlgorithm(EvolutionaryAlgorithmConfig::POPULATION_SIZE);
    Chromosome chromosome = ea.getPopulation()[0];
    Car car = Car(world, x, y, chromosome, bodyColor, wheelColor);

    EXPECT_NO_THROW(applyAirResistance(car));
}

TEST(UtilityTest, generateGroundTest) {
    b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
    float x = 0.0f, y = 0.0f;
    sf::Color bodyColor = sf::Color::Red;
    sf::Color wheelColor = sf::Color::Blue;

    EvolutionaryAlgorithm ea = EvolutionaryAlgorithm(EvolutionaryAlgorithmConfig::POPULATION_SIZE);
    Chromosome chromosome = ea.getPopulation()[0];
    std::vector<Car> cars;

    Car car = Car(world, x, y, chromosome, bodyColor, wheelColor);
    cars.push_back(car);

    std::vector<b2Vec2> groundVertecies = {
        b2Vec2(MapGenConfig::GROUND_STARTING_X, MapGenConfig::GROUND_STARTING_Y),
        b2Vec2(MapGenConfig::GROUND_STARTING_X + MapGenConfig::GROUND_LEG_LENGTH,
               MapGenConfig::GROUND_STARTING_Y),
        b2Vec2(MapGenConfig::GROUND_STARTING_X + MapGenConfig::GROUND_LEG_LENGTH,
               MapGenConfig::GROUND_STARTING_Y + MapGenConfig::GROUND_PART_LENGTH)};

    std::vector<Polygon> groundVector = {createGround(world, MapGenConfig::GROUND_STARTING_X,
                                                      MapGenConfig::GROUND_STARTING_Y,
                                                      groundVertecies, sf::Color(18, 36, 35))};

    EXPECT_NO_THROW(generateGround(world, &groundVector, cars));

    EXPECT_EQ(groundVector.size(), 2);
}

TEST(UtilityTest, getNextGroundPartDegreeTest) { EXPECT_NO_THROW(getNextGroundPartDegree()); }

TEST(UtilityTest, generateCarTest) {
    b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
    float x = 0.0f, y = 0.0f;
    sf::Color bodyColor = sf::Color::Red;
    sf::Color wheelColor = sf::Color::Blue;

    EvolutionaryAlgorithm ea = EvolutionaryAlgorithm(EvolutionaryAlgorithmConfig::POPULATION_SIZE);
    Chromosome chromosome = ea.getPopulation()[0];

    EXPECT_NO_THROW(generateCar(world, chromosome));
}

TEST(UtilityTest, getFurthestCarPosTest) {
    b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
    float x = 0.0f, y = 0.0f;
    sf::Color bodyColor = sf::Color::Red;
    sf::Color wheelColor = sf::Color::Blue;

    EvolutionaryAlgorithm ea = EvolutionaryAlgorithm(EvolutionaryAlgorithmConfig::POPULATION_SIZE);
    Chromosome chromosome = ea.getPopulation()[0];
    std::vector<Car> cars;

    Car car = Car(world, x, y, chromosome, bodyColor, wheelColor);
    cars.push_back(car);

    EXPECT_NO_THROW(getFurthestCarPos(cars));
    EXPECT_EQ(getFurthestCarPos(cars).x, x);
    EXPECT_EQ(getFurthestCarPos(cars).y, y);
}

```

```
TEST(UtilityTest, SFMLColorToImVec4) {
    sf::Color color = sf::Color::Red;
    ImVec4 imVec4 = SFMLColorToImVec4(color);

    EXPECT_EQ(imVec4.x, 1.0f);
    EXPECT_EQ(imVec4.y, 0.0f);
    EXPECT_EQ(imVec4.z, 0.0f);
    EXPECT_EQ(imVec4.w, 1.0f);
}

TEST(UtilityTest, getIdxOfGrndClstToLocTest) {
    b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
    std::vector<b2Vec2> groundVertecies = {
        b2Vec2(MapGenConfig::GROUND_STARTING_X, MapGenConfig::GROUND_STARTING_Y),
        b2Vec2(MapGenConfig::GROUND_STARTING_X + MapGenConfig::GROUND_LEG_LENGTH,
               MapGenConfig::GROUND_STARTING_Y),
        b2Vec2(MapGenConfig::GROUND_STARTING_X + MapGenConfig::GROUND_LEG_LENGTH,
               MapGenConfig::GROUND_STARTING_Y + MapGenConfig::GROUND_PART_LENGTH)};

    Polygon ground =
        createGround(world, MapGenConfig::GROUND_STARTING_X, MapGenConfig::GROUND_STARTING_Y,
                     groundVertecies, sf::Color(18, 36, 35));

    std::vector<Polygon> groundVector;
    groundVector.push_back(ground);

    EXPECT_NO_THROW(getIdxOfGrndClstToLoc(groundVector, 0.0f));
    EXPECT_EQ(getIdxOfGrndClstToLoc(groundVector, 0.0f), 0);
}

TEST(UtilityTest, removeCarsTest) {
    b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
    float x = 0.0f, y = 0.0f;
    sf::Color bodyColor = sf::Color::Red;
    sf::Color wheelColor = sf::Color::Blue;

    EvolutionaryAlgorithm ea = EvolutionaryAlgorithm(EvolutionaryAlgorithmConfig::POPULATION_SIZE);
    Chromosome chromosome = ea.getPopulation()[0];
    std::vector<Car> cars;

    Car car = Car(world, x, y, chromosome, bodyColor, wheelColor);
    cars.push_back(car);

    EXPECT_NO_THROW(removeCars(world, &cars));
    EXPECT_EQ(cars.size(), 0);
}

TEST(UtilityTest, getRootDirTest) { EXPECT_NO_THROW(getRootDir()); }

TEST(UtilityTest, setIconTest) {
    sf::RenderWindow window(sf::VideoMode(800, 600), "SFML window");
    EXPECT_NO_THROW(setIcon(window));
}
```

Index

applyAirResistance
- Utility.cc, 117
- Utility.h, 132

b2WorldPtr
- Car.h, 52
- Main.cc, 71
- Shape.h, 102
- Utility.h, 132

BACK_WHEEL_POS
- Config, 21

BG_SPRITES_COUNT
- MapGenConfig, 38

body
- Box, 6
- Circle, 19
- Polygon, 41

bodyDensity
- Chromosome, 18

bodyLengths
- Chromosome, 18

Box, 5
- body, 6
- color, 6
- height, 6
- width, 6

Car, 7
- Car, 8
- getBackWheel, 10
- getBody, 10
- getBodyColor, 10
- getFrontWheel, 11
- getPosX, 11
- getPosXVec, 12
- getPosY, 12
- getPosYVec, 12
- getVelocity, 13
- getVelocityVec, 13
- getVelX, 13
- getVelY, 14
- setCollisionFilter, 14

Car.cc
- createVertices, 49

Car.h
- b2WorldPtr, 52
- createVertices, 52

CAR_STARTING_X
- MapGenConfig, 38

CAR_STARTING_Y
- MapGenConfig, 38

CAR_TORQUE
- CarConfig, 16

CAR_VERTICES
- CarConfig, 16

CarConfig, 15
- CAR_TORQUE, 16
- CAR_VERTICES, 16
- MAX_JOINT_LENGTH, 16

CarTest.cc
- TEST, 145

CATEGORY_BITS
- Config, 21

Chromosome, 17
- bodyDensity, 18
- bodyLengths, 18
- fitness, 18
- wheelDensity, 18
- wheelRadius, 18

Circle, 19
- body, 19
- color, 19
- radius, 20

color
- Box, 6
- Circle, 19
- Polygon, 41

Config, 20
- BACK_WHEEL_POS, 21
- CATEGORY_BITS, 21
- DEBUG, 21
- DEG_PER_RAD, 21
- FRICTION, 21
- FRONT_WHEEL_POS, 21
- GENERATION_TIME, 21
- GRAVITATIONAL_ACCELERATION, 22
- GROUND_PARTS_RENDERED, 22
- MASK_BITS, 22
- MAX_FPS, 22
- PI, 22
- PPM, 22
- SAVE_FILE_NAME, 22
- SAVE_TO_FILE, 23
- VELOCITY_ARRAY_SIZE, 23
- WINDOW_HEIGHT, 23
- WINDOW_WIDTH, 23

config/CarConfig.h, 43, 44

config/Config.h, 44, 45

config/EvolutionaryAlgorithmConfig.h, 45, 46

config/MapGenConfig.h, 47, 48
createBox
    Shape.cc, 94
    Shape.h, 102
createCircle
    Shape.cc, 95
    Shape.h, 104
createGround
    Shape.cc, 96
    Shape.h, 105
createPolygon
    Shape.cc, 97
    Shape.h, 106
createVertices
    Car.cc, 49
    Car.h, 52

DEBUG
    Config, 21
DEG_PER_RAD
    Config, 21

EvolutionaryAlgorithm, 24
    EvolutionaryAlgorithm, 25
    exportPopulation, 25
    getGeneration, 26
    getPopulation, 26
    getPopulationSize, 27
    mutate, 27
    nextGeneration, 29
    setFitness, 30
    tournamentSelection, 30
EvolutionaryAlgorithmConfig, 31
    INITIAL_BODY_DENSITY_MEAN, 33
    INITIAL_BODY_DENSITY_VARIANCE, 33
    INITIAL_BODY_LENGTH_MEAN, 33
    INITIAL_BODY_LENGTH_VARIANCE, 33
    INITIAL_WHEEL_DENSITY_MEAN, 34
    INITIAL_WHEEL_DENSITY_VARIANCE, 34
    INITIAL_WHEEL_RADIUS_MEAN, 34
    INITIAL_WHEEL_RADIUS_VARIANCE, 34
    MAX_BODY_DENSITY, 34
    MAX_BODY_LENGTH, 34
    MAX_WHEEL_DENSITY, 34
    MAX_WHEEL_RADIUS, 35
    MIN_BODY_DENSITY, 35
    MIN_BODY_LENGTH, 35
    MIN_WHEEL_DENSITY, 35
    MIN_WHEEL_RADIUS, 35
    MUTATION_FACTOR_BODY_DENSITY, 35
    MUTATION_FACTOR_BODY_LENGTHS, 35
    MUTATION_FACTOR_WHEEL_DENSITY, 36
    MUTATION_FACTOR_WHEEL_RADIUS, 36
    MUTATION_RATE_BODY_DENSITY, 36
    MUTATION_RATE_BODY_LENGTHS, 36
    MUTATION_RATE_WHEEL_DENSITY, 36
    MUTATION_RATE_WHEEL_RADIUS, 36
    POPULATION_SIZE, 36
    TOURNAMENT_SIZE, 37
EvolutionaryAlgorithmTest.cc
    TEST, 148, 149
exportPopulation
    EvolutionaryAlgorithm, 25

fitness
    Chromosome, 18
FRICTION
    Config, 21
FRONT_WHEEL_POS
    Config, 21

GENERATE_DISTANCE
    MapGenConfig, 38
generateCar
    Utility.cc, 118
    Utility.h, 133
generateGround
    Utility.cc, 119
    Utility.h, 134
GENERATION_TIME
    Config, 21
getBackWheel
    Car, 10
getBody
    Car, 10
getBodyColor
    Car, 10
getFrontWheel
    Car, 11
getFurthestCarPos
    Utility.cc, 120
    Utility.h, 135
getGeneration
    EvolutionaryAlgorithm, 26
getIdxOfGrndClstToLoc
    Utility.cc, 121
    Utility.h, 136
getNextGroundPartDegree
    Utility.cc, 122
    Utility.h, 137
getPopulation
    EvolutionaryAlgorithm, 26
getPopulationSize
    EvolutionaryAlgorithm, 27
getPosX
    Car, 11
getPosXVec
    Car, 12
getPosY
    Car, 12
getPosYVec
    Car, 12
getRootDir
    Utility.cc, 122
    Utility.h, 137
getVelocity
    Car, 13
getVelocityVec

Car, 13
getVelX
Car, 13
getVelY
Car, 14
GRAVITATIONAL_ACCELERATION
Config, 22
GROUND_DEGREE_DEVIATION
MapGenConfig, 38
GROUND_LEG_LENGTH
MapGenConfig, 38
GROUND_PART_LENGTH
MapGenConfig, 39
GROUND_PARTS_RENDERED
Config, 22
GROUND_STARTING_X
MapGenConfig, 39
GROUND_STARTING_Y
MapGenConfig, 39
GUI.cc
printEAInfo, 60
renderPositionPlot, 60
renderVelocityPlot, 61
GUI.h
printEAInfo, 66
renderPositionPlot, 66
renderVelocityPlot, 67

handleEvents
UserInput.cc, 110
UserInput.h, 114
handleUserInput
UserInput.cc, 110
UserInput.h, 114
height
Box, 6

INITIAL_BODY_DENSITY_MEAN
EvolutionaryAlgorithmConfig, 33
INITIAL_BODY_DENSITY_VARIANCE
EvolutionaryAlgorithmConfig, 33
INITIAL_BODY_LENGTH_MEAN
EvolutionaryAlgorithmConfig, 33
INITIAL_BODY_LENGTH_VARIANCE
EvolutionaryAlgorithmConfig, 33
INITIAL_WHEEL_DENSITY_MEAN
EvolutionaryAlgorithmConfig, 34
INITIAL_WHEEL_DENSITY_VARIANCE
EvolutionaryAlgorithmConfig, 34
INITIAL_WHEEL_RADIUS_MEAN
EvolutionaryAlgorithmConfig, 34
INITIAL_WHEEL_RADIUS_VARIANCE
EvolutionaryAlgorithmConfig, 34

loadBGSprite
Utility.cc, 123
Utility.h, 138
loadBGSprites
Utility.cc, 124
Utility.h, 139
loadBGTextures
Utility.cc, 125
Utility.h, 140

main
Main.cc, 71
Main.cc
b2WorldPtr, 71
main, 71
world, 74
MapGenConfig, 37
BG_SPRITES_COUNT, 38
CAR_STARTING_X, 38
CAR_STARTING_Y, 38
GENERATE_DISTANCE, 38
GROUND_DEGREE_DEVIATION, 38
GROUND_LEG_LENGTH, 38
GROUND_PART_LENGTH, 39
GROUND_STARTING_X, 39
GROUND_STARTING_Y, 39
MAX_GROUND_DEGREE, 39
MASK_BITS
Config, 22
MAX_BODY_DENSITY
EvolutionaryAlgorithmConfig, 34
MAX_BODY_LENGTH
EvolutionaryAlgorithmConfig, 34
MAX_FPS
Config, 22
MAX_GROUND_DEGREE
MapGenConfig, 39
MAX_JOINT_LENGTH
CarConfig, 16
MAX_WHEEL_DENSITY
EvolutionaryAlgorithmConfig, 34
MAX_WHEEL_RADIUS
EvolutionaryAlgorithmConfig, 35
MIN_BODY_DENSITY
EvolutionaryAlgorithmConfig, 35
MIN_BODY_LENGTH
EvolutionaryAlgorithmConfig, 35
MIN_WHEEL_DENSITY
EvolutionaryAlgorithmConfig, 35
MIN_WHEEL_RADIUS
EvolutionaryAlgorithmConfig, 35
mutate
EvolutionaryAlgorithm, 27
MUTATION_FACTOR_BODY_DENSITY
EvolutionaryAlgorithmConfig, 35
MUTATION_FACTOR_BODY_LENGTHS
EvolutionaryAlgorithmConfig, 35
MUTATION_FACTOR_WHEEL_DENSITY
EvolutionaryAlgorithmConfig, 36
MUTATION_FACTOR_WHEEL_RADIUS
EvolutionaryAlgorithmConfig, 36
MUTATION_RATE_BODY_DENSITY
EvolutionaryAlgorithmConfig, 36
MUTATION_RATE_BODY_LENGTHS

EvolutionaryAlgorithmConfig, 36
MUTATION_RATE_WHEEL_DENSITY
EvolutionaryAlgorithmConfig, 36
MUTATION_RATE_WHEEL_RADIUS
EvolutionaryAlgorithmConfig, 36

nextGeneration
EvolutionaryAlgorithm, 29

PI
Config, 22
Polygon, 40
body, 41
color, 41
vertices, 41
POPULATION_SIZE
EvolutionaryAlgorithmConfig, 36
PPM
Config, 22
printEAInfo
GUI.cc, 60
GUI.h, 66

radius
Circle, 20
removeCars
Utility.cc, 126
Utility.h, 141
render
Render.cc, 77
Render.h, 86
Render.cc
render, 77
renderCar, 79
renderCircle, 80
renderCircleDebug, 81
renderPolygon, 82
renderPolygonDebug, 82
Render.h
render, 86
renderCar, 88
renderCircle, 89
renderCircleDebug, 90
renderPolygon, 91
renderPolygonDebug, 91
renderCar
Render.cc, 79
Render.h, 88
renderCircle
Render.cc, 80
Render.h, 89
renderCircleDebug
Render.cc, 81
Render.h, 90
renderPolygon
Render.cc, 82
Render.h, 91
renderPolygonDebug
Render.cc, 82
Render.h, 91
renderPositionPlot
GUI.cc, 60
GUI.h, 66
renderVelocityPlot
GUI.cc, 61
GUI.h, 67

SAVE_FILE_NAME
Config, 22
SAVE_TO_FILE
Config, 23
setCollisionFilter
Car, 14
setFitness
EvolutionaryAlgorithm, 30
setIcon
Utility.cc, 127
Utility.h, 142
SFMLColorToImVec4
Utility.cc, 128
Utility.h, 143
Shape.cc
createBox, 94
createCircle, 95
createGround, 96
createPolygon, 97
Shape.h
b2WorldPtr, 102
createBox, 102
createCircle, 104
createGround, 105
createPolygon, 106
ShapeTest.cc
TEST, 152–159
src/Car.cc, 48, 49
src/Car.h, 51, 53
src/EvolutionaryAlgorithm.cc, 54, 55
src/EvolutionaryAlgorithm.h, 57, 58
src/GUI.cc, 59, 63
src/GUI.h, 64, 69
src/Main.cc, 70, 75
src/Render.cc, 76, 83
src/Render.h, 85, 92
src/Shape.cc, 93, 99
src/Shape.h, 100, 108
src/UserInput.cc, 109, 111
src/UserInput.h, 112, 115
src/Utility.cc, 116, 129
src/Utility.h, 130, 144

TEST
CarTest.cc, 145
EvolutionaryAlgorithmTest.cc, 148, 149
ShapeTest.cc, 152–159
UtilityTest.cc, 163–168
test/CarTest.cc, 144, 146
test/EvolutionaryAlgorithmTest.cc, 147, 150
test/ShapeTest.cc, 151, 160

test/UtilityTest.cc, 162, 169
TOURNAMENT_SIZE
    EvolutionaryAlgorithmConfig, 37
tournamentSelection
    EvolutionaryAlgorithm, 30

UserInput.cc
    handleEvents, 110
    handleUserInput, 110
UserInput.h
    handleEvents, 114
    handleUserInput, 114
Utility.cc
    applyAirResistance, 117
    generateCar, 118
    generateGround, 119
    getFurthestCarPos, 120
    getIdxOfGrndClstToLoc, 121
    getNextGroundPartDegree, 122
    getRootDir, 122
    loadBGSprite, 123
    loadBGSprites, 124
    loadBGTextures, 125
    removeCars, 126
    setIcon, 127
    SFMLColorToImVec4, 128
Utility.h
    applyAirResistance, 132
    b2WorldPtr, 132
    generateCar, 133
    generateGround, 134
    getFurthestCarPos, 135
    getIdxOfGrndClstToLoc, 136
    getNextGroundPartDegree, 137
    getRootDir, 137
    loadBGSprite, 138
    loadBGSprites, 139
    loadBGTextures, 140
    removeCars, 141
    setIcon, 142
    SFMLColorToImVec4, 143
UtilityTest.cc
    TEST, 163–168

VELOCITY_ARRAY_SIZE
    Config, 23
vertices
    Polygon, 41

wheelDensity
    Chromosome, 18
wheelRadius
    Chromosome, 18
width
    Box, 6
WINDOW_HEIGHT
    Config, 23
WINDOW_WIDTH
    Config, 23
world
    Main.cc, 74