ZPR 23L Projekt - EvoRacer

Generated by Doxygen 1.9.7

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

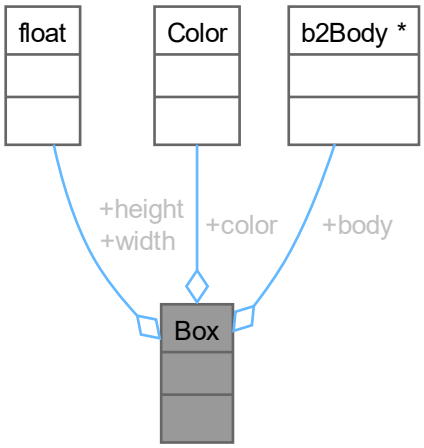
Class Documentation

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:

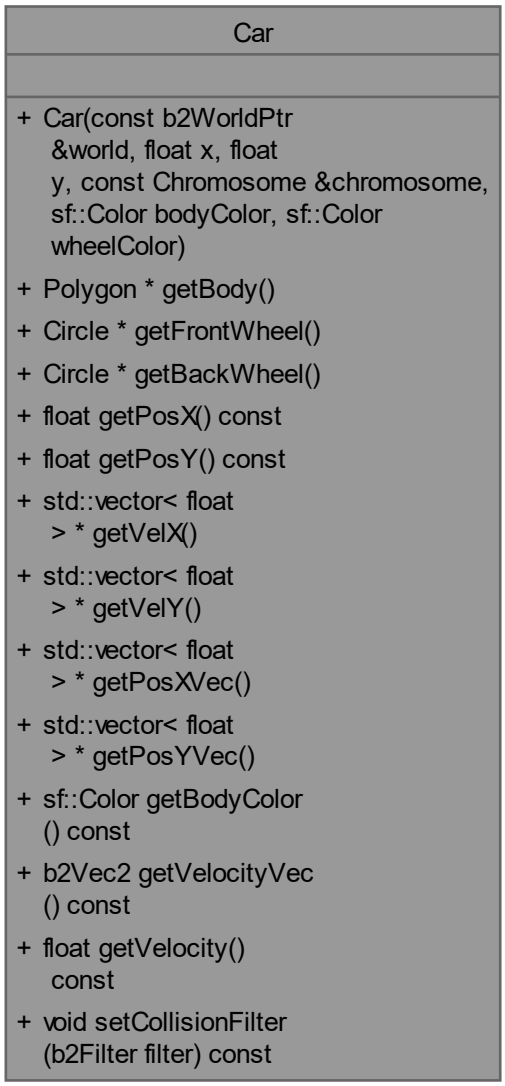
- src/Shape.h

3.2 Car Class Reference

Class representing a car.

`#include <Car.h>`

Collaboration diagram for Car:



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.

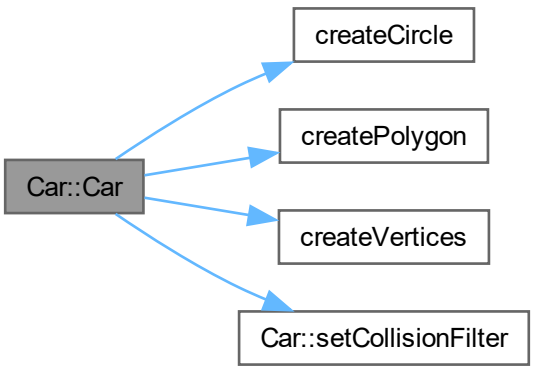
Parameters

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.

```
00013                                                {
00014     // Create a polygon (octagon)
00015
00016     auto vertices = createVertices(chromosome.bodyLengths);
00017
00018     body_ =
00019         createPolygon(world, x, y, vertices, chromosome.bodyDensity, Config::FRICTION, bodyColor);
00020
00021     // Create a circle
00022     frontWheel_ = createCircle(world, x, y, chromosome.wheelRadius.first,
00023                                chromosome.wheelDensity.first, Config::FRICTION, wheelColor);
00024
00025     // Create another circle
00026     backWheel_ = createCircle(world, x, y, chromosome.wheelRadius.second,
00027                               chromosome.wheelDensity.second, Config::FRICTION, wheelColor);
00028
00029     b2DistanceJointDef jointDef2;
00030     jointDef2.bodyA = body_.body;
00031     jointDef2.bodyB = frontWheel_.body;
00032     jointDef2.localAnchorA = vertices[Config::BACK_WHEEL_POS];
00033     jointDef2.localAnchorB = b2Vec2(0.0f, 0.0f);
00034     jointDef2.maxLength = CarConfig::MAX_JOINT_LENGTH;
00035     jointDef2.collideConnected = false;
00036     world->CreateJoint(&jointDef2);
00037
00038     jointDef2.bodyA = body_.body;
00039     jointDef2.bodyB = backWheel_.body;
00040     jointDef2.localAnchorA = vertices[Config::FRONT_WHEEL_POS];
00041     jointDef2.localAnchorB = b2Vec2(0.0f, 0.0f);
00042     jointDef2.maxLength = CarConfig::MAX_JOINT_LENGTH;
00043     jointDef2.collideConnected = false;
00044     world->CreateJoint(&jointDef2);
00045
00046     // Make cars pass through each-other
00047     // by setting collision filtering
00048     b2Filter filter;
00049     filter.categoryBits = Config::CATEGORY_BITS;
00050     filter.maskBits = Config::MASK_BITS;
00051     this->setCollisionFilter(filter);
00052
00053     std::vector<float> v_axis(Config::VELOCITY_ARRAY_SIZE);
00054     std::vector<float> v_values(Config::VELOCITY_ARRAY_SIZE);
00055
00056     std::iota(std::begin(v_axis), std::end(v_axis), 1);
00057
00058     velX_ = v_axis;
00059     velY_ = v_values;
00060
00061     posX_ = v_axis;
00062     posY_ = v_values;
00063 }
```

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.

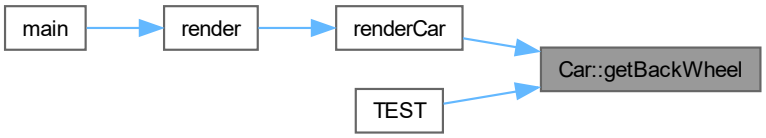
Returns

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).

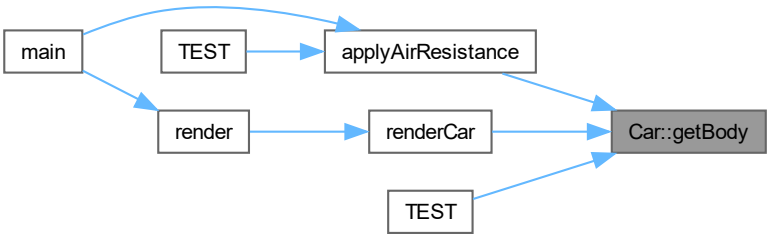
Returns

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.

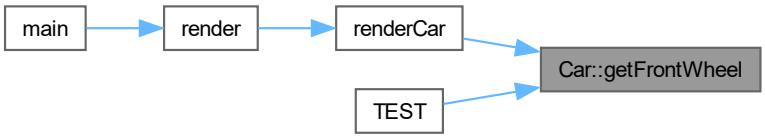
Returns

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.

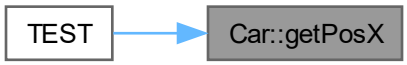
Returns

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.

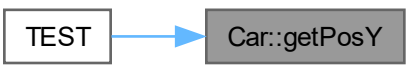
Returns

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.

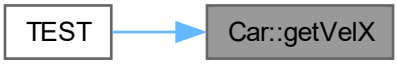
Returns

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.

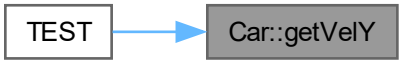
Returns

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:

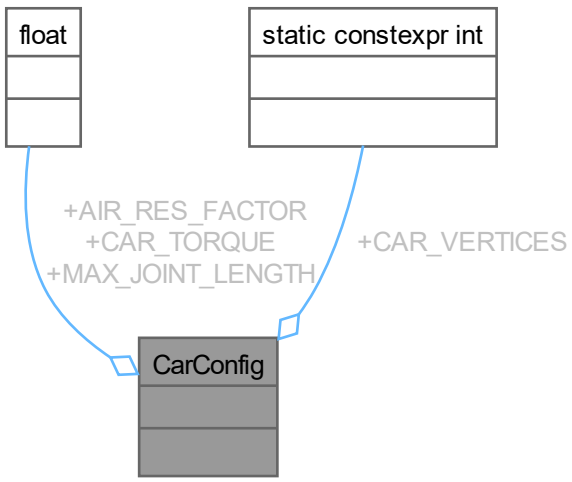
The documentation for this class was generated from the following files:

- 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
- static constexpr float AIR_RES_FACTOR = 3.4f

3.3.1 Detailed Description

Definition at line 12 of file CarConfig.h.

3.3.2 Member Data Documentation

3.3.2.1 AIR_RES_FACTOR

```
constexpr float CarConfig::AIR_RES_FACTOR = 3.4f  [static], [constexpr]
```

Definition at line 28 of file CarConfig.h.

3.3.2.2 CAR_TORQUE

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

Definition at line 15 of file CarConfig.h.

3.3.2.3 CAR_VERTICES

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

Definition at line 19 of file CarConfig.h.

3.3.2.4 MAX_JOINT_LENGTH

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

Definition at line 17 of file CarConfig.h.

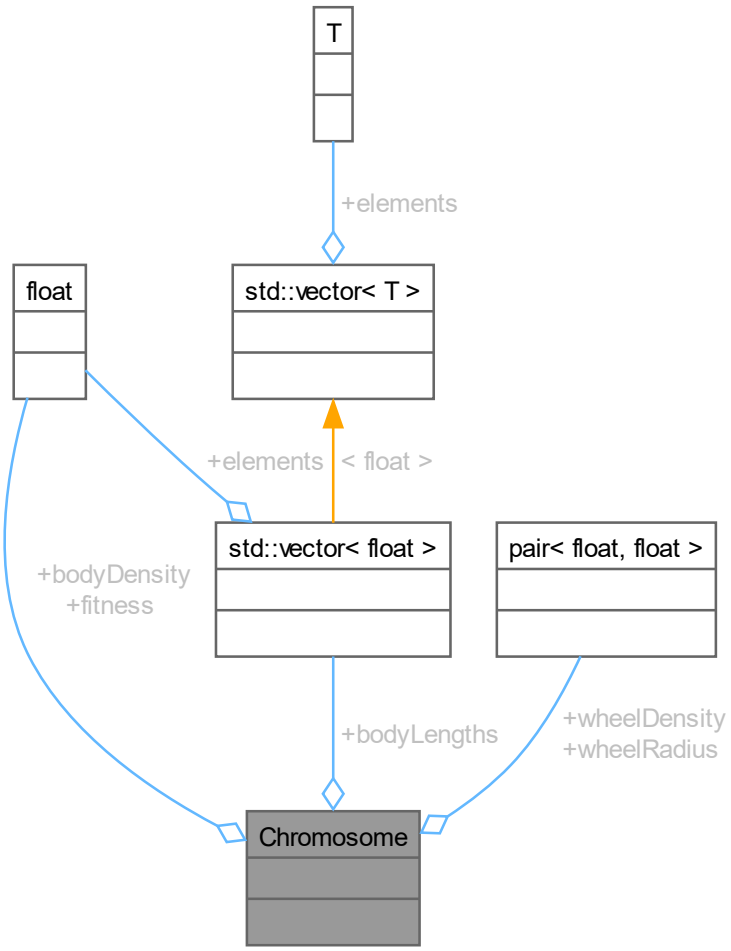
The documentation for this class was generated from the following file:

- 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 22 of file EvolutionaryAlgorithm.h.

3.4.2 Member Data Documentation

3.4.2.1 bodyDensity

```
float Chromosome::bodyDensity
```

Definition at line 24 of file EvolutionaryAlgorithm.h.

3.4.2.2 bodyLengths

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

Definition at line 23 of file EvolutionaryAlgorithm.h.

3.4.2.3 fitness

```
float Chromosome::fitness
```

Definition at line 27 of file EvolutionaryAlgorithm.h.

3.4.2.4 wheelDensity

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

Definition at line 26 of file EvolutionaryAlgorithm.h.

3.4.2.5 wheelRadius

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

Definition at line 25 of file EvolutionaryAlgorithm.h.

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

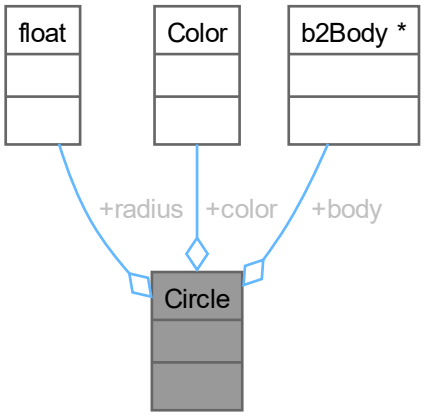
- src/EvolutionaryAlgorithm.h

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.

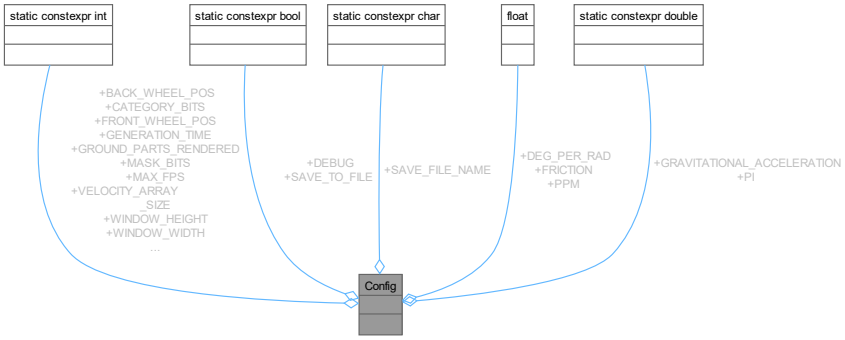
The documentation for this struct was generated from the following file:

- 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.

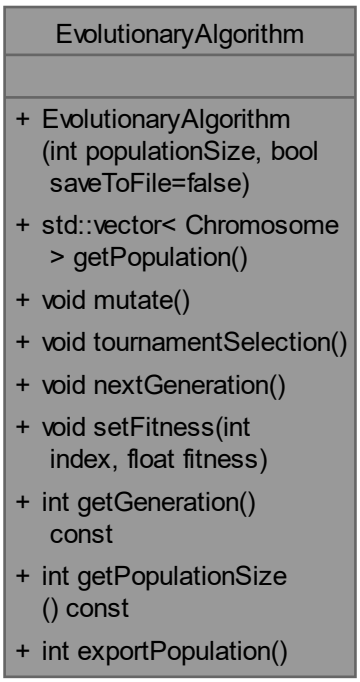
The documentation for this class was generated from the following file:

- config/Config.h

3.7 EvolutionaryAlgorithm Class Reference

`#include <EvolutionaryAlgorithm.h>`

Collaboration diagram for EvolutionaryAlgorithm:



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 30 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 ( )
```

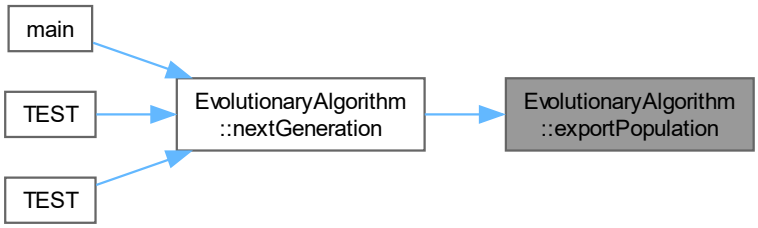
exports the population to a json file

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:



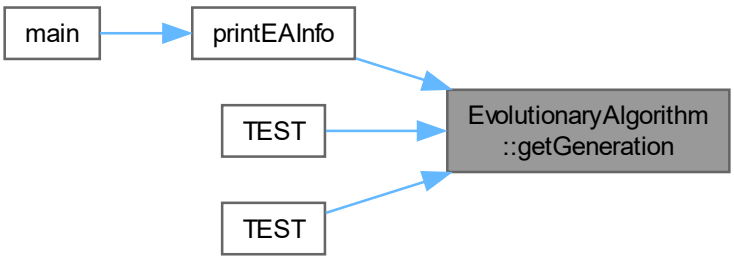
3.7.3.2 getGeneration()

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

Definition at line 68 of file EvolutionaryAlgorithm.h.

```
00068 { return generation_; }
```

Here is the caller graph for this function:



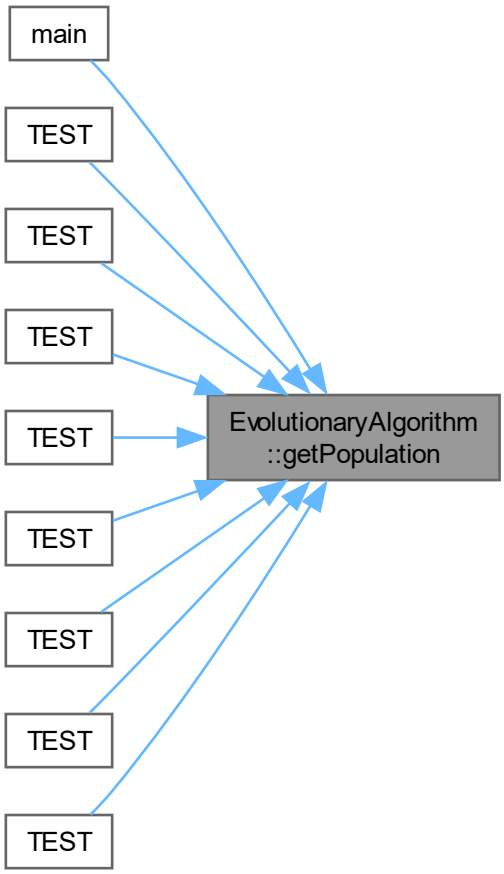
3.7.3.3 getPopulation()

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

Definition at line 54 of file EvolutionaryAlgorithm.h.

```
00054 { return population_; }
```

Here is the caller graph for this function:



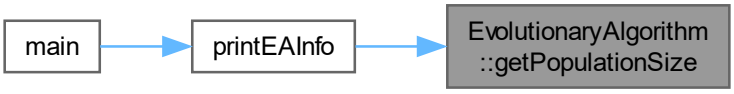
3.7.3.4 getPopulationSize()

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

Definition at line 69 of file EvolutionaryAlgorithm.h.

```
00069 { return populationSize_; }
```

Here is the caller graph for this function:



3.7.3.5 mutate()

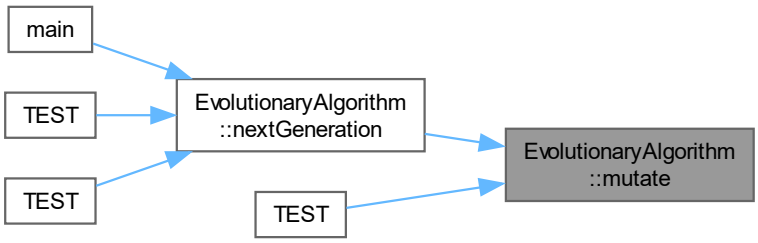
```
void EvolutionaryAlgorithm::mutate ( )
```

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:



3.7.3.6 nextGeneration()

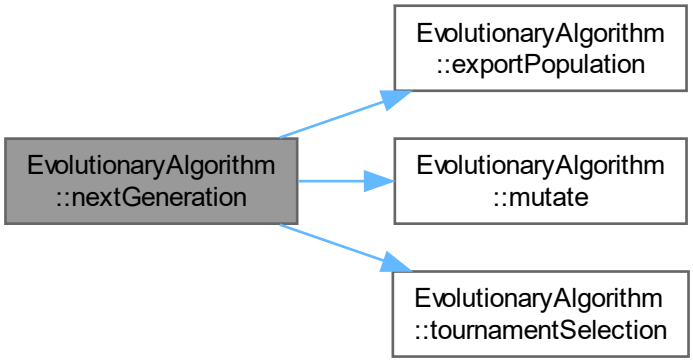
```
void EvolutionaryAlgorithm::nextGeneration ( )
```

steps the algorithm to the next generation

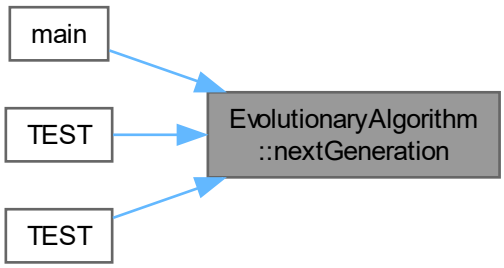
Definition at line 180 of file EvolutionaryAlgorithm.cc.

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

Here is the call graph for this function:



Here is the caller graph for this function:



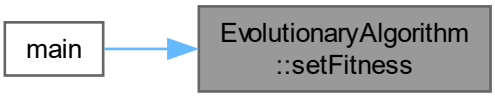
3.7.3.7 setFitness()

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

Definition at line 67 of file EvolutionaryAlgorithm.h.

```
00067 { population_[index].fitness = fitness; }
```

Here is the caller graph for this function:



3.7.3.8 tournamentSelection()

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

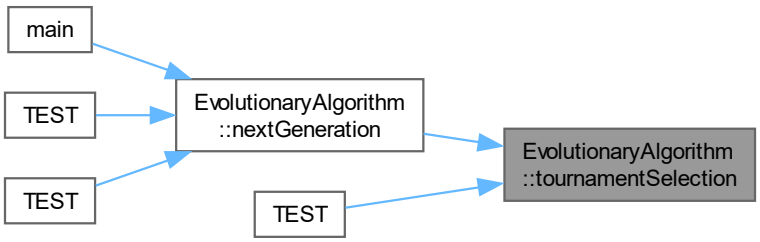
performs tournament selection on the population

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:



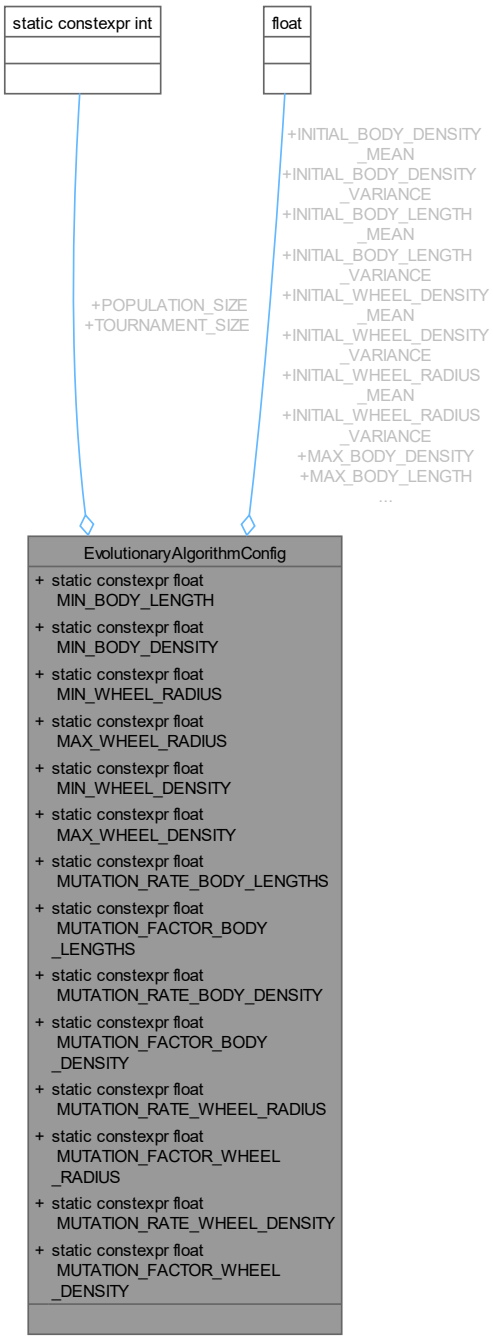
The documentation for this class was generated from the following files:

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

3.8 EvolutionaryAlgorithmConfig Class Reference

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

Collaboration diagram for EvolutionaryAlgorithmConfig:



Static Public Attributes

- static constexpr int POPULATION_SIZE = 16
- 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 37 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 38 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 34 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 35 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 43 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 44 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 40 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 41 of file EvolutionaryAlgorithmConfig.h.

3.8.2.9 MAX_BODY_DENSITY

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

Definition at line 24 of file EvolutionaryAlgorithmConfig.h.

3.8.2.10 MAX_BODY_LENGTH

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

Definition at line 21 of file EvolutionaryAlgorithmConfig.h.

3.8.2.11 MAX_WHEEL_DENSITY

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

Definition at line 30 of file EvolutionaryAlgorithmConfig.h.

3.8.2.12 MAX_WHEEL_RADIUS

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

Definition at line 27 of file EvolutionaryAlgorithmConfig.h.

3.8.2.13 MIN_BODY_DENSITY

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

Definition at line 23 of file EvolutionaryAlgorithmConfig.h.

3.8.2.14 MIN_BODY_LENGTH

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

Definition at line 20 of file EvolutionaryAlgorithmConfig.h.

3.8.2.15 MIN_WHEEL_DENSITY

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

Definition at line 29 of file EvolutionaryAlgorithmConfig.h.

3.8.2.16 MIN_WHEEL_RADIUS

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

Definition at line 26 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 52 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 49 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 58 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 55 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 51 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 48 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 57 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 54 of file EvolutionaryAlgorithmConfig.h.

3.8.2.25 POPULATION_SIZE

```
constexpr int EvolutionaryAlgorithmConfig::POPULATION_SIZE = 16  [static], [constexpr]
```

Definition at line 16 of file EvolutionaryAlgorithmConfig.h.

3.8.2.26 TOURNAMENT_SIZE

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

Definition at line 62 of file EvolutionaryAlgorithmConfig.h.

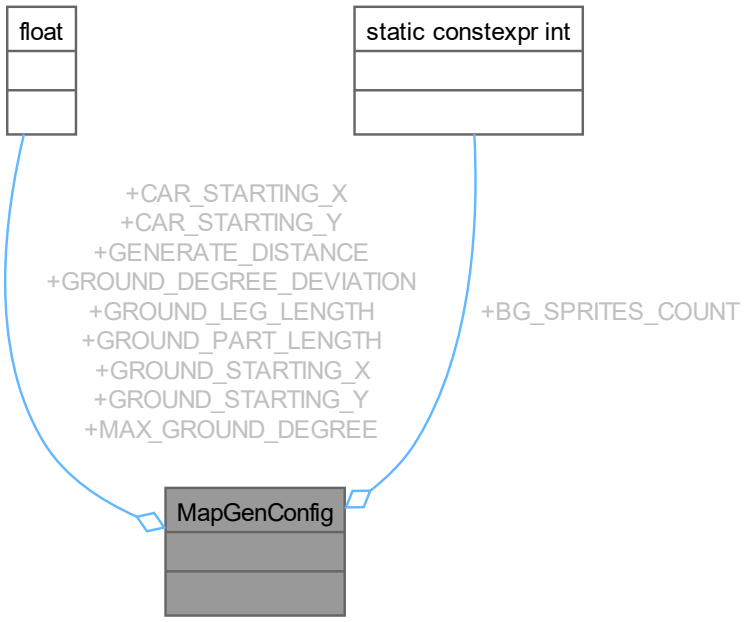
The documentation for this class was generated from the following file:

- 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 26 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 27 of file MapGenConfig.h.

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

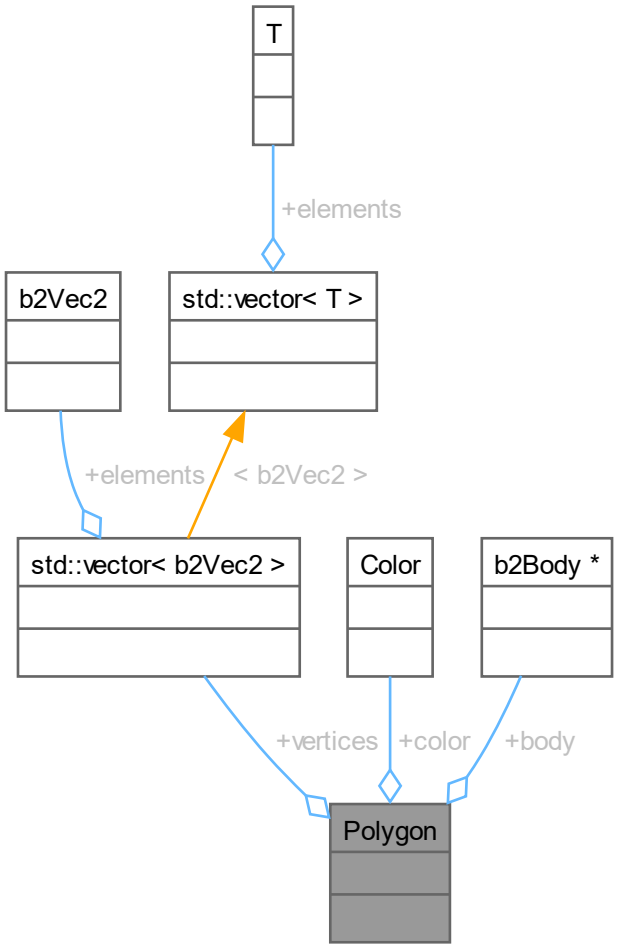
- config/MapGenConfig.h

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

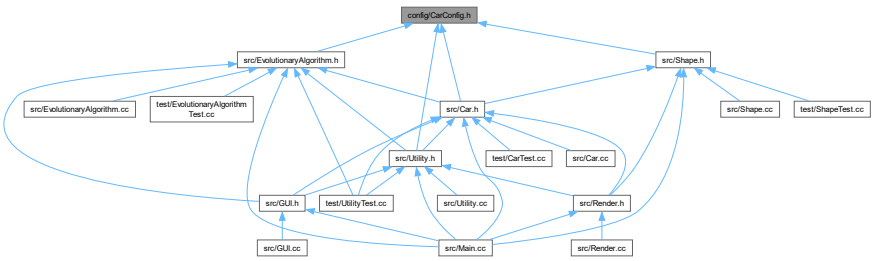
Chapter 4

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.

Authors

Mateusz Krakowski, Jakub Marcowski

Date

2023-06-07

Definition in file CarConfig.h.

4.2 CarConfig.h

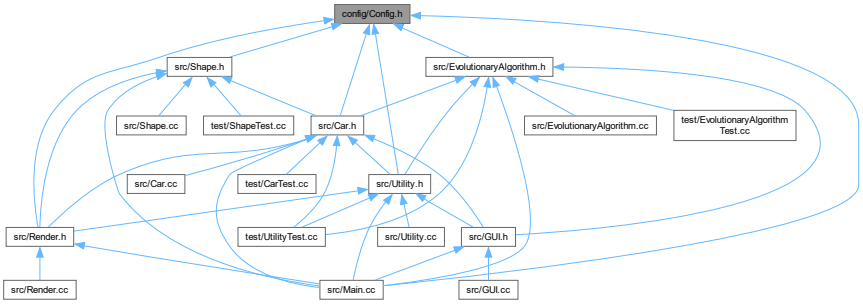
Go to the documentation of this file.

```
00001
00009 #ifndef CARCONFIG_H
00010 #define CARCONFIG_H
00011
00012 class CarConfig {
00013    public:
00014     // Car speed is dependent on the car's torque
00015     static constexpr float CAR_TORQUE = 2000.0f;
00016
00017     static constexpr float MAX_JOINT_LENGTH = 0.01f;
00018     // Number of vertices in a car's body polygon
00019     static constexpr int CAR_VERTICES = 8;
00020
00021     // F = V^2 * k
00022     // k  1/2 *  * A * C_d  3.4
00023     //  = 1.293 kg/m^3
00024     // A = ? (let's assume 5 m^2)
00025     // C_d = ? (let's assume 1.05)
00026     //
00027     // F = 3.4 * V^2
00028     static constexpr float AIR_RES_FACTOR = 3.4f;
00029 };
00030
00031 #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

2023-06-07

Definition in file Config.h.

4.4 Config.h

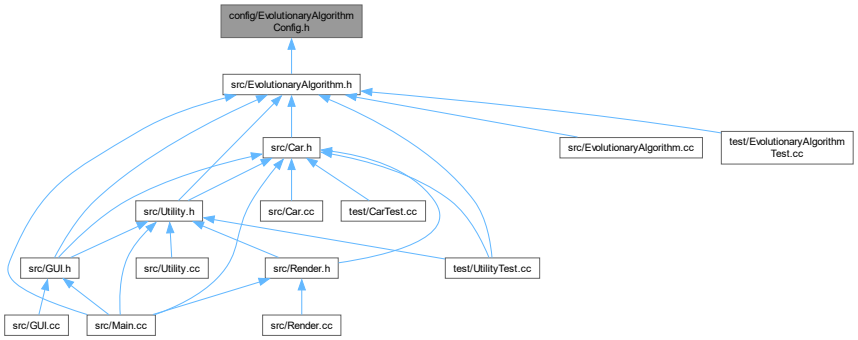
Go to the documentation of this file.

```
00001
00009 #ifndef CONFIG_H
00010 #define CONFIG_H
00011
00012 class Config {
00013   public:
00014     static constexpr int WINDOW_WIDTH = 1280;
00015     static constexpr int WINDOW_HEIGHT = 720;
00016
00017     // 60 for real time, 120 for fast forward - anything else is undefined behaviour
00018     static constexpr int MAX_FPS = 60;
00019
00020     static constexpr int GROUND_PARTS_RENDERED = 32;
00021
00022     // Exporting to file
00023     static constexpr bool SAVE_TO_FILE = true;
00024     static constexpr char SAVE_FILE_NAME[] = "evoRacerOutput.json";
00025
00026     static constexpr int GENERATION_TIME =
00027         3000;  // in frames, about 60 frames per second => 50 seconds
00028     // Pixels per meter. Box2D uses metric units, so we need PPM for conversion purposes
00029     static constexpr float PPM = 30.0F;
00030
00031     // SFML uses degrees for angles while Box2D uses radians
00032     static constexpr float DEG_PER_RAD = 57.2957795F;
00033
00034     // Draw debug geometry
00035     static constexpr bool DEBUG = true;
00036
00037     // Physics
00038     static constexpr double GRAVITATIONAL_ACCELERATION = -9.81f;
00039     static constexpr float FRICTION = 0.3f;
00040     static constexpr int VELOCITY_ARRAY_SIZE = 1000;
00041
00042     static constexpr double PI = 3.14159265358979323846;
00043     static constexpr int BACK_WHEEL_POS = 1;
00044     static constexpr int FRONT_WHEEL_POS = 3;
00045     static constexpr int CATEGORY_BITS = 2;
00046     static constexpr int MASK_BITS = 1;
00047 };
00048
00049 #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

2023-06-07

Definition in file EvolutionaryAlgorithmConfig.h.

4.6 EvolutionaryAlgorithmConfig.h

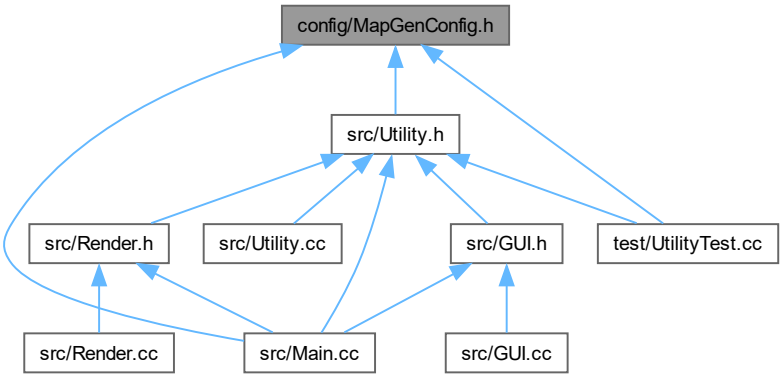
Go to the documentation of this file.

```
00001
00009 #ifndef EVOLUTIONARY_ALGORITHM_CONFIG_H
00010 #define EVOLUTIONARY_ALGORITHM_CONFIG_H
00011
00012 class EvolutionaryAlgorithmConfig {
00013    public:
00014     // Evolutionary algorithm parameters
00015
00016     static constexpr int POPULATION_SIZE = 16;
00017
00018     // Boundaries for the chromosomes
00019
00020     static constexpr float MIN_BODY_LENGTH = 1.0f;
00021     static constexpr float MAX_BODY_LENGTH = 5.0f;
00022
00023     static constexpr float MIN_BODY_DENSITY = 10.0f;
00024     static constexpr float MAX_BODY_DENSITY = 1000.0f;
00025
00026     static constexpr float MIN_WHEEL_RADIUS = 2.0f;
00027     static constexpr float MAX_WHEEL_RADIUS = 40.0f;
00028
00029     static constexpr float MIN_WHEEL_DENSITY = 10.0f;
00030     static constexpr float MAX_WHEEL_DENSITY = 1000.0f;
00031
00032     // Population initialization hyper parameters
00033
00034     static constexpr float INITIAL_BODY_LENGTH_MEAN = 3.0f;
00035     static constexpr float INITIAL_BODY_LENGTH_VARIANCE = 1.0f;
00036
00037     static constexpr float INITIAL_BODY_DENSITY_MEAN = 100.0f;
00038     static constexpr float INITIAL_BODY_DENSITY_VARIANCE = 100.0f;
00039
00040     static constexpr float INITIAL_WHEEL_RADIUS_MEAN = 25.0f;
00041     static constexpr float INITIAL_WHEEL_RADIUS_VARIANCE = 10.0f;
00042
00043     static constexpr float INITIAL_WHEEL_DENSITY_MEAN = 100.0f;
00044     static constexpr float INITIAL_WHEEL_DENSITY_VARIANCE = 100.0f;
00045
00046     // Mutation hyper parameters
00047
00048     static constexpr float MUTATION_RATE_BODY_LENGTHS = 0.1f;
00049     static constexpr float MUTATION_FACTOR_BODY_LENGTHS = 0.5f;
00050
00051     static constexpr float MUTATION_RATE_BODY_DENSITY = 0.2f;
00052     static constexpr float MUTATION_FACTOR_BODY_DENSITY = 20.0f;
00053
00054     static constexpr float MUTATION_RATE_WHEEL_RADIUS = 0.3f;
00055     static constexpr float MUTATION_FACTOR_WHEEL_RADIUS = 2.0f;
00056
00057     static constexpr float MUTATION_RATE_WHEEL_DENSITY = 0.1f;
00058     static constexpr float MUTATION_FACTOR_WHEEL_DENSITY = 20.0f;
00059
00060     // Selection hyper parameters
00061
00062     static constexpr int TOURNAMENT_SIZE = 3;  // Has to be equal to or lesser than POPULATION_SIZE
00063 };
00064
00065 #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

2023-06-07

Definition in file MapGenConfig.h.

4.8 MapGenConfig.h

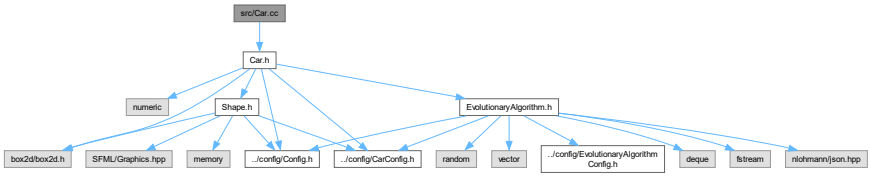
Go to the documentation of this file.

```
00001
00009 #ifndef MAPGENCONFIG_H
00010 #define MAPGENCONFIG_H
00011
00012 class MapGenConfig {
00013    public:
00014     static constexpr float GENERATE_DISTANCE = 666.0;
00015     static constexpr float GROUND_STARTING_X = 0.0;
00016     static constexpr float GROUND_STARTING_Y = 360.0;
00017     static constexpr float GROUND_LEG_LENGTH = 4.0;
00018     static constexpr float GROUND_PART_LENGTH = 1.5;
00019     static constexpr int BG_SPRITES_COUNT = 5;
00020
00021     static constexpr float CAR_STARTING_X = 250.0;
00022     static constexpr float CAR_STARTING_Y = 650.0;
00023
00024     // Change the mapgen behaviour here
00025
00026     static constexpr float GROUND_DEGREE_DEVIATION = 12.0f;
00027     static constexpr float MAX_GROUND_DEGREE = 50.0f;
00028 };
00029
00030 #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

2023-06-07

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.

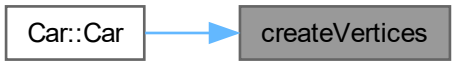
Returns

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_ =
```

```
00019         createPolygon(world, x, y, vertices, chromosome.bodyDensity, Config::FRICTION, bodyColor);
00020
00021     // Create a circle
00022     frontWheel_ = createCircle(world, x, y, chromosome.wheelRadius.first,
00023                                chromosome.wheelDensity.first, Config::FRICTION, wheelColor);
00024
00025     // Create another circle
00026     backWheel_ = createCircle(world, x, y, chromosome.wheelRadius.second,
00027                               chromosome.wheelDensity.second, Config::FRICTION, wheelColor);
00028
00029     b2DistanceJointDef jointDef2;
00030     jointDef2.bodyA = body_.body;
00031     jointDef2.bodyB = frontWheel_.body;
00032     jointDef2.localAnchorA = vertices[Config::BACK_WHEEL_POS];
00033     jointDef2.localAnchorB = b2Vec2(0.0f, 0.0f);
00034     jointDef2.maxLength = CarConfig::MAX_JOINT_LENGTH;
00035     jointDef2.collideConnected = false;
00036     world->CreateJoint(&jointDef2);
00037
00038     jointDef2.bodyA = body_.body;
00039     jointDef2.bodyB = backWheel_.body;
00040     jointDef2.localAnchorA = vertices[Config::FRONT_WHEEL_POS];
00041     jointDef2.localAnchorB = b2Vec2(0.0f, 0.0f);
00042     jointDef2.maxLength = CarConfig::MAX_JOINT_LENGTH;
00043     jointDef2.collideConnected = false;
00044     world->CreateJoint(&jointDef2);
00045
00046     // Make cars pass through each-other
00047     // by setting collision filtering
00048     b2Filter filter;
00049     filter.categoryBits = Config::CATEGORY_BITS;
00050     filter.maskBits = Config::MASK_BITS;
00051     this->setCollisionFilter(filter);
00052
00053     std::vector<float> v_axis(Config::VELOCITY_ARRAY_SIZE);
00054     std::vector<float> v_values(Config::VELOCITY_ARRAY_SIZE);
00055
00056     std::iota(std::begin(v_axis), std::end(v_axis), 1);
00057
00058     velX_ = v_axis;
00059     velY_ = v_values;
00060
00061     posX_ = v_axis;
00062     posY_ = v_values;
00063 }
00064
00065 Polygon* Car::getBody() { return &body_; }
00066
00067 Circle* Car::getFrontWheel() { return &frontWheel_; }
00068
00069 Circle* Car::getBackWheel() { return &backWheel_; }
00070
00071 float Car::getPosX() const { return body_.body->GetPosition().x; }
00072
00073 float Car::getPosY() const { return body_.body->GetPosition().y; }
00074
00075 std::vector<float>* Car::getVelX() { return &velX_; }
00076
00077 std::vector<float>* Car::getVelY() { return &velY_; }
00078
00079 std::vector<float>* Car::getPosXVec() { return &posX_; }
00080
00081 std::vector<float>* Car::getPosYVec() { return &posY_; }
00082
00083 sf::Color Car::getBodyColor() const { return body_.color; }
00084
00085 b2Vec2 Car::getVelocityVec() const { return body_.body->GetLinearVelocity(); }
00086
00087 float Car::getVelocity() const { return body_.body->GetLinearVelocity().Length(); }
00088
00089 void Car::setCollisionFilter(b2Filter filter) const {
00090     body_.body->GetFixtureList()->SetFilterData(filter);
00091     frontWheel_.body->GetFixtureList()->SetFilterData(filter);
00092     backWheel_.body->GetFixtureList()->SetFilterData(filter);
00093 }
00094
00095 std::vector<b2Vec2> createVertices(std::vector<float> lengths) {
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
```

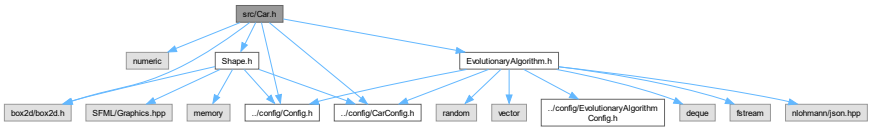
```
00106     for (int i = 0; i < lengths.size(); ++i) {
00107         vertices.emplace_back(lengths[i] * cos(angle), lengths[i] * sin(angle));
00108         angle += (angles[i] / 180.0f) * Config::PI;
00109     }
00110     return vertices;
00111 }
```

4.11 src/Car.h File Reference

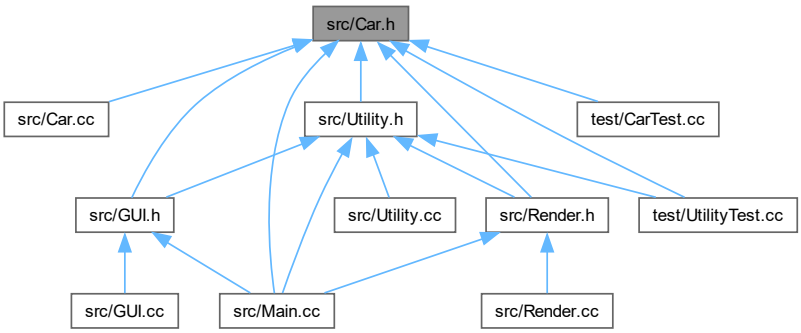
Header file for the Car class.

```
#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

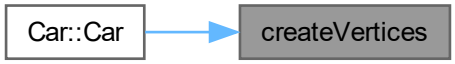
std::vector<b2Vec2> Vector of vertices.

Definition at line 95 of file Car.cc.

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

```
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:



4.12 Car.h

Go to the documentation of this file.

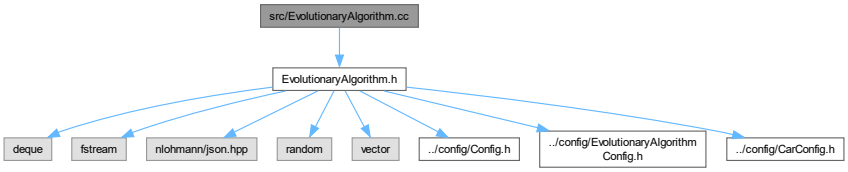
```
00001
00009 #ifndef CAR_H
00010 #define CAR_H
00011
00012 #include <numeric>
00013
00014 #include "box2d/box2d.h"
00015
00016 #include "../config/Config.h"
00017 #include "../config/CarConfig.h"
00018 #include "Shape.h"
00019 #include "EvolutionaryAlgorithm.h"
00020
00021 typedef std::shared_ptr<b2World> b2WorldPtr;
00022
00026 class Car {
00027    private:
00028     b2WorldPtr world_;
00029     Polygon body_;
00030     Circle frontWheel_;
00031     Circle backWheel_;
00032     std::vector<float> velX_;
00033     std::vector<float> velY_;
00034     std::vector<float> posX_;
00035     std::vector<float> posY_;
00036
00037    public:
00048     Car(const b2WorldPtr& world, float x, float y, const Chromosome& chromosome,
00049         sf::Color bodyColor, sf::Color wheelColor);
00050
00056     Polygon* getBody();
00057
00063     Circle* getFrontWheel();
00064
00070     Circle* getBackWheel();
00071
00077     float getPosX() const;
00078
00084     float getPosY() const;
00085
00091     std::vector<float>* getVelX();
00092
00098     std::vector<float>* getVelY();
00099
```

```
00105     std::vector<float>* getPosXVec();
00106
00112     std::vector<float>* getPosYVec();
00113
00119     sf::Color getBodyColor() const;
00120
00126     b2Vec2 getVelocityVec() const;
00127
00133     float getVelocity() const;
00134
00140     void setCollisionFilter(b2Filter filter) const;
00141 };
00142
00149 std::vector<b2Vec2> createVertices(std::vector<float> lengths);
00150
00151 #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.

```
00001
00010 #include "EvolutionaryAlgorithm.h"
00011
00012 EvolutionaryAlgorithm::EvolutionaryAlgorithm(int populationSize, bool saveToFile) {
00013     populationSize_ = populationSize;
00014     generation_ = 0;
00015     initializePopulation();
00016     saveToFile_ = saveToFile;
00017 }
00018
00019 void EvolutionaryAlgorithm::initializePopulation() {
00020     std::random_device rd;
00021     std::mt19937 gen(rd());
00022     std::normal_distribution<float> dist(0.0, 1.0);
00023     // add variation and mean
00024     for (int i = 0; i < populationSize_; ++i) {
00025         Chromosome chrom;
00026
00027         for (int p = 0; p < CarConfig::CAR_VERTICES; ++p) {
00028             float length = dist(gen) * EvolutionaryAlgorithmConfig::INITIAL_BODY_LENGTH_VARIANCE +
00029                            EvolutionaryAlgorithmConfig::INITIAL_BODY_LENGTH_MEAN;
00030             length = std::max(length, EvolutionaryAlgorithmConfig::MIN_BODY_LENGTH);
00031             length = std::min(length, EvolutionaryAlgorithmConfig::MAX_BODY_LENGTH);
00032             chrom.bodyLengths.push_back(length);
00033         }
00034
00035         // Initialize bodyDensity
00036
00037         chrom.bodyDensity = dist(gen) * EvolutionaryAlgorithmConfig::INITIAL_BODY_DENSITY_VARIANCE +
00038                             EvolutionaryAlgorithmConfig::INITIAL_BODY_DENSITY_MEAN;
00039
00040         chrom.bodyDensity =
00041             std::max(chrom.bodyDensity, EvolutionaryAlgorithmConfig::MIN_BODY_DENSITY);
00042         chrom.bodyDensity =
00043             std::min(chrom.bodyDensity, EvolutionaryAlgorithmConfig::MAX_BODY_DENSITY);
00044
00045         // initialize wheelRadius
00046
00047         chrom.wheelRadius.first =
00048             dist(gen) * EvolutionaryAlgorithmConfig::INITIAL_WHEEL_RADIUS_VARIANCE +
00049             EvolutionaryAlgorithmConfig::INITIAL_WHEEL_RADIUS_MEAN;
00050         chrom.wheelRadius.first =
00051             std::max(chrom.wheelRadius.first, EvolutionaryAlgorithmConfig::MIN_WHEEL_RADIUS);
00052         chrom.wheelRadius.first =
00053
00054             chrom.wheelRadius.second +=
00055             dist(gen) * EvolutionaryAlgorithmConfig::INITIAL_WHEEL_RADIUS_VARIANCE +
00056             EvolutionaryAlgorithmConfig::INITIAL_WHEEL_RADIUS_MEAN;
00057         chrom.wheelRadius.second =
00058             std::max(chrom.wheelRadius.second, EvolutionaryAlgorithmConfig::MIN_WHEEL_RADIUS);
00059         chrom.wheelRadius.second =
00060             std::min(chrom.wheelRadius.second, EvolutionaryAlgorithmConfig::MAX_WHEEL_RADIUS);
00061
00062         // Initialize wheelDensity
00063
00064         chrom.wheelDensity.first +=
00065             dist(gen) * EvolutionaryAlgorithmConfig::INITIAL_WHEEL_DENSITY_VARIANCE +
00066             EvolutionaryAlgorithmConfig::INITIAL_WHEEL_DENSITY_MEAN;
00067         chrom.wheelDensity.first =
00068             std::max(chrom.wheelDensity.first, EvolutionaryAlgorithmConfig::MIN_WHEEL_DENSITY);
00069         chrom.wheelDensity.first =
00070             std::min(chrom.wheelDensity.first, EvolutionaryAlgorithmConfig::MAX_WHEEL_DENSITY);
00071
00072         chrom.wheelDensity.second +=
00073             dist(gen) * EvolutionaryAlgorithmConfig::INITIAL_WHEEL_DENSITY_VARIANCE +
00074             EvolutionaryAlgorithmConfig::INITIAL_WHEEL_DENSITY_MEAN;
00075         chrom.wheelDensity.second =
00076             std::max(chrom.wheelDensity.second, EvolutionaryAlgorithmConfig::MIN_WHEEL_DENSITY);
00077         chrom.wheelDensity.second =
00078             std::min(chrom.wheelDensity.second, EvolutionaryAlgorithmConfig::MAX_WHEEL_DENSITY);
00079
00080         population_.push_back(chrom);
00081     }
00082 }
00083
00084 void EvolutionaryAlgorithm::mutate() {
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 }
00147 Chromosome EvolutionaryAlgorithm::tournament() {
00148     std::random_device rd;
00149     std::mt19937 gen(rd());
00150     std::uniform_int_distribution<> uniform_dist(0, populationSize_ - 1);
00151 
00152     std::vector<Chromosome> candidates;
00153 
00154     candidates.reserve(EvolutionaryAlgorithmConfig::TOURNAMENT_SIZE);
00155     for (int i = 0; i < EvolutionaryAlgorithmConfig::TOURNAMENT_SIZE; ++i) {
00156         candidates.push_back(population_[uniform_dist(gen)]);
00157     }
00158 
00159     Chromosome tournament_winner = candidates[0];
00160 
00161     for (int i = 1; i < EvolutionaryAlgorithmConfig::TOURNAMENT_SIZE; ++i) {
00162         if (candidates[i].fitness > tournament_winner.fitness) {
00163             tournament_winner = candidates[i];
00164         }
00165     }
00166 
00167     return tournament_winner;
00168 }
00169 void EvolutionaryAlgorithm::tournamentSelection() {
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;
```

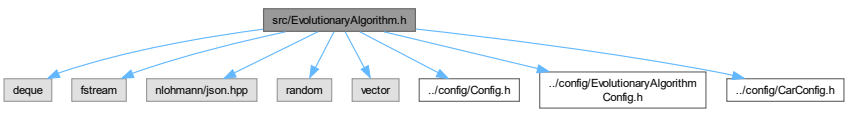
```
00178 }
00179
00180 void EvolutionaryAlgorithm::nextGeneration() {
00181     if (saveToFile_) {
00182         exportPopulation();
00183     }
00184     tournamentSelection();
00185     mutate();
00186     ++generation_;
00187 }
00188
00189 int EvolutionaryAlgorithm::exportPopulation() {
00190     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 }
```

4.15 src/EvolutionaryAlgorithm.h File Reference

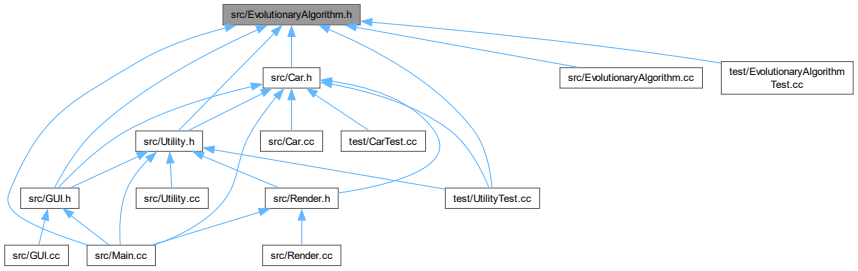
Header file for EvolutionaryAlgorithm class.

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

Include dependency graph for EvolutionaryAlgorithm.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.

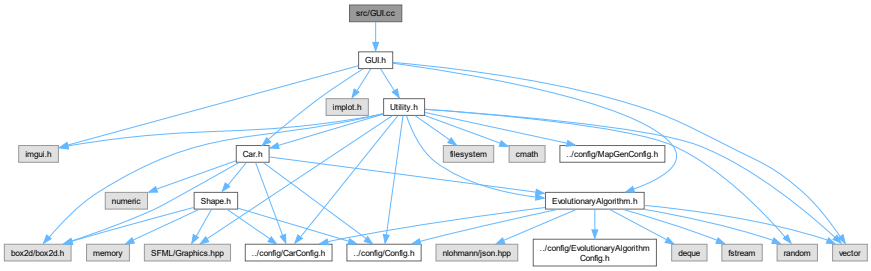
```
00001
00009 #ifndef GENOME_H
00010 #define GENOME_H
00011
00012 #include <deque>
00013 #include <fstream>
00014 #include <nlohmann/json.hpp>
00015 #include <random>
00016 #include <vector>
00017
00018 #include "../config/Config.h"
00019 #include "../config/EvolutionaryAlgorithmConfig.h"
00020 #include "../config/CarConfig.h"
00021
00022 struct Chromosome {
00023     std::vector<float> bodyLengths;
00024     float bodyDensity;
00025     std::pair<float, float> wheelRadius;
00026     std::pair<float, float> wheelDensity;
00027     float fitness;
00028 };
00029
```

```
00030 class EvolutionaryAlgorithm {
00031    private:
00032     unsigned long int generation_;
00033     unsigned long int populationSize_;
00034     std::vector<Chromosome> population_;
00035     bool saveToFile_ = false;
00040     Chromosome tournament();
00044     void initializePopulation();
00045
00046    public:
00053     explicit EvolutionaryAlgorithm(int populationSize, bool saveToFile = false);
00054     std::vector<Chromosome> getPopulation() { return population_; }
00058     void mutate();
00062     void tournamentSelection();
00066     void nextGeneration();
00067     void setFitness(int index, float fitness) { population_[index].fitness = fitness; }
00068     int getGeneration() const { return generation_; }
00069     int getPopulationSize() const { return populationSize_; }
00073     int exportPopulation();
00074 };
00075
00076 #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

2023-06-07

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.

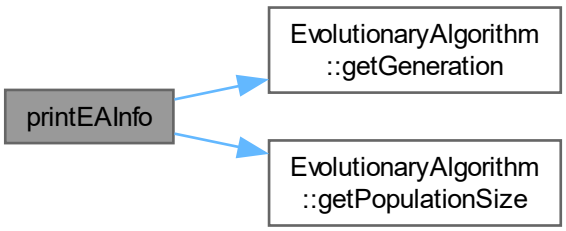
Parameters

ea	Evolutionary Algorithm object.

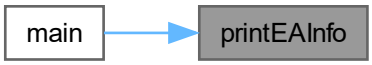
Definition at line 61 of file GUI.cc.

```
00061                                                {
00062     ImGui::Begin("EA Info");
00063     ImGui::Text("Generation: %d", ea.getGeneration());
00064     ImGui::Text("Population size: %d", ea.getPopulationSize());
00065     ImGui::End();
00066 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.17.2.2 renderPositionPlot()

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

Renders position plot.

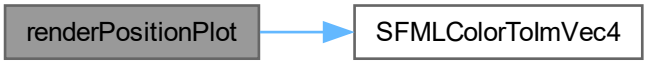
Parameters

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

Definition at line 35 of file GUI.cc.

```
00035                                                                    {
00036     ImGui::Begin("Cars' Position");
00037     ImPlot::SetNextAxesToFit();
00038     if (ImPlot::BeginPlot("Position")) {
00039         ImPlot::SetupLegend(ImPlotLocation_West, ImPlotLegendFlags_Outside);
00040         for (int i = 0; i < cars.size(); ++i) {
00041             if (!paused) {
00042                 cars[i].getPosXVec()->push_back(cars[i].getPosXVec()->back() + 1);
00043                 cars[i].getPosYVec()->push_back(cars[i].getBody()->body->GetPosition().x);
00044             }
00045             std::vector<float> v_axis_crop =
00046                 std::vector<float>(cars[i].getPosXVec()->end() - Config::VELOCITY_ARRAY_SIZE,
00047                                    cars[i].getPosXVec()->end());
00048             std::vector<float> v_values_crop =
00049                 std::vector<float>(cars[i].getPosYVec()->end() - Config::VELOCITY_ARRAY_SIZE,
00050                                    cars[i].getPosYVec()->end());
00051             ImPlot::PushStyleColor(ImPlotCol_Line, SFMLColorToImVec4(cars[i].getBodyColor()));
00052             ImPlot::PlotLine(std::to_string(i).c_str(), &(v_axis_crop[0]), &(v_values_crop[0]),
00053                              Config::VELOCITY_ARRAY_SIZE);
00054             ImPlot::PopStyleColor();
00055         }
00056         ImPlot::EndPlot();
00057     }
00058     ImGui::End();
00059 }
```

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.

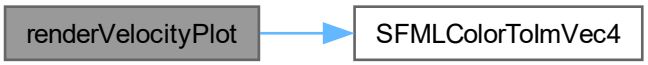
Parameters

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             if (!paused) {
00018                 cars[i].getVelX()->push_back(cars[i].getVelX()->back() + 1);
00019                 cars[i].getVelY()->push_back(cars[i].getVelocity());
00020             }
00021             std::vector<float> v_axis_crop = std::vector<float>(
00022                 cars[i].getVelX()->end() - Config::VELOCITY_ARRAY_SIZE, cars[i].getVelX()->end());
00023             std::vector<float> v_values_crop = std::vector<float>(
00024                 cars[i].getVelY()->end() - Config::VELOCITY_ARRAY_SIZE, cars[i].getVelY()->end());
00025             ImPlot::PushStyleColor(ImPlotCol_Line, SFMLColorToImVec4(cars[i].getBodyColor()));
00026             ImPlot::PlotLine(std::to_string(i).c_str(), &(v_axis_crop[0]), &(v_values_crop[0]),
00027                              Config::VELOCITY_ARRAY_SIZE);
00028             ImPlot::PopStyleColor();
00029         }
00030         ImPlot::EndPlot();
00031     }
00032     ImGui::End();
00033 }
```

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"
00010
00011 void renderVelocityPlot(std::vector<Car>& cars, bool paused) {
00012     ImGui::Begin("Cars' Velocity");
```

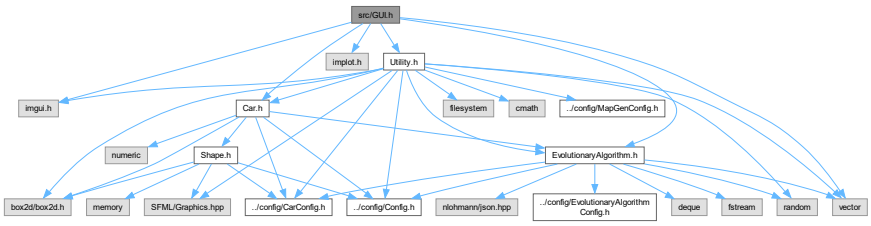
```
00013     ImPlot::SetNextAxesToFit();
00014     if (ImPlot::BeginPlot("Velocity")) {
00015         ImPlot::SetupLegend(ImPlotLocation_West, ImPlotLegendFlags_Outside);
00016         for (int i = 0; i < cars.size(); ++i) {
00017             if (!paused) {
00018                 cars[i].getVelX()->push_back(cars[i].getVelX()->back() + 1);
00019                 cars[i].getVelY()->push_back(cars[i].getVelocity());
00020             }
00021             std::vector<float> v_axis_crop = std::vector<float>(
00022                 cars[i].getVelX()->end() - Config::VELOCITY_ARRAY_SIZE, cars[i].getVelX()->end());
00023             std::vector<float> v_values_crop = std::vector<float>(
00024                 cars[i].getVelY()->end() - Config::VELOCITY_ARRAY_SIZE, cars[i].getVelY()->end());
00025             ImPlot::PushStyleColor(ImPlotCol_Line, SFMLColorToImVec4(cars[i].getBodyColor()));
00026             ImPlot::PlotLine(std::to_string(i).c_str(), &(v_axis_crop[0]), &(v_values_crop[0]),
00027                              Config::VELOCITY_ARRAY_SIZE);
00028             ImPlot::PopStyleColor();
00029         }
00030         ImPlot::EndPlot();
00031     }
00032     ImGui::End();
00033 }
00034
00035 void renderPositionPlot(std::vector<Car>& cars, bool paused) {
00036     ImGui::Begin("Cars' Position");
00037     ImPlot::SetNextAxesToFit();
00038     if (ImPlot::BeginPlot("Position")) {
00039         ImPlot::SetupLegend(ImPlotLocation_West, ImPlotLegendFlags_Outside);
00040         for (int i = 0; i < cars.size(); ++i) {
00041             if (!paused) {
00042                 cars[i].getPosXVec()->push_back(cars[i].getPosXVec()->back() + 1);
00043                 cars[i].getPosYVec()->push_back(cars[i].getBody()->body->GetPosition().x);
00044             }
00045             std::vector<float> v_axis_crop =
00046                 std::vector<float>(cars[i].getPosXVec()->end() - Config::VELOCITY_ARRAY_SIZE,
00047                                    cars[i].getPosXVec()->end());
00048             std::vector<float> v_values_crop =
00049                 std::vector<float>(cars[i].getPosYVec()->end() - Config::VELOCITY_ARRAY_SIZE,
00050                                    cars[i].getPosYVec()->end());
00051             ImPlot::PushStyleColor(ImPlotCol_Line, SFMLColorToImVec4(cars[i].getBodyColor()));
00052             ImPlot::PlotLine(std::to_string(i).c_str(), &(v_axis_crop[0]), &(v_values_crop[0]),
00053                              Config::VELOCITY_ARRAY_SIZE);
00054             ImPlot::PopStyleColor();
00055         }
00056         ImPlot::EndPlot();
00057     }
00058     ImGui::End();
00059 }
00060
00061 void printEAInfo(EvolutionaryAlgorithm& ea) {
00062     ImGui::Begin("EA Info");
00063     ImGui::Text("Generation: %d", ea.getGeneration());
00064     ImGui::Text("Population size: %d", ea.getPopulationSize());
00065     ImGui::End();
00066 }
```

4.19 src/GUI.h File Reference

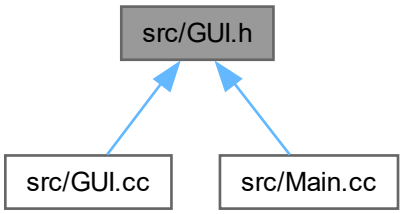
Header for a file containing GUI functions.

```
#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.

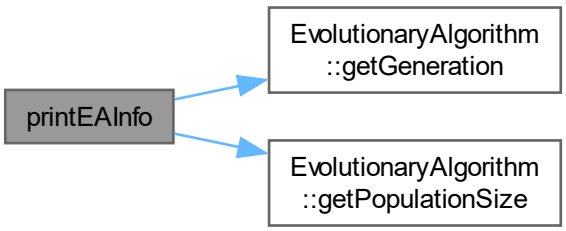
Parameters

ea	Evolutionary Algorithm object.

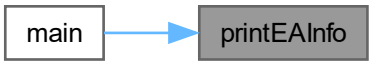
Definition at line 61 of file GUI.cc.

```
00061                                           {
00062     ImGui::Begin("EA Info");
00063     ImGui::Text("Generation: %d", ea.getGeneration());
00064     ImGui::Text("Population size: %d", ea.getPopulationSize());
00065     ImGui::End();
00066 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.19.2.2 renderPositionPlot()

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

Renders position plot.

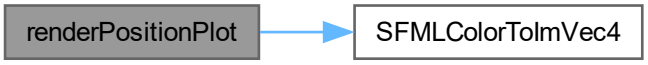
Parameters

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

Definition at line 35 of file GUI.cc.

```
00035                                                                    {
00036     ImGui::Begin("Cars' Position");
00037     ImPlot::SetNextAxesToFit();
00038     if (ImPlot::BeginPlot("Position")) {
00039         ImPlot::SetupLegend(ImPlotLocation_West, ImPlotLegendFlags_Outside);
00040         for (int i = 0; i < cars.size(); ++i) {
00041             if (!paused) {
00042                 cars[i].getPosXVec()->push_back(cars[i].getPosXVec()->back() + 1);
00043                 cars[i].getPosYVec()->push_back(cars[i].getBody()->body->GetPosition().x);
00044             }
00045             std::vector<float> v_axis_crop =
00046                 std::vector<float>(cars[i].getPosXVec()->end() - Config::VELOCITY_ARRAY_SIZE,
00047                                    cars[i].getPosXVec()->end());
00048             std::vector<float> v_values_crop =
00049                 std::vector<float>(cars[i].getPosYVec()->end() - Config::VELOCITY_ARRAY_SIZE,
00050                                    cars[i].getPosYVec()->end());
00051             ImPlot::PushStyleColor(ImPlotCol_Line, SFMLColorToImVec4(cars[i].getBodyColor()));
00052             ImPlot::PlotLine(std::to_string(i).c_str(), &(v_axis_crop[0]), &(v_values_crop[0]),
00053                              Config::VELOCITY_ARRAY_SIZE);
00054             ImPlot::PopStyleColor();
00055         }
00056         ImPlot::EndPlot();
00057     }
00058     ImGui::End();
00059 }
```

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.

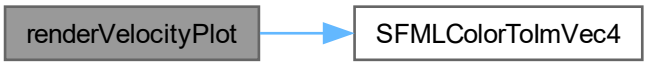
Parameters

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             if (!paused) {
00018                 cars[i].getVelX()->push_back(cars[i].getVelX()->back() + 1);
00019                 cars[i].getVelY()->push_back(cars[i].getVelocity());
00020             }
00021             std::vector<float> v_axis_crop = std::vector<float>(
00022                 cars[i].getVelX()->end() - Config::VELOCITY_ARRAY_SIZE, cars[i].getVelX()->end());
00023             std::vector<float> v_values_crop = std::vector<float>(
00024                 cars[i].getVelY()->end() - Config::VELOCITY_ARRAY_SIZE, cars[i].getVelY()->end());
00025             ImPlot::PushStyleColor(ImPlotCol_Line, SFMLColorToImVec4(cars[i].getBodyColor()));
00026             ImPlot::PlotLine(std::to_string(i).c_str(), &(v_axis_crop[0]), &(v_values_crop[0]),
00027                              Config::VELOCITY_ARRAY_SIZE);
00028             ImPlot::PopStyleColor();
00029         }
00030         ImPlot::EndPlot();
00031     }
00032     ImGui::End();
00033 }
```

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
00010 #define GUI_H
00011
00012 #include <vector>
```

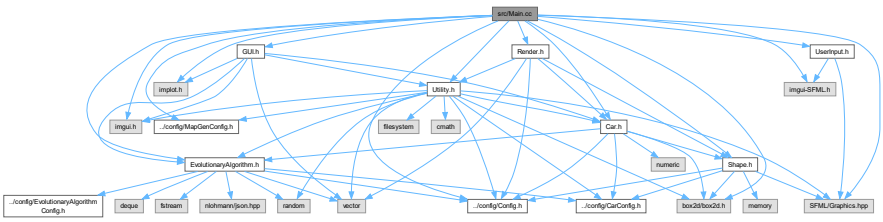
```
00013
00014 #include "imgui.h"
00015 #include "implot.h"
00016
00017 #include "Car.h"
00018 #include "EvolutionaryAlgorithm.h"
00019 #include "Utility.h"
00020
00026 void renderVelocityPlot(std::vector<Car>& cars, bool paused);
00027
00034 void renderPositionPlot(std::vector<Car>& cars, bool paused);
00035
00041 void printEAInfo(EvolutionaryAlgorithm& ea);
00042
00043 #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

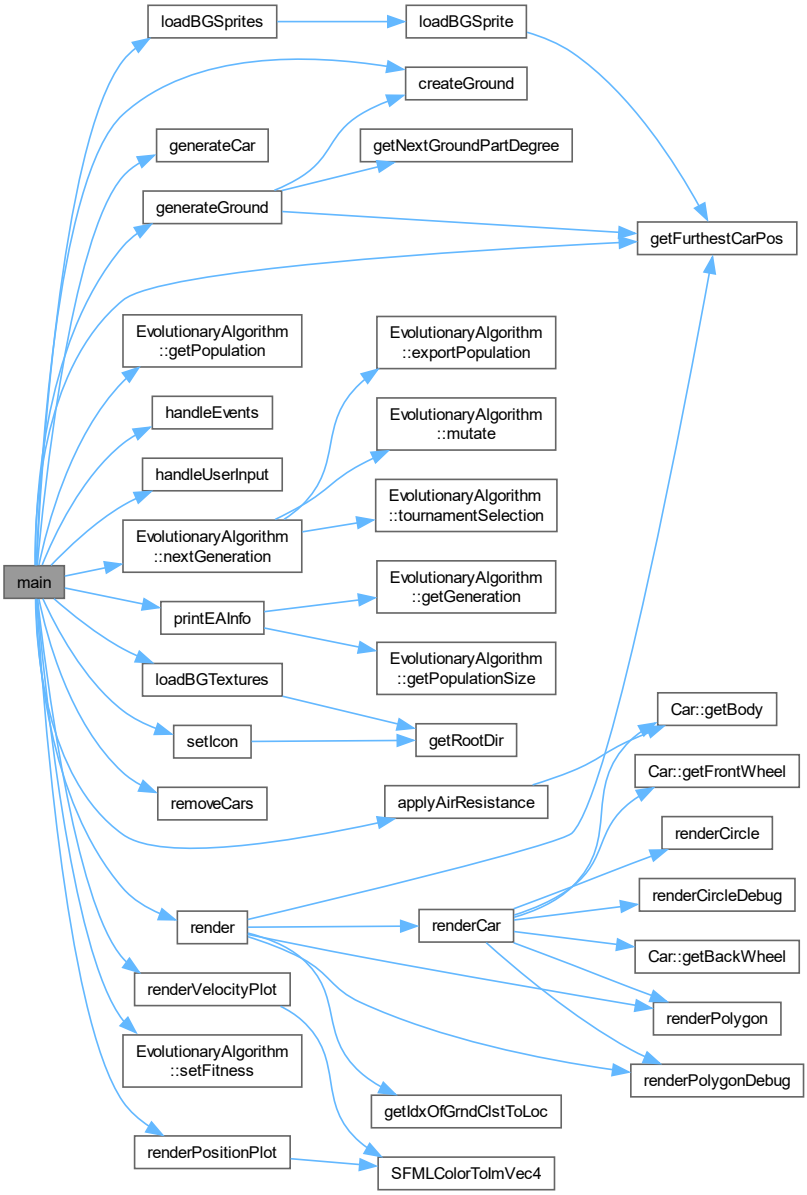
Definition at line 30 of file Main.cc.

```
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)};
00053     Polygon ground =
00054         createGround(world, MapGenConfig::GROUND_STARTING_X, MapGenConfig::GROUND_STARTING_Y,
00055                      groundVertices, sf::Color(18, 36, 35));
00056     groundVector.push_back(ground);
00057 
00058     EvolutionaryAlgorithm ea(EvolutionaryAlgorithmConfig::POPULATION_SIZE, Config::SAVE_TO_FILE);
00059 
00060     for (const Chromosome& chromosome : ea.getPopulation()) {
```

```
00061         cars.push_back(generateCar(world, chromosome));
00062     }
00063
00064     bool paused = false;      // Should we pause the simulation?
00065     bool pauseCheck = true;   // Should we check if the user wants to flip `paused`?
00066     bool nextGen = false;     // Should we generate the next generation?
00067     bool nextGCheck = true;   // Should we check if the user wants to flip `nextGen`?
00068     bool focus = true;        // Is the window in focus? (used to prevent input when not in focus)
00069     int timer = 0;
00070
00071     // Set window icon
00072     setIcon(w);
00073
00074     auto textures = loadBGTextures();
00075     auto sprites = loadBGSprites(textures, cars);
00076
00077     sf::Clock deltaClock;
00079     while (w.isOpen()) {
00080         // Update the world, standard arguments
00081         if (!paused) {
00082             world->Step(1 / 60.0f, 6, 3);
00083             ++timer;
00084             if (timer >= Config::GENERATION_TIME) {
00085                 nextGen = true;
00086             }
00087         }
00088
00089         if (nextGen) {
00090             nextGen = false;
00091             for (int i = 0; i < cars.size(); ++i) {
00092                 ea.setFitness(i, cars[i].getPosX());
00093             }
00094             ea.nextGeneration();
00095             removeCars(world, &cars);
00096             for (const Chromosome& chromosome : ea.getPopulation()) {
00097                 cars.push_back(generateCar(world, chromosome));
00098             }
00099             timer = 0;
00100         }
00101
00102         // Render everything
00103         render(w, sprites, groundVector, cars);
00104
00105         ImGui::SFML::Update(w, deltaClock.restart());
00106
00107         ImGui::PushStyleColor(ImGuiCol_WindowBg, ImVec4(0.071f, 0.141f, 0.137f, 0.5f));
00108
00109         ImGui::SetNextWindowSize(ImVec2(340, 340), ImGuiCond_FirstUseEver);
00110         ImGui::SetNextWindowPos(ImVec2(10, 10), ImGuiCond_FirstUseEver);
00111         renderVelocityPlot(cars, paused);
00112
00113         ImGui::SetNextWindowSize(ImVec2(340, 340), ImGuiCond_FirstUseEver);
00114         ImGui::SetNextWindowPos(ImVec2(930, 10), ImGuiCond_FirstUseEver);
00115         renderPositionPlot(cars, paused);
00116
00117         ImGui::SetNextWindowSize(ImVec2(175, 75), ImGuiCond_FirstUseEver);
00118         ImGui::SetNextWindowPos(ImVec2(543, 10), ImGuiCond_FirstUseEver);
00119         printEAInfo(ea);
00120
00121         ImGui::PopStyleColor();
00122
00123         generateGround(world, &groundVector, cars);
00124
00125         ImGui::SFML::Render(w);
00126
00127         w.display();
00128
00129         // Attach camera to the car's body
00130         sf::View cameraView =
00131             sf::View(sf::Vector2f(getFurthestCarPos(cars).x * Config::PPM,
00132                                   Config::WINDOW_HEIGHT - getFurthestCarPos(cars).y * Config::PPM),
00133                      sf::Vector2f(Config::WINDOW_WIDTH, Config::WINDOW_HEIGHT));
00134         w.setView(cameraView);
00135
00136         // If the camera moves, shift backgrounds accordingly to create a parallax effect
00137         for (int i = 0; i < 5; ++i) {
00138             sprites[i].setPosition(
00139                 cameraView.getCenter().x * (1.0 - 0.2 * i) - Config::WINDOW_WIDTH * (1.4 - 0.1 * i),
00140                 cameraView.getCenter().y - Config::WINDOW_HEIGHT / 2.0);
00141         }
00142
00143         if (!paused) {
00144             for (auto& car : cars) {
00145                 car.getFrontWheel()->body->ApplyTorque(-CarConfig::CAR_TORQUE, false);
00146                 car.getBackWheel()->body->ApplyTorque(-CarConfig::CAR_TORQUE, false);
00147                 applyAirResistance(car);
00148             }
```

```
00149         }
00150
00151         // Display FPS in window title
00152         w.setTitle("EvoRacer, FPS: " + std::to_string((int)ImGui::GetIO().Framerate));
00153
00154         handleEvents(w, pauseCheck, nextGCheck, focus);
00155         handleUserInput(w, paused, pauseCheck, nextGen, nextGCheck, focus);
00156     }
00157
00158     ImPlot::DestroyContext();
00159     ImGui::SFML::Shutdown();
00160
00161     return 0;
00162 }
```

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.

```
00001
00009 #include "box2d/box2d.h"
00010 #include "imgui.h"
00011 #include "imgui-SFML.h"
00012 #include "implot.h"
00013 #include "SFML/Graphics.hpp"
00014
00015 #include "../config/Config.h"
00016 #include "../config/MapGenConfig.h"
00017 #include "Car.h"
00018 #include "EvolutionaryAlgorithm.h"
00019 #include "GUI.h"
00020 #include "Render.h"
00021 #include "Shape.h"
00022 #include "UserInput.h"
00023 #include "Utility.h"
00024
00025 typedef std::shared_ptr<b2World> b2WorldPtr;
00026
00027 // initialize the world as a shared pointer
00028 b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
00029
00030 int main() {
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)};
00053     Polygon ground =
00054         createGround(world, MapGenConfig::GROUND_STARTING_X, MapGenConfig::GROUND_STARTING_Y,
00055                      groundVertices, sf::Color(18, 36, 35));
00056     groundVector.push_back(ground);
00057
00058     EvolutionaryAlgorithm ea(EvolutionaryAlgorithmConfig::POPULATION_SIZE, Config::SAVE_TO_FILE);
00059
00060     for (const Chromosome& chromosome : ea.getPopulation()) {
00061         cars.push_back(generateCar(world, chromosome));
00062     }
00063
00064     bool paused = false;      // Should we pause the simulation?
00065     bool pauseCheck = true;   // Should we check if the user wants to flip `paused`?
00066     bool nextGen = false;     // Should we generate the next generation?
00067     bool nextGCheck = true;   // Should we check if the user wants to flip `nextGen`?
00068     bool focus = true;        // Is the window in focus? (used to prevent input when not in focus)
00069     int timer = 0;
00070
00071     // Set window icon
00072     setIcon(w);
00073
```

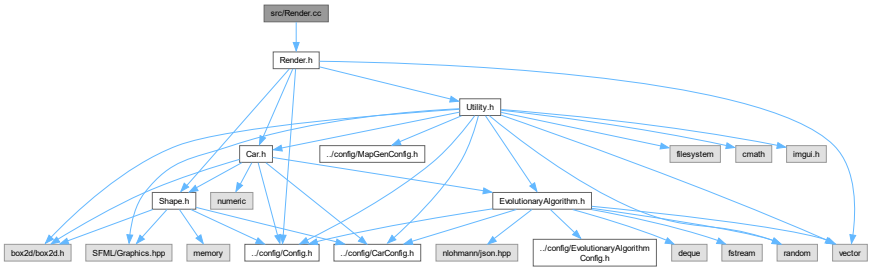
```
00074     auto textures = loadBGTextures();
00075     auto sprites = loadBGSprites(textures, cars);
00076
00077     sf::Clock deltaClock;
00079     while (w.isOpen()) {
00080         // Update the world, standard arguments
00081         if (!paused) {
00082             world->Step(1 / 60.0f, 6, 3);
00083             ++timer;
00084             if (timer >= Config::GENERATION_TIME) {
00085                 nextGen = true;
00086             }
00087         }
00088
00089         if (nextGen) {
00090             nextGen = false;
00091             for (int i = 0; i < cars.size(); ++i) {
00092                 ea.setFitness(i, cars[i].getPosX());
00093             }
00094             ea.nextGeneration();
00095             removeCars(world, &cars);
00096             for (const Chromosome& chromosome : ea.getPopulation()) {
00097                 cars.push_back(generateCar(world, chromosome));
00098             }
00099             timer = 0;
00100         }
00101
00102         // Render everything
00103         render(w, sprites, groundVector, cars);
00104
00105         ImGui::SFML::Update(w, deltaClock.restart());
00106
00107         ImGui::PushStyleColor(ImGuiCol_WindowBg, ImVec4(0.071f, 0.141f, 0.137f, 0.5f));
00108
00109         ImGui::SetNextWindowSize(ImVec2(340, 340), ImGuiCond_FirstUseEver);
00110         ImGui::SetNextWindowPos(ImVec2(10, 10), ImGuiCond_FirstUseEver);
00111         renderVelocityPlot(cars, paused);
00112
00113         ImGui::SetNextWindowSize(ImVec2(340, 340), ImGuiCond_FirstUseEver);
00114         ImGui::SetNextWindowPos(ImVec2(930, 10), ImGuiCond_FirstUseEver);
00115         renderPositionPlot(cars, paused);
00116
00117         ImGui::SetNextWindowSize(ImVec2(175, 75), ImGuiCond_FirstUseEver);
00118         ImGui::SetNextWindowPos(ImVec2(543, 10), ImGuiCond_FirstUseEver);
00119         printEAInfo(ea);
00120
00121         ImGui::PopStyleColor();
00122
00123         generateGround(world, &groundVector, cars);
00124
00125         ImGui::SFML::Render(w);
00126
00127         w.display();
00128
00129         // Attach camera to the car's body
00130         sf::View cameraView =
00131             sf::View(sf::Vector2f(getFurthestCarPos(cars).x * Config::PPM,
00132                                   Config::WINDOW_HEIGHT - getFurthestCarPos(cars).y * Config::PPM),
00133                      sf::Vector2f(Config::WINDOW_WIDTH, Config::WINDOW_HEIGHT));
00134         w.setView(cameraView);
00135
00136         // If the camera moves, shift backgrounds accordingly to create a parallax effect
00137         for (int i = 0; i < 5; ++i) {
00138             sprites[i].setPosition(
00139                 cameraView.getCenter().x * (1.0 - 0.2 * i) - Config::WINDOW_WIDTH * (1.4 - 0.1 * i),
00140                 cameraView.getCenter().y - Config::WINDOW_HEIGHT / 2.0);
00141         }
00142
00143         if (!paused) {
00144             for (auto& car : cars) {
00145                 car.getFrontWheel()->body->ApplyTorque(-CarConfig::CAR_TORQUE, false);
00146                 car.getBackWheel()->body->ApplyTorque(-CarConfig::CAR_TORQUE, false);
00147                 applyAirResistance(car);
00148             }
00149         }
00150
00151         // Display FPS in window title
00152         w.setTitle("EvoRacer, FPS: " + std::to_string((int)ImGui::GetIO().Framerate));
00153
00154         handleEvents(w, pauseCheck, nextGCheck, focus);
00155         handleUserInput(w, paused, pauseCheck, nextGen, nextGCheck, focus);
00156     }
00157
00158     ImPlot::DestroyContext();
00159     ImGui::SFML::Shutdown();
00160
00161     return 0;
```

```
00162 }
```

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.

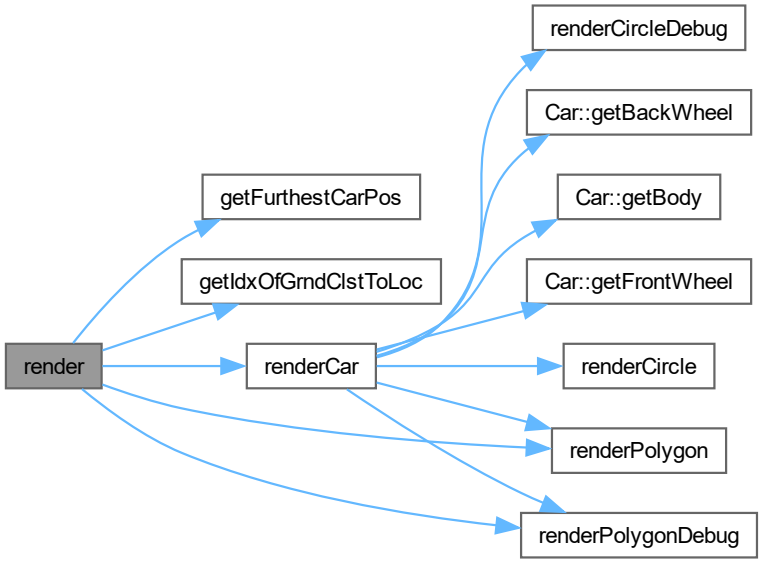
Parameters

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

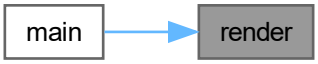
Definition at line 99 of file Render.cc.

```
00100                                                                    {
00101     w.clear();
00102     for (const sf::Sprite &BG : BGs) {
00103         w.draw(BG);
00104     }
00105
00106     int groundBeginIndex = 0;
00107     int centerIndex = getIdxOfGrndClstToLoc(groundVector, getFurthestCarPos(cars).x);
00108     int groundEndIndex = groundVector.size();
00109
00110     if (centerIndex - Config::GROUND_PARTS_RENDERED / 2 > 0) {
00111         groundBeginIndex = centerIndex - Config::GROUND_PARTS_RENDERED / 2;
00112     }
00113     if (centerIndex + Config::GROUND_PARTS_RENDERED / 2 < groundEndIndex) {
00114         groundEndIndex = centerIndex + Config::GROUND_PARTS_RENDERED / 2;
00115     }
00116     std::vector<Polygon> groundSlice(groundVector.begin() + groundBeginIndex,
00117                                      groundVector.begin() + groundEndIndex);
00118
00119     for (Polygon ground : groundSlice) {
00120         renderPolygon(w, &ground);
00121         if (Config::DEBUG) {
00122             renderPolygonDebug(w, &ground);
00123         }
00124     }
00125
00126     // new cars should be rendered behind the old ones
00127     for (int i = cars.size() - 1; i >= 0; --i) {
00128         renderCar(w, cars[i]);
00129     }
00130 }
```

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.

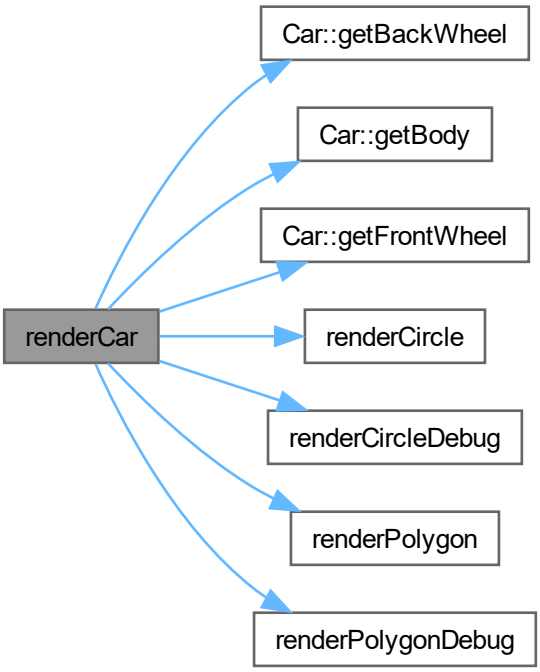
Parameters

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

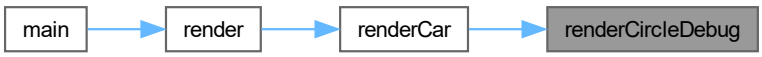
w	SFML's RenderWindow.
circle	Pointer to the circle.

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.

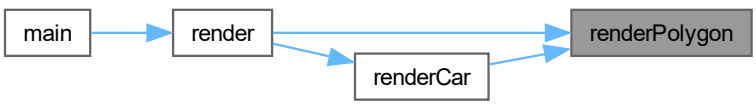
Parameters

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.

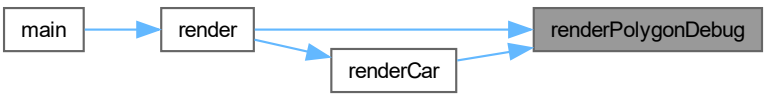
Parameters

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.

```
00001
00010 #include "Render.h"
00011
00012 void renderCircle(sf::RenderWindow &w, Circle *circle) {
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 }
00027
00028 void renderCircleDebug(sf::RenderWindow &w, Circle *circle) {
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 }
00042
00043 void renderPolygon(sf::RenderWindow &w, Polygon *polygon) {
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 }
00065
00066 void renderPolygonDebug(sf::RenderWindow &w, Polygon *polygon) {
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 }
00087
00088 void renderCar(sf::RenderWindow &w, Car car) {
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 }
00098
00099 void render(sf::RenderWindow &w, const std::vector<sf::Sprite> &BGs,
00100             std::vector<Polygon> &groundVector, std::vector<Car> &cars) {
00101     w.clear();
00102     for (const sf::Sprite &BG : BGs) {
00103         w.draw(BG);
00104     }
00105
00106     int groundBeginIndex = 0;
00107     int centerIndex = getIdxOfGrndClstToLoc(groundVector, getFurthestCarPos(cars).x);
00108     int groundEndIndex = groundVector.size();
00109
```

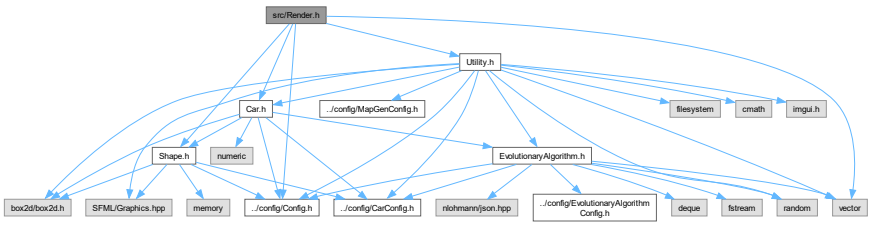
```
00110     if (centerIndex - Config::GROUND_PARTS_RENDERED / 2 > 0) {
00111         groundBeginIndex = centerIndex - Config::GROUND_PARTS_RENDERED / 2;
00112     }
00113     if (centerIndex + Config::GROUND_PARTS_RENDERED / 2 < groundEndIndex) {
00114         groundEndIndex = centerIndex + Config::GROUND_PARTS_RENDERED / 2;
00115     }
00116     std::vector<Polygon> groundSlice(groundVector.begin() + groundBeginIndex,
00117                                      groundVector.begin() + groundEndIndex);
00118
00119     for (Polygon ground : groundSlice) {
00120         renderPolygon(w, &ground);
00121         if (Config::DEBUG) {
00122             renderPolygonDebug(w, &ground);
00123         }
00124     }
00125
00126     // new cars should be rendered behind the old ones
00127     for (int i = cars.size() - 1; i >= 0; --i) {
00128         renderCar(w, cars[i]);
00129     }
00130 }
```

4.25 src/Render.h File Reference

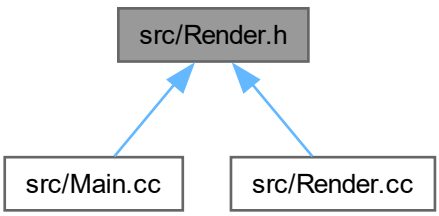
Header file for render function.

```
#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.

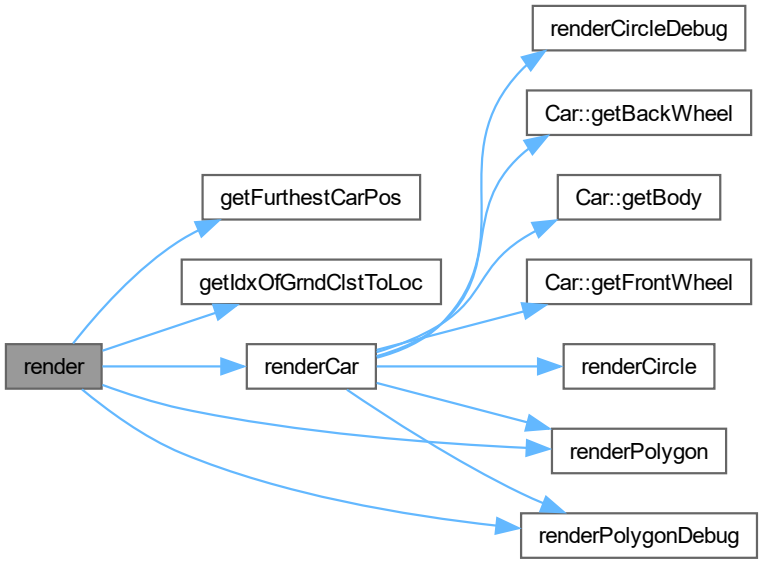
Parameters

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

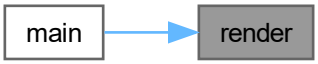
Definition at line 99 of file Render.cc.

```
00100                                                                    {
00101     w.clear();
00102     for (const sf::Sprite &BG : BGs) {
00103         w.draw(BG);
00104     }
00105
00106     int groundBeginIndex = 0;
00107     int centerIndex = getIdxOfGrndClstToLoc(groundVector, getFurthestCarPos(cars).x);
00108     int groundEndIndex = groundVector.size();
00109
00110     if (centerIndex - Config::GROUND_PARTS_RENDERED / 2 > 0) {
00111         groundBeginIndex = centerIndex - Config::GROUND_PARTS_RENDERED / 2;
00112     }
00113     if (centerIndex + Config::GROUND_PARTS_RENDERED / 2 < groundEndIndex) {
00114         groundEndIndex = centerIndex + Config::GROUND_PARTS_RENDERED / 2;
00115     }
00116     std::vector<Polygon> groundSlice(groundVector.begin() + groundBeginIndex,
00117                                      groundVector.begin() + groundEndIndex);
00118
00119     for (Polygon ground : groundSlice) {
00120         renderPolygon(w, &ground);
00121         if (Config::DEBUG) {
00122             renderPolygonDebug(w, &ground);
00123         }
00124     }
00125
00126     // new cars should be rendered behind the old ones
00127     for (int i = cars.size() - 1; i >= 0; --i) {
00128         renderCar(w, cars[i]);
00129     }
00130 }
```

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.

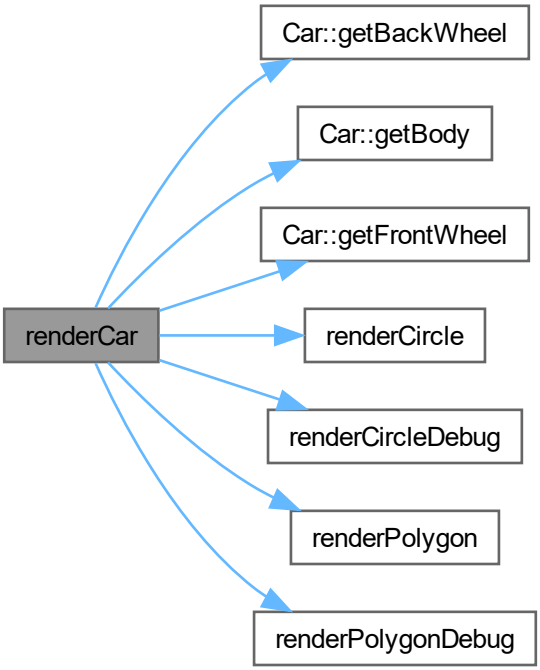
Parameters

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

Definition at line 88 of file Render.cc.

```
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:

4.25.2.3 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.

```
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:

4.25.2.4 renderCircleDebug()

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

Function for rendering a circle's debug information.

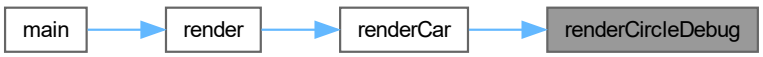
Parameters

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

Definition at line 28 of file Render.cc.

```
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:



4.25.2.5 renderPolygon()

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

Function for rendering a polygon.

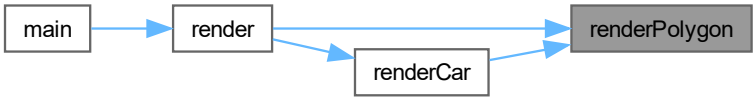
Parameters

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.

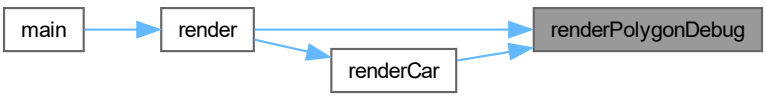
Parameters

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.

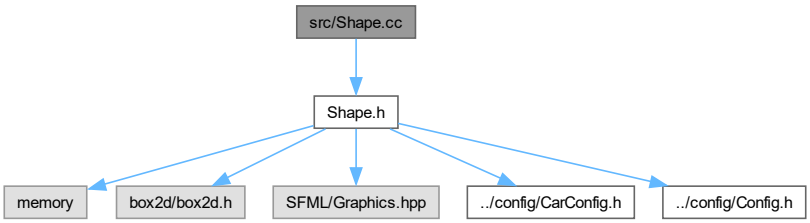
```
00001
00009 #ifndef RENDER_H
00010 #define RENDER_H
00011
00012 #include <vector>
00013
00014 #include "../config/Config.h"
00015 #include "Car.h"
00016 #include "Shape.h"
00017 #include "Utility.h"
00018
00025 void renderCircle(sf::RenderWindow &w, Circle *circle);
00026
00033 void renderCircleDebug(sf::RenderWindow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

```
00034
00041 void renderPolygon(sf::RenderWindow &w, Polygon *polygon);
00042
00049 void renderPolygonDebug(sf::RenderWindow &w, Polygon *polygon);
00050
00057 void renderCar(sf::RenderWindow &w, Car car);
00058
00067 void render(sf::RenderWindow &w, const std::vector<sf::Sprite> &BGs, std::vector<Polygon> &ground,
00068             std::vector<Car> &cars);
00069
00070 #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.

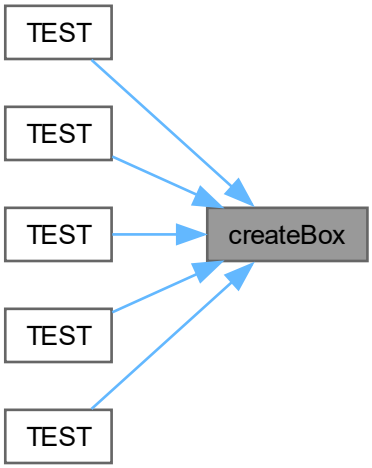
Returns

Box

Definition at line 11 of file Shape.cc.

```
00012                                                {
00013     // Argument validation
00014     if (width <= 0) {
00015         throw std::invalid_argument("Invalid width parameter");
00016     } else if (height <= 0.0f) {
00017         throw std::invalid_argument("Invalid height parameter");
00018     } else if (density <= 0.0f) {
00019         throw std::invalid_argument("Invalid density parameter");
00020     } else if (friction <= 0.0f) {
00021         throw std::invalid_argument("Invalid friction parameter");
00022     }
00023     // Body definition
00024     b2BodyDef boxBodyDef;
00025     boxBodyDef.position.Set(x / Config::PPM, y / Config::PPM);
00026     boxBodyDef.type = b2_dynamicBody;
00027
00028     // Shape definition
00029     b2PolygonShape boxShape;
00030     boxShape.SetAsBox(width / 2 / Config::PPM, height / 2 / Config::PPM);
00031
00032     // Fixture definition
00033     b2FixtureDef fixtureDef;
00034     fixtureDef.density = density;
00035     fixtureDef.friction = friction;
00036     fixtureDef.shape = &boxShape;
00037
00038     // Now we have a body for our Box object
00039     b2Body* boxBody = world->CreateBody(&boxBodyDef);
00040     // Lastly, assign the fixture
00041     boxBody->CreateFixture(&fixtureDef);
00042
00043     return Box{width, height, color, boxBody};
00044 }
```

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.

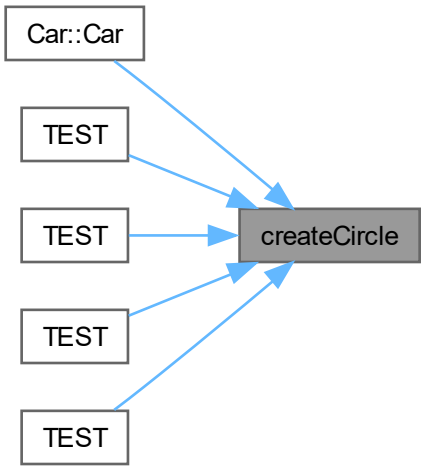
Returns

Circle

Definition at line 65 of file Shape.cc.

```
00066                                                 {
00067     // Argument validation
00068     if (radius <= 0.0f) {
00069         throw std::invalid_argument("Invalid width parameter");
00070     } else if (density <= 0.0f) {
00071         throw std::invalid_argument("Invalid density parameter");
00072     } else if (friction <= 0.0f) {
00073         throw std::invalid_argument("Invalid friction parameter");
00074     }
00075
00076     b2BodyDef boxBodyDef;
00077     boxBodyDef.position.Set(x / Config::PPM, y / Config::PPM);
00078     boxBodyDef.type = b2_dynamicBody;
00079
00080     b2CircleShape circleShape;
00081     circleShape.m_radius = radius / Config::PPM;
00082
00083     b2FixtureDef fixtureDef;
00084     fixtureDef.density = density;
00085     fixtureDef.friction = friction;
00086     fixtureDef.shape = &circleShape;
00087
00088     b2Body* boxBody = world->CreateBody(&boxBodyDef);
00089
00090     boxBody->CreateFixture(&fixtureDef);
00091
00092     return Circle{radius, color, boxBody};
00093 }
```

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.

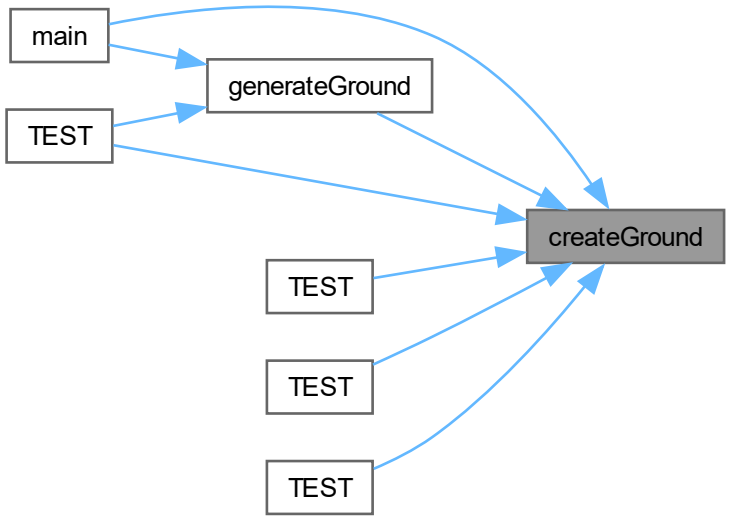
Returns

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.

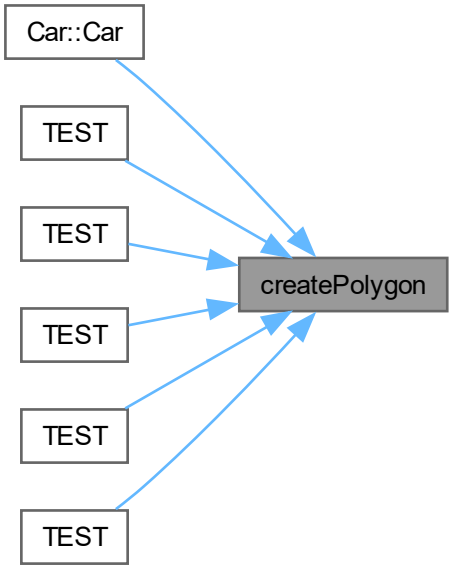
Returns

Polygon

Definition at line 95 of file Shape.cc.

```
00096                                                                                {
00097     // Argument validation
00098     if (vertices.size() < 3 || vertices.size() > CarConfig::CAR_VERTICES) {
00099         throw std::invalid_argument("Invalid vertices size");
00100     } else if (density <= 0.0f) {
00101         throw std::invalid_argument("Invalid density parameter");
00102     } else if (friction <= 0.0f) {
00103         throw std::invalid_argument("Invalid friction parameter");
00104     }
00105     b2BodyDef boxBodyDef;
00106     boxBodyDef.position.Set(x / Config::PPM, y / Config::PPM);
00107     boxBodyDef.type = b2_dynamicBody;
00108
00109     b2PolygonShape boxShape;
00110     boxShape.Set(vertices.data(), vertices.size());
00111
00112     b2FixtureDef fixtureDef;
00113     fixtureDef.density = density;
00114     fixtureDef.friction = friction;
00115     fixtureDef.shape = &boxShape;
00116
00117     b2Body* boxBody = world->CreateBody(&boxBodyDef);
00118
00119     boxBody->CreateFixture(&fixtureDef);
00120
00121     // create a Polygon object with a shared pointer to the b2Body
00122     return Polygon{vertices, color, boxBody};
00123 }
```

Here is the caller graph for this function:



4.28 Shape.cc

Go to the documentation of this file.

```
00001
00009 #include "Shape.h"
00010
00011 Box createBox(const b2WorldPtr& world, float x, float y, float width, float height, float density,
00012                float friction, sf::Color color) {
00013     // Argument validation
00014     if (width <= 0) {
00015         throw std::invalid_argument("Invalid width parameter");
00016     } else if (height <= 0.0f) {
00017         throw std::invalid_argument("Invalid height parameter");
00018     } else if (density <= 0.0f) {
00019         throw std::invalid_argument("Invalid density parameter");
00020     } else if (friction <= 0.0f) {
00021         throw std::invalid_argument("Invalid friction parameter");
00022     }
00023     // Body definition
00024     b2BodyDef boxBodyDef;
00025     boxBodyDef.position.Set(x / Config::PPM, y / Config::PPM);
00026     boxBodyDef.type = b2_dynamicBody;
00027
00028     // Shape definition
00029     b2PolygonShape boxShape;
00030     boxShape.SetAsBox(width / 2 / Config::PPM, height / 2 / Config::PPM);
00031
00032     // Fixture definition
00033     b2FixtureDef fixtureDef;
00034     fixtureDef.density = density;
00035     fixtureDef.friction = friction;
00036     fixtureDef.shape = &boxShape;
00037
00038     // Now we have a body for our Box object
00039     b2Body* boxBody = world->CreateBody(&boxBodyDef);
00040     // Lastly, assign the fixture
00041     boxBody->CreateFixture(&fixtureDef);
00042
00043     return Box{width, height, color, boxBody};
00044 }
```

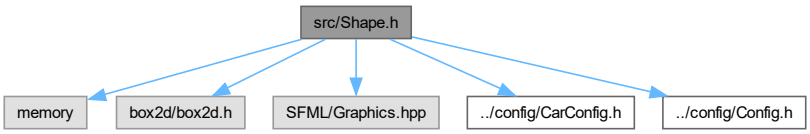
```
00045
00046 Polygon createGround(const b2WorldPtr& world, float x, float y, const std::vector<b2Vec2>& vertices,
00047                       sf::Color color) {
00048     // Argument validation
00049     if (vertices.size() < 3) {
00050         throw std::invalid_argument("Invalid number of vertices");
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 }
00064
00065 Circle createCircle(const b2WorldPtr& world, float x, float y, float radius, float density,
00066                     float friction, sf::Color color) {
00067     // Argument validation
00068     if (radius <= 0.0f) {
00069         throw std::invalid_argument("Invalid width parameter");
00070     } else if (density <= 0.0f) {
00071         throw std::invalid_argument("Invalid density parameter");
00072     } else if (friction <= 0.0f) {
00073         throw std::invalid_argument("Invalid friction parameter");
00074     }
00075
00076     b2BodyDef boxBodyDef;
00077     boxBodyDef.position.Set(x / Config::PPM, y / Config::PPM);
00078     boxBodyDef.type = b2_dynamicBody;
00079
00080     b2CircleShape circleShape;
00081     circleShape.m_radius = radius / Config::PPM;
00082
00083     b2FixtureDef fixtureDef;
00084     fixtureDef.density = density;
00085     fixtureDef.friction = friction;
00086     fixtureDef.shape = &circleShape;
00087
00088     b2Body* boxBody = world->CreateBody(&boxBodyDef);
00089
00090     boxBody->CreateFixture(&fixtureDef);
00091
00092     return Circle{radius, color, boxBody};
00093 }
00094
00095 Polygon createPolygon(const b2WorldPtr& world, float x, float y, std::vector<b2Vec2> vertices,
00096                       float density, float friction, sf::Color color) {
00097     // Argument validation
00098     if (vertices.size() < 3 || vertices.size() > CarConfig::CAR_VERTICES) {
00099         throw std::invalid_argument("Invalid vertices size");
00100     } else if (density <= 0.0f) {
00101         throw std::invalid_argument("Invalid density parameter");
00102     } else if (friction <= 0.0f) {
00103         throw std::invalid_argument("Invalid friction parameter");
00104     }
00105     b2BodyDef boxBodyDef;
00106     boxBodyDef.position.Set(x / Config::PPM, y / Config::PPM);
00107     boxBodyDef.type = b2_dynamicBody;
00108
00109     b2PolygonShape boxShape;
00110     boxShape.Set(vertices.data(), vertices.size());
00111
00112     b2FixtureDef fixtureDef;
00113     fixtureDef.density = density;
00114     fixtureDef.friction = friction;
00115     fixtureDef.shape = &boxShape;
00116
00117     b2Body* boxBody = world->CreateBody(&boxBodyDef);
00118
00119     boxBody->CreateFixture(&fixtureDef);
00120
00121     // create a Polygon object with a shared pointer to the b2Body
00122     return Polygon{vertices, color, boxBody};
00123 }
```

4.29 src/Shape.h File Reference

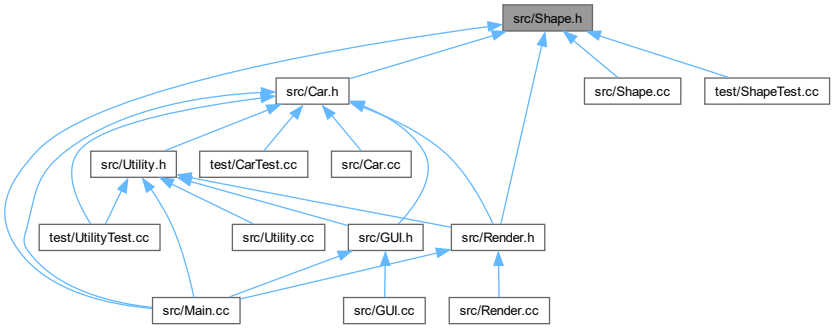
Header file for functions for creating Box2D objects.

```
#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.

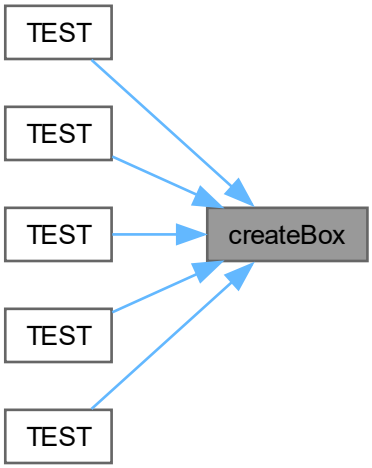
Returns

Box

Definition at line 11 of file Shape.cc.

```
00012                                                {
00013     // Argument validation
00014     if (width <= 0) {
00015         throw std::invalid_argument("Invalid width parameter");
00016     } else if (height <= 0.0f) {
00017         throw std::invalid_argument("Invalid height parameter");
00018     } else if (density <= 0.0f) {
00019         throw std::invalid_argument("Invalid density parameter");
00020     } else if (friction <= 0.0f) {
00021         throw std::invalid_argument("Invalid friction parameter");
00022     }
00023     // Body definition
00024     b2BodyDef boxBodyDef;
00025     boxBodyDef.position.Set(x / Config::PPM, y / Config::PPM);
00026     boxBodyDef.type = b2_dynamicBody;
00027
00028     // Shape definition
00029     b2PolygonShape boxShape;
00030     boxShape.SetAsBox(width / 2 / Config::PPM, height / 2 / Config::PPM);
00031
00032     // Fixture definition
00033     b2FixtureDef fixtureDef;
00034     fixtureDef.density = density;
00035     fixtureDef.friction = friction;
00036     fixtureDef.shape = &boxShape;
00037
00038     // Now we have a body for our Box object
00039     b2Body* boxBody = world->CreateBody(&boxBodyDef);
00040     // Lastly, assign the fixture
00041     boxBody->CreateFixture(&fixtureDef);
00042
00043     return Box{width, height, color, boxBody};
00044 }
```

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.

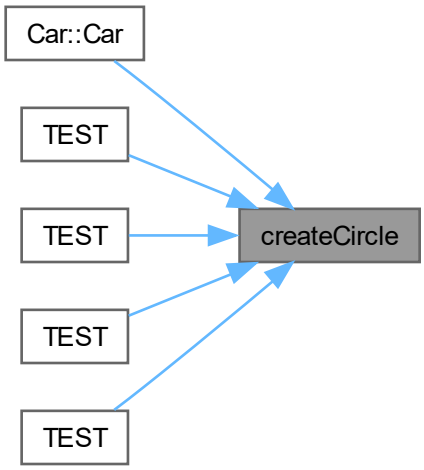
Returns

Circle

Definition at line 65 of file Shape.cc.

```
00066                                                   {
00067     // Argument validation
00068     if (radius <= 0.0f) {
00069         throw std::invalid_argument("Invalid width parameter");
00070     } else if (density <= 0.0f) {
00071         throw std::invalid_argument("Invalid density parameter");
00072     } else if (friction <= 0.0f) {
00073         throw std::invalid_argument("Invalid friction parameter");
00074     }
00075
00076     b2BodyDef boxBodyDef;
00077     boxBodyDef.position.Set(x / Config::PPM, y / Config::PPM);
00078     boxBodyDef.type = b2_dynamicBody;
00079
00080     b2CircleShape circleShape;
00081     circleShape.m_radius = radius / Config::PPM;
00082
00083     b2FixtureDef fixtureDef;
00084     fixtureDef.density = density;
00085     fixtureDef.friction = friction;
00086     fixtureDef.shape = &circleShape;
00087
00088     b2Body* boxBody = world->CreateBody(&boxBodyDef);
00089
00090     boxBody->CreateFixture(&fixtureDef);
00091
00092     return Circle{radius, color, boxBody};
00093 }
```

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.

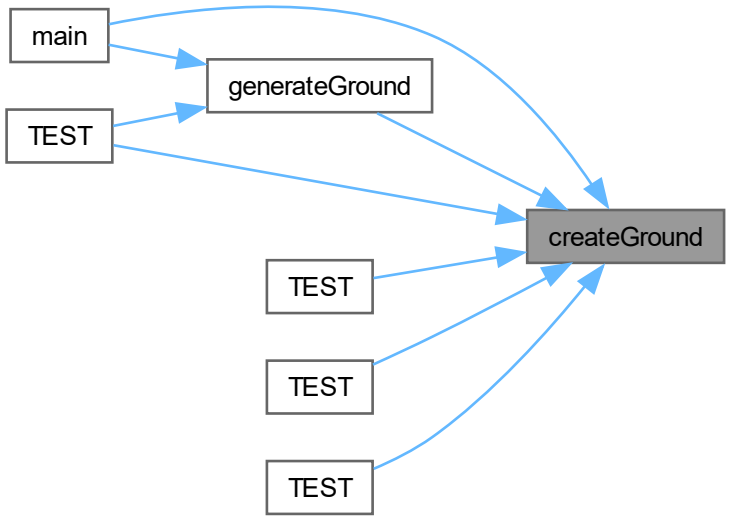
Returns

Polygon

Definition at line 46 of file Shape.cc.

```
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:



4.29.3.4 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.

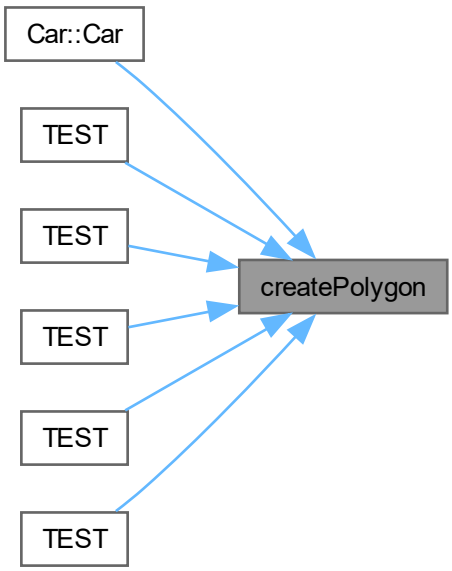
Returns

Polygon

Definition at line 95 of file Shape.cc.

```
00096                                                                                {
00097     // Argument validation
00098     if (vertices.size() < 3 || vertices.size() > CarConfig::CAR_VERTICES) {
00099         throw std::invalid_argument("Invalid vertices size");
00100     } else if (density <= 0.0f) {
00101         throw std::invalid_argument("Invalid density parameter");
00102     } else if (friction <= 0.0f) {
00103         throw std::invalid_argument("Invalid friction parameter");
00104     }
00105     b2BodyDef boxBodyDef;
00106     boxBodyDef.position.Set(x / Config::PPM, y / Config::PPM);
00107     boxBodyDef.type = b2_dynamicBody;
00108
00109     b2PolygonShape boxShape;
00110     boxShape.Set(vertices.data(), vertices.size());
00111
00112     b2FixtureDef fixtureDef;
00113     fixtureDef.density = density;
00114     fixtureDef.friction = friction;
00115     fixtureDef.shape = &boxShape;
00116
00117     b2Body* boxBody = world->CreateBody(&boxBodyDef);
00118
00119     boxBody->CreateFixture(&fixtureDef);
00120
00121     // create a Polygon object with a shared pointer to the b2Body
00122     return Polygon{vertices, color, boxBody};
00123 }
```

Here is the caller graph for this function:



4.30 Shape.h

Go to the documentation of this file.

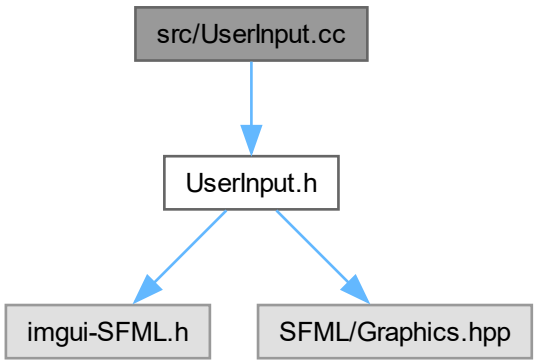
```
00001
00009 #ifndef SHAPE_H
00010 #define SHAPE_H
00011
00012 #include <memory>
00013
00014 #include "box2d/box2d.h"
00015 #include "SFML/Graphics.hpp"
00016
00017 #include "../config/CarConfig.h"
00018 #include "../config/Config.h"
00019
00023 struct Box {
00024     float width{};
00025     float height{};
00026     sf::Color color;
00027     b2Body* body{};
00028 };
00029
00033 struct Circle {
00034     float radius{};
00035     sf::Color color;
00036     b2Body* body{};
00037 };
00038
00042 struct Polygon {
00043     std::vector<b2Vec2> vertices;
00044     sf::Color color;
00045     b2Body* body;
00046 };
00047
00048 typedef std::shared_ptr<b2World> b2WorldPtr;
00049
00063 Box createBox(const b2WorldPtr& world, float x, float y, float width, float height, float density,
00064               float friction, sf::Color color);
00065
00076 Polygon createGround(const b2WorldPtr& world, float x, float y, const std::vector<b2Vec2>& vertices,
```

```
00077                     sf::Color color);
00078
00091 Circle createCircle(const b2WorldPtr& world, float x, float y, float radius, float density,
00092                     float friction, sf::Color color);
00093
00106 Polygon createPolygon(const b2WorldPtr& world, float x, float y, std::vector<b2Vec2> vertices,
00107                      float density, float friction, sf::Color color);
00108
00109 #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.

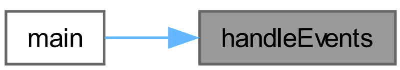
Parameters

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.

```
00012                                                       {
00013     if (focus) {
00014         if (sf::Keyboard::isKeyPressed(sf::Keyboard::Q) ||
00015             sf::Keyboard::isKeyPressed(sf::Keyboard::Escape)) {
00016             // Close the window
00017             w.close();
00018         }
00019
00020         if (sf::Keyboard::isKeyPressed(sf::Keyboard::P) ||
00021             sf::Keyboard::isKeyPressed(sf::Keyboard::Space)) {
00022             // Pause the simulation
00023             if (pause_check) {
00024                 paused = !paused;
00025                 pause_check = false;
00026             }
00027         }
00028         if (sf::Keyboard::isKeyPressed(sf::Keyboard::N)) {
00029             // Generate the next generation
00030             if (nxt_g_check) {
00031                 next_gen = true;
00032                 nxt_g_check = false;
00033             }
00034         }
00035     }
00036 }
```

Here is the caller graph for this function:

4.32 UserInput.cc

Go to the documentation of this file.

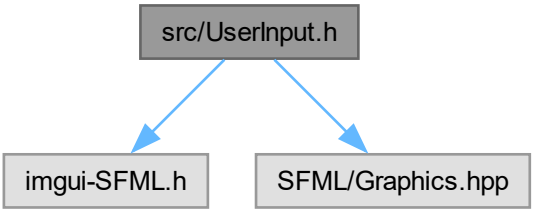
```
00001
00009 #include "UserInput.h"
00010
00011 void handleUserInput(sf::RenderWindow &w, bool &paused, bool &pause_check, bool &next_gen,
00012                      bool &nxt_g_check, bool &focus) {
00013     if (focus) {
00014         if (sf::Keyboard::isKeyPressed(sf::Keyboard::Q) ||
00015             sf::Keyboard::isKeyPressed(sf::Keyboard::Escape)) {
00016             // Close the window
00017             w.close();
00018         }
00019
00020         if (sf::Keyboard::isKeyPressed(sf::Keyboard::P) ||
00021             sf::Keyboard::isKeyPressed(sf::Keyboard::Space)) {
00022             // Pause the simulation
00023             if (pause_check) {
00024                 paused = !paused;
00025                 pause_check = false;
00026             }
00027         }
00028         if (sf::Keyboard::isKeyPressed(sf::Keyboard::N)) {
00029             // Generate the next generation
00030             if (nxt_g_check) {
00031                 next_gen = true;
00032                 nxt_g_check = false;
00033             }
00034         }
00035     }
00036 }
00037
00038 void handleEvents(sf::RenderWindow &w, bool &pause_check, bool &nxt_g_check, bool &focus) {
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 }
```

4.33 src/UserInput.h File Reference

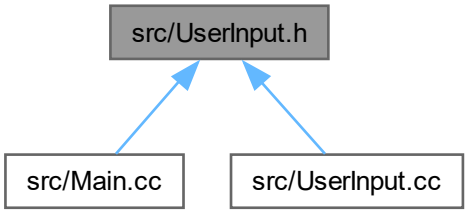
Header for a file containing user input functions.

```
#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.

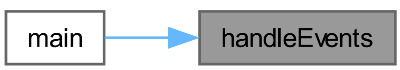
Parameters

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.

```
00012                                                       {
00013     if (focus) {
00014         if (sf::Keyboard::isKeyPressed(sf::Keyboard::Q) ||
00015             sf::Keyboard::isKeyPressed(sf::Keyboard::Escape)) {
00016             // Close the window
00017             w.close();
00018         }
00019
00020         if (sf::Keyboard::isKeyPressed(sf::Keyboard::P) ||
00021             sf::Keyboard::isKeyPressed(sf::Keyboard::Space)) {
00022             // Pause the simulation
00023             if (pause_check) {
00024                 paused = !paused;
00025                 pause_check = false;
00026             }
00027         }
00028         if (sf::Keyboard::isKeyPressed(sf::Keyboard::N)) {
00029             // Generate the next generation
00030             if (nxt_g_check) {
00031                 next_gen = true;
00032                 nxt_g_check = false;
00033             }
00034         }
00035     }
00036 }
```

Here is the caller graph for this function:

4.34 UserInput.h

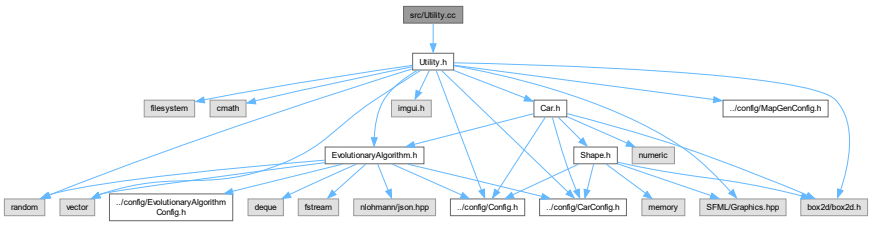
Go to the documentation of this file.

```
00001
00009 #ifndef USER_INPUT_H
00010 #define USER_INPUT_H
00011
00012 #include "imgui-SFML.h"
00013 #include "SFML/Graphics.hpp"
00014
00025 void handleUserInput(sf::RenderWindow &w, bool &paused, bool &pause_check, bool &next_gen,
00026                      bool &nxt_g_check, bool &focus);
00027
00036 void handleEvents(sf::RenderWindow &w, bool &pause_check, bool &nxt_g_check, bool &focus);
00037
00038 #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

2023-06-07

Definition in file Utility.cc.

4.35.2 Function Documentation

4.35.2.1 applyAirResistance()

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

Simplified air drag.

Parameters

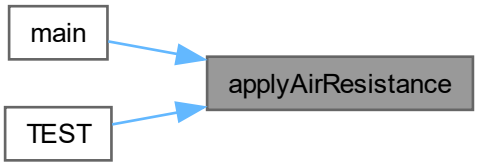
car	Car to apply air resistance to.

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(-sqrt(CarConfig::AIR_RES_FACTOR) * car.getBody()->body->GetLinearVelocity().x *
00021                    abs(car.getBody()->body->GetLinearVelocity().x),
00022                -sqrt(CarConfig::AIR_RES_FACTOR) * 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:



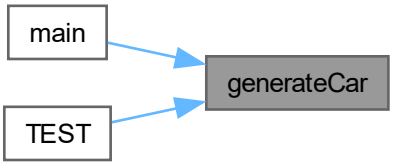
4.35.2.2 generateCar()

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

Definition at line 66 of file Utility.cc.

```
00066                                                                   {
00067     std::random_device rd;
00068     std::mt19937 gen(rd());
00069     std::uniform_int_distribution<> rgb_value(50, 200);
00070
00071     sf::Color bodyColor = sf::Color(rgb_value(gen), rgb_value(gen), rgb_value(gen));
00072     sf::Color wheelColor = sf::Color(rgb_value(gen), rgb_value(gen), rgb_value(gen));
00073
00074     return {world,
00075             MapGenConfig::CAR_STARTING_X,
00076             MapGenConfig::CAR_STARTING_Y,
00077             chromosome,
00078             bodyColor,
00079             wheelColor};
00080 }
```

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.

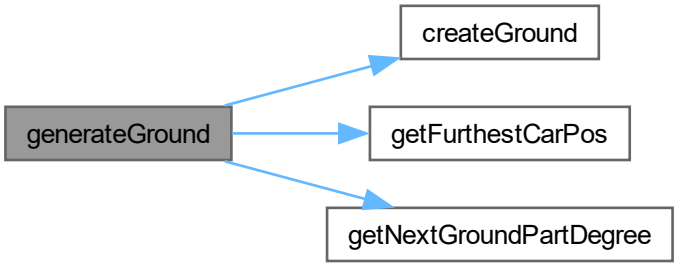
Parameters

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

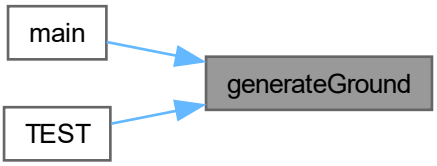
Definition at line 27 of file Utility.cc.

```
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

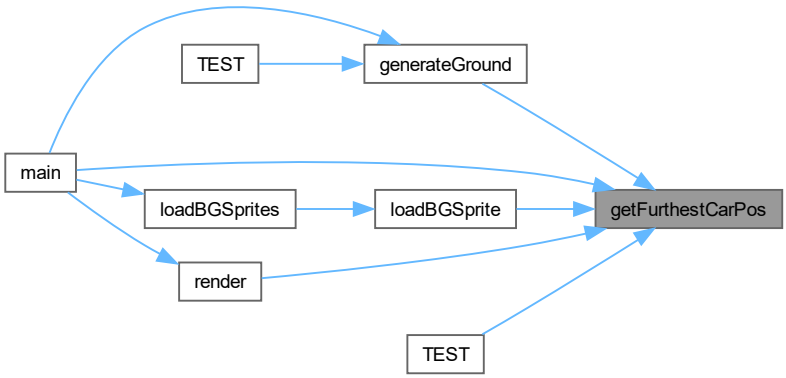
b2Vec2 Position of the furthest car.

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

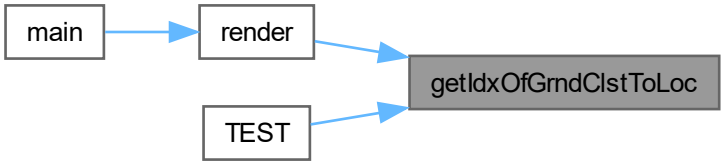
int Index of the ground element.

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.

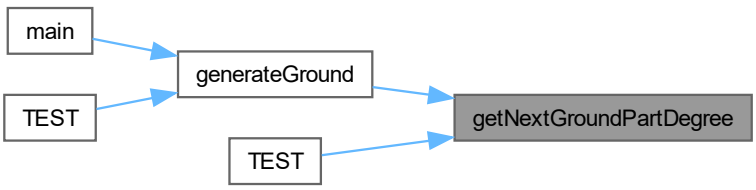
Returns

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.

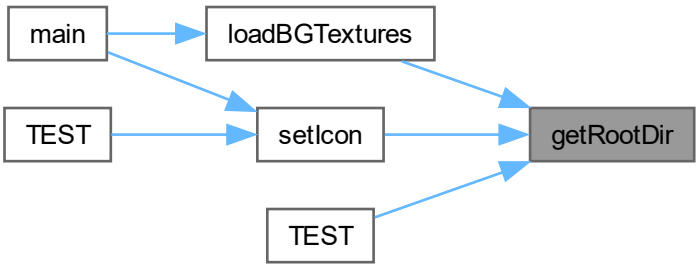
Returns

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	

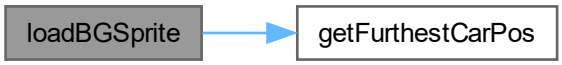
Returns

sf::Sprite

Definition at line 147 of file Utility.cc.

```
00147                                                                    {
00148     sf::Sprite sprite(*texture);
00149     sprite.setScale(sf::Vector2f(Config::WINDOW_WIDTH / sprite.getLocalBounds().width,
00150                                  Config::WINDOW_HEIGHT / sprite.getLocalBounds().height));
00151     sprite.setTextureRect(sf::IntRect(0, 0, 256 * Config::WINDOW_WIDTH, Config::WINDOW_HEIGHT));
00152     sprite.setPosition(
00153         sf::Vector2f(getFurthestCarPos(cars).x * Config::PPM, 0.5 * Config::WINDOW_HEIGHT) -
00154         sf::Vector2f(Config::WINDOW_WIDTH / 2.0f, Config::WINDOW_HEIGHT / 2.0f));
00155     return sprite;
00156 }
```

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>

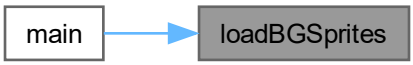
Definition at line 158 of file Utility.cc.

```
00159                                                                    {
00160     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

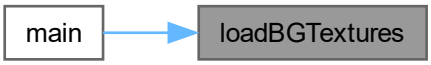
std::vector<sf::Texture*>

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.

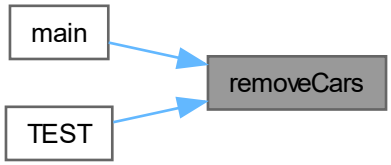
Parameters

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.

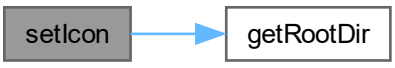
Parameters

window	

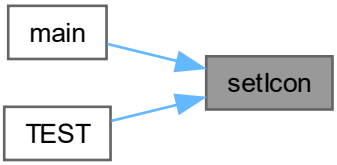
Definition at line 126 of file Utility.cc.

```
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.

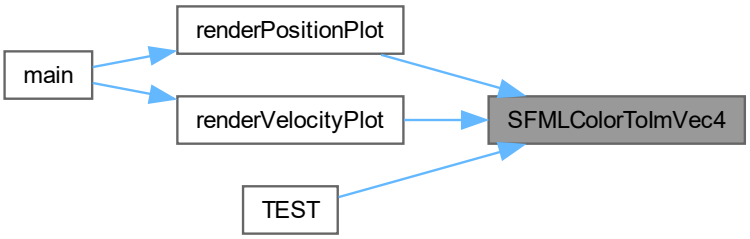
Returns

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.

```
00001
00009 #include "Utility.h"
00010
00011 void applyAirResistance(Car car) {
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(-sqrt(CarConfig::AIR_RES_FACTOR) * car.getBody()->body->GetLinearVelocity().x *
00021                    abs(car.getBody()->body->GetLinearVelocity().x),
00022                -sqrt(CarConfig::AIR_RES_FACTOR) * car.getBody()->body->GetLinearVelocity().y *
00023                    abs(car.getBody()->body->GetLinearVelocity().y)),
00024         true);
00025 }
00026
00027 void generateGround(const b2WorldPtr& world, std::vector<Polygon>* groundVector,
00028                     const std::vector<Car>& cars) {
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 }
00050
00051 float getNextGroundPartDegree() {
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 }
00065
00066 Car generateCar(const b2WorldPtr& world, const Chromosome& chromosome) {
00067     std::random_device rd;
00068     std::mt19937 gen(rd());
00069     std::uniform_int_distribution<> rgb_value(50, 200);
00070
00071     sf::Color bodyColor = sf::Color(rgb_value(gen), rgb_value(gen), rgb_value(gen));
00072     sf::Color wheelColor = sf::Color(rgb_value(gen), rgb_value(gen), rgb_value(gen));
00073
00074     return {world,
00075             MapGenConfig::CAR_STARTING_X,
00076             MapGenConfig::CAR_STARTING_Y,
00077             chromosome,
00078             bodyColor,
00079             wheelColor};
00080 }
00081
00082 ImVec4 SFMLColorToImVec4(sf::Color color) {
00083     return {color.r / 255.0f, color.g / 255.0f, color.b / 255.0f, color.a / 255.0f};
00084 }
00085
00086 b2Vec2 getFurthestCarPos(const std::vector<Car>& cars) {
00087     float furthestCarX = 0;
00088     float furthestCarY = 0;
00089     for (auto car : cars) {
```

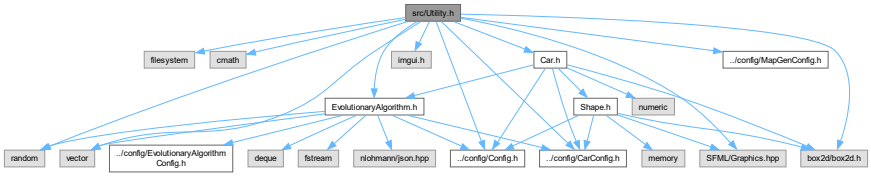
```
00090         float currentCarX = car.getBody()->body->GetPosition().x;
00091         float currentCarY = car.getBody()->body->GetPosition().y;
00092         if (currentCarX > furthestCarX) {
00093             furthestCarX = currentCarX;
00094             furthestCarY = currentCarY;
00095         }
00096     }
00097     return {furthestCarX, furthestCarY};
00098 }
00099
00100 int getIdxOfGrndClstToLoc(std::vector<Polygon> ground, float x) {
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 }
00110
00111 void removeCars(const b2WorldPtr& world, std::vector<Car>* cars) {
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 }
00119
00120 std::filesystem::path getRootDir() {
00121     std::filesystem::path filePath = std::filesystem::path(__FILE__);
00122     std::filesystem::path dirPath = filePath.parent_path();
00123     return dirPath;
00124 }
00125
00126 void setIcon(sf::RenderWindow& window) {
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 }
00133
00134 std::vector<sf::Texture*> loadBGTextures() {
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 }
00146
00147 sf::Sprite loadBGSprite(sf::Texture* texture, const std::vector<Car>& cars) {
00148     sf::Sprite sprite(*texture);
00149     sprite.setScale(sf::Vector2f(Config::WINDOW_WIDTH / sprite.getLocalBounds().width,
00150                                  Config::WINDOW_HEIGHT / sprite.getLocalBounds().height));
00151     sprite.setTextureRect(sf::IntRect(0, 0, 256 * Config::WINDOW_WIDTH, Config::WINDOW_HEIGHT));
00152     sprite.setPosition(
00153         sf::Vector2f(getFurthestCarPos(cars).x * Config::PPM, 0.5 * Config::WINDOW_HEIGHT) -
00154         sf::Vector2f(Config::WINDOW_WIDTH / 2.0f, Config::WINDOW_HEIGHT / 2.0f));
00155     return sprite;
00156 }
00157
00158 std::vector<sf::Sprite> loadBGSprites(std::vector<sf::Texture*> textures,
00159                                       const std::vector<Car>& cars) {
00160     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 }
```

4.37 src/Utility.h File Reference

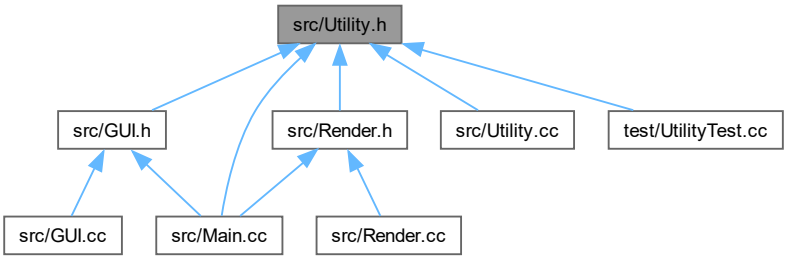
Header for a file containing utility functions.

```
#include <filesystem>
#include <cmath>
#include <random>
#include <vector>
#include "box2d/box2d.h"
#include "imgui.h"
#include "SFML/Graphics.hpp"
#include "EvolutionaryAlgorithm.h"
#include "../config/CarConfig.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

2023-06-07

Definition in file Utility.h.

4.37.2 Typedef Documentation

4.37.2.1 b2WorldPtr

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

Definition at line 27 of file Utility.h.

4.37.3 Function Documentation

4.37.3.1 applyAirResistance()

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

Simplified air drag.

Parameters

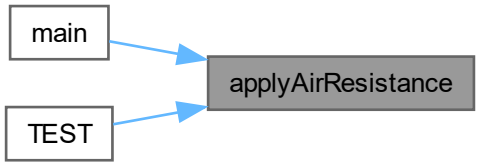
car	Car to apply air resistance to.

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(-sqrt(CarConfig::AIR_RES_FACTOR) * car.getBody()->body->GetLinearVelocity().x *
00021                    abs(car.getBody()->body->GetLinearVelocity().x),
00022                -sqrt(CarConfig::AIR_RES_FACTOR) * 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()

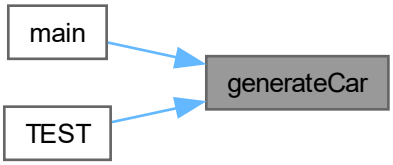
```
Car generateCar (
             const b2WorldPtr & world,
             const Chromosome & chromosome )
```

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);
```

```
00070
00071     sf::Color bodyColor = sf::Color(rgb_value(gen), rgb_value(gen), rgb_value(gen));
00072     sf::Color wheelColor = sf::Color(rgb_value(gen), rgb_value(gen), rgb_value(gen));
00073
00074     return {world,
00075             MapGenConfig::CAR_STARTING_X,
00076             MapGenConfig::CAR_STARTING_Y,
00077             chromosome,
00078             bodyColor,
00079             wheelColor};
00080 }
```

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

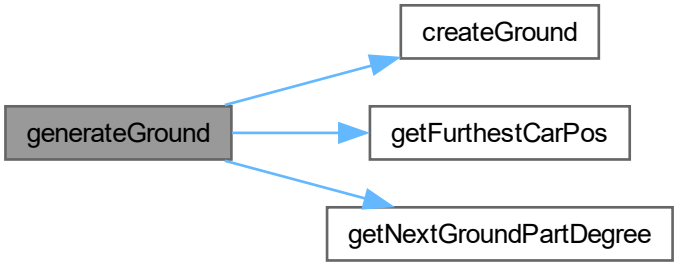
world	A shared pointer to the box2d world.
groundVector	A vector of ground polygons.
cars	A vector of cars.

Definition at line 27 of file Utility.cc.

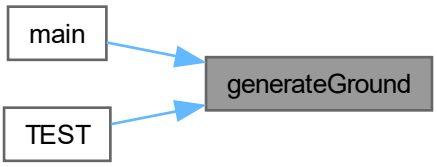
```
00028                                           {
00029     Polygon lastGround = groundVector->back();
00030     if (lastGround.vertices[1].x * Config::PPM <
00031         getFurthestCarPos(cars).x * Config::PPM + MapGenConfig::GENERATE_DISTANCE) {
00032         float degree = getNextGroundPartDegree();
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:



4.37.3.4 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.

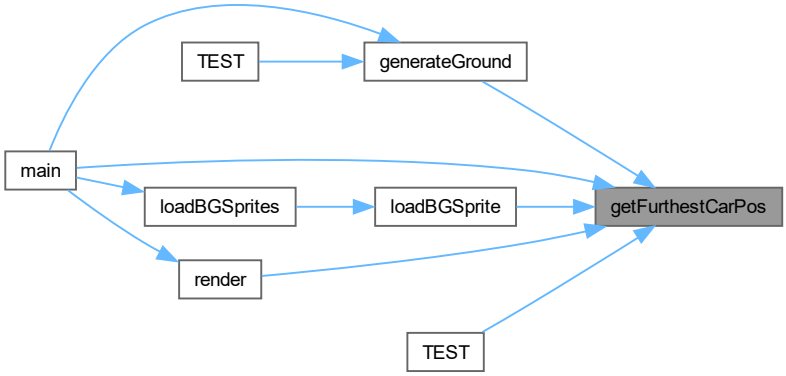
Returns

b2Vec2 Position of the furthest car.

Definition at line 86 of file Utility.cc.

```
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:



4.37.3.5 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

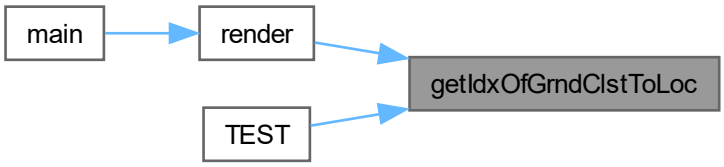
int Index of the ground element.

Definition at line 100 of file Utility.cc.

```
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:



4.37.3.6 getNextGroundPartDegree()

```
float getNextGroundPartDegree ( )
```

Get the angle of the next ground part.

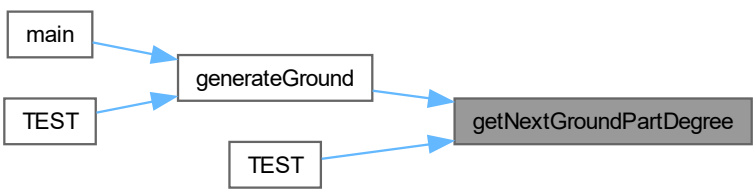
Returns

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.

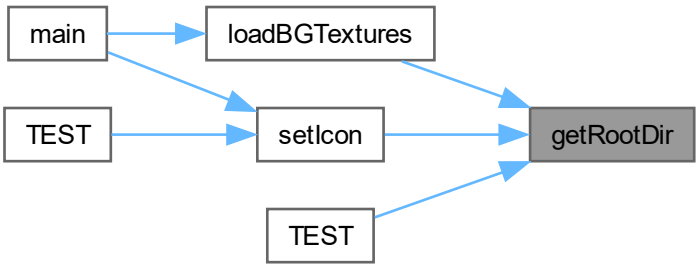
Returns

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	

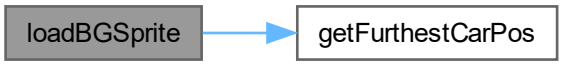
Returns

sf::Sprite

Definition at line 147 of file Utility.cc.

```
00147                                                                   {
00148     sf::Sprite sprite(*texture);
00149     sprite.setScale(sf::Vector2f(Config::WINDOW_WIDTH / sprite.getLocalBounds().width,
00150                                  Config::WINDOW_HEIGHT / sprite.getLocalBounds().height));
00151     sprite.setTextureRect(sf::IntRect(0, 0, 256 * Config::WINDOW_WIDTH, Config::WINDOW_HEIGHT));
00152     sprite.setPosition(
00153         sf::Vector2f(getFurthestCarPos(cars).x * Config::PPM, 0.5 * Config::WINDOW_HEIGHT) -
00154         sf::Vector2f(Config::WINDOW_WIDTH / 2.0f, Config::WINDOW_HEIGHT / 2.0f));
00155     return sprite;
00156 }
```

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>

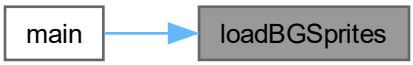
Definition at line 158 of file Utility.cc.

```
00159                                                                   {
00160     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

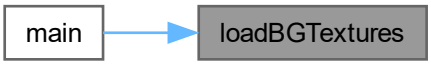
std::vector<sf::Texture*>

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.

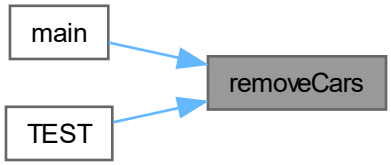
Parameters

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.

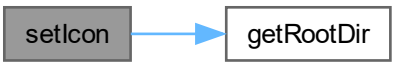
Parameters

window	

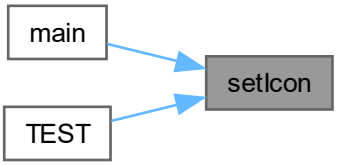
Definition at line 126 of file Utility.cc.

```
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.

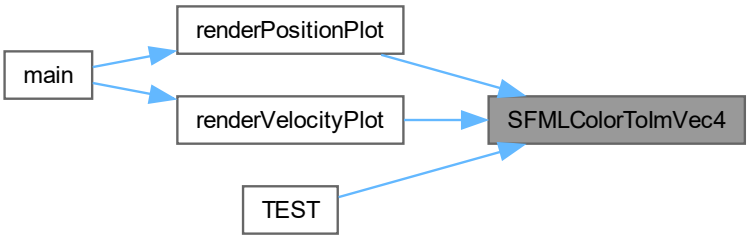
Returns

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.

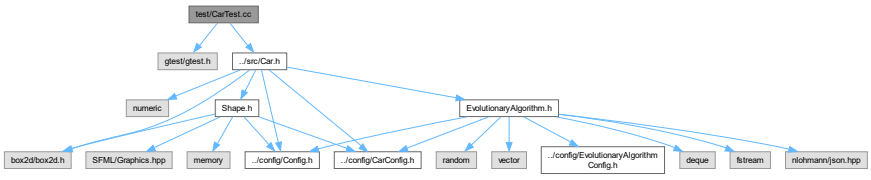
```
00001
00009 #ifndef UTILITY_H
00010 #define UTILITY_H
00011
00012 #include <filesystem>
00013 #include <cmath>
00014 #include <random>
00015 #include <vector>
00016
00017 #include "box2d/box2d.h"
00018 #include "imgui.h"
00019 #include "SFML/Graphics.hpp"
00020 #include "EvolutionaryAlgorithm.h"
00021
00022 #include "../config/CarConfig.h"
00023 #include "../config/Config.h"
00024 #include "../config/MapGenConfig.h"
00025 #include "Car.h"
00026
00027 typedef std::shared_ptr<b2World> b2WorldPtr;
00028
00033 void applyAirResistance(Car car);
00034
00041 void generateGround(const b2WorldPtr& world, std::vector<Polygon>* groundVector,
00042                     const std::vector<Car>& cars);
00043
00048 float getNextGroundPartDegree();
00049
00050 Car generateCar(const b2WorldPtr& world, const Chromosome& chromosome);
00051
00058 ImVec4 SFMLColorToImVec4(sf::Color color);
00059
00067 b2Vec2 getFurthestCarPos(const std::vector<Car>& cars);
00068
00077 int getIdxOfGrndClstToLoc(std::vector<Polygon> ground, float x);
00078
00085 void removeCars(const b2WorldPtr& world, std::vector<Car>* cars);
00086
00092 std::filesystem::path getRootDir();
00093
00099 void setIcon(sf::RenderWindow& window);
00100
00106 std::vector<sf::Texture*> loadBGTextures();
00107
00115 sf::Sprite loadBGSprite(sf::Texture* texture, const std::vector<Car>& cars);
00116
00124 std::vector<sf::Sprite> loadBGSprites(std::vector<sf::Texture*> textures,
00125                                       const std::vector<Car>& cars);
00126
00127 #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

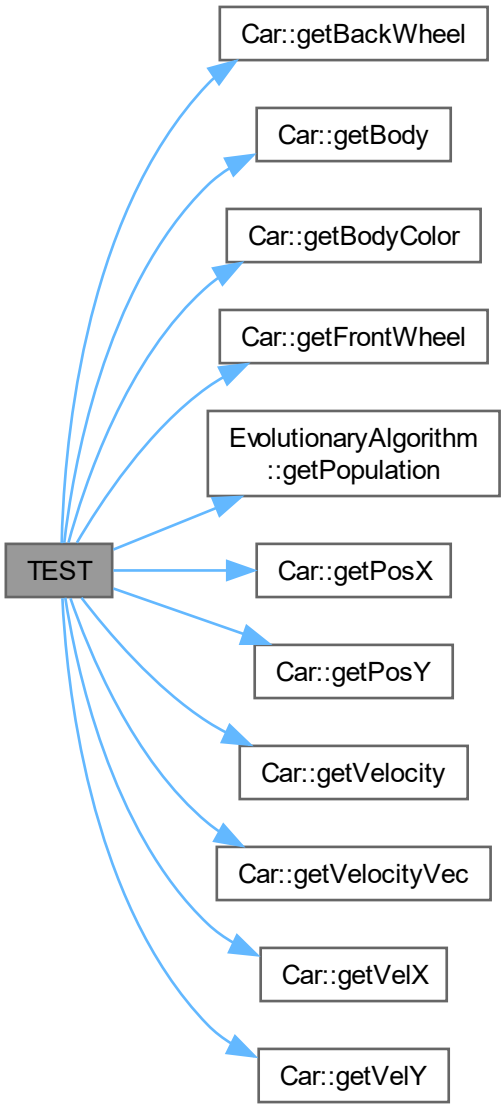
4.39.2.1 TEST()

```
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);
```

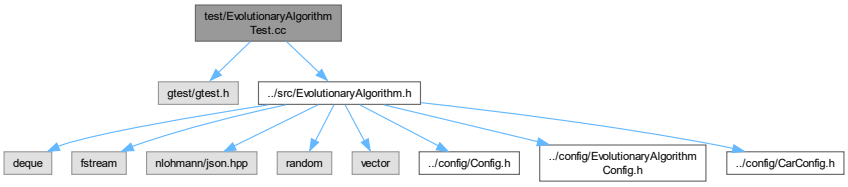
```
00020     Chromosome chromosome = ea.getPopulation()[0];
00021     Car car = Car(world, x, y, chromosome, bodyColor, wheelColor);
00022
00023     EXPECT_EQ(car.getPosX(), x);
00024     EXPECT_EQ(car.getPosY(), y);
00025     EXPECT_EQ(car.getBodyColor(), bodyColor);
00026     EXPECT_NO_THROW(car.getBody());
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 }
```

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:



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

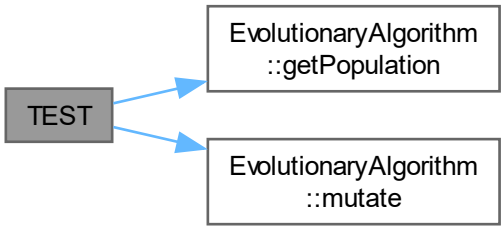
4.41.2.1 TEST() [1/4]

```
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:



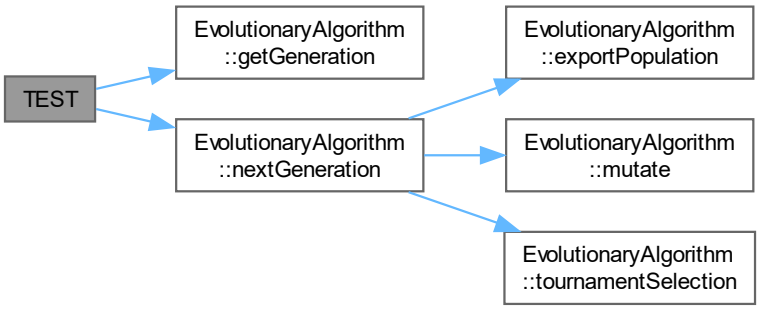
4.41.2.2 TEST() [2/4]

```
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:



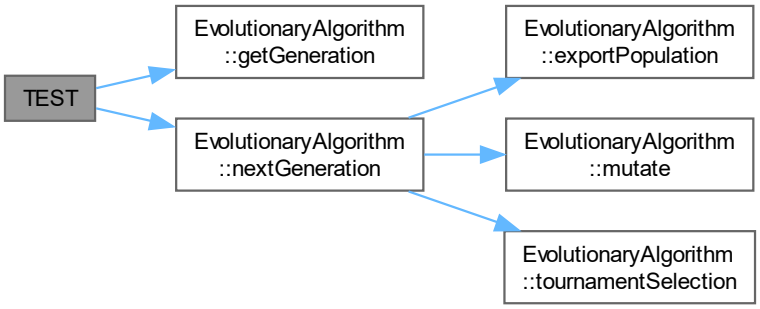
4.41.2.3 TEST() [3/4]

```
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:



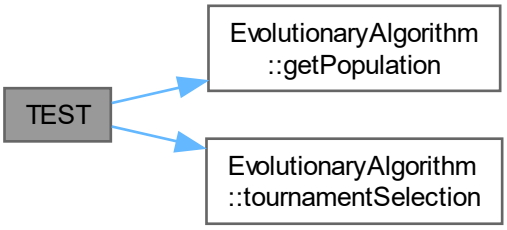
4.41.2.4 TEST() [4/4]

```
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:



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) {
```

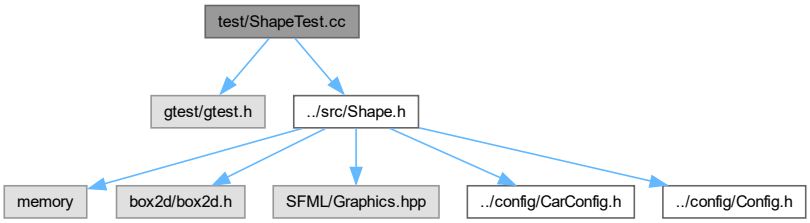
```
00041         evo.nextGeneration();
00042     }
00043
00044     ASSERT_EQ(evo.getGeneration(), 50);
00045 }
00046
00047 TEST(EvolutionaryAlgorithmTest, SaveToJsonTest) {
00048     EvolutionaryAlgorithm evo(2, true);
00049     evo.nextGeneration();
00050     for (int i = 0; i < 5; ++i) {
00051         evo.nextGeneration();
00052     }
00053
00054     ASSERT_EQ(evo.getGeneration(), 6);
00055 }
```

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

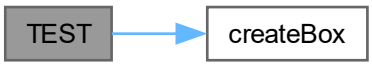
4.43.2.1 TEST() [1/16]

```
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:



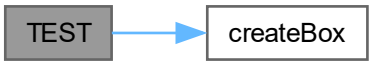
4.43.2.2 TEST() [2/16]

```
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:



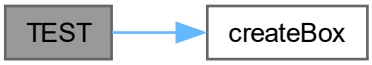
4.43.2.3 TEST() [3/16]

```
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:



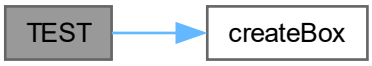
4.43.2.4 TEST() [4/16]

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

Definition at line 38 of file ShapeTest.cc.

```
00038                                       {
00039     // Test with invalid input (negative width and height)
00040     b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
00041     float x = 0.0f, y = 0.0f, width = 10.0f, height = 0.0f, density = 1.0f, friction = 0.5f;
00042     sf::Color color = sf::Color::Red;
00043
00044     ASSERT_THROW(createBox(world, x, y, width, height, density, friction, color),
00045                  std::invalid_argument);
00046 }
```

Here is the call graph for this function:



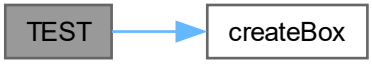
4.43.2.5 TEST() [5/16]

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

Definition at line 28 of file ShapeTest.cc.

```
00028                                       {
00029     // Test with invalid input (negative width and height)
00030     b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
00031     float x = 0.0f, y = 0.0f, width = -10.0f, height = 10.0f, density = 1.0f, friction = 0.5f;
00032     sf::Color color = sf::Color::Red;
00033
00034     ASSERT_THROW(createBox(world, x, y, width, height, density, friction, color),
00035                  std::invalid_argument);
00036 }
```

Here is the call graph for this function:



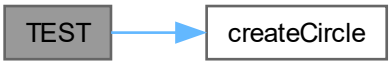
4.43.2.6 TEST() [6/16]

```
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:



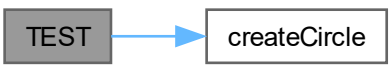
4.43.2.7 TEST() [7/16]

```
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:



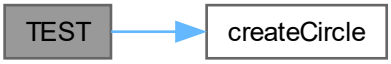
4.43.2.8 TEST() [8/16]

```
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:



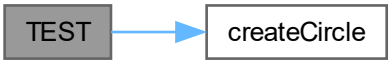
4.43.2.9 TEST() [9/16]

```
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:



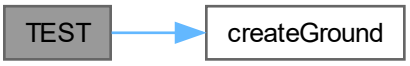
4.43.2.10 TEST() [10/16]

```
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:



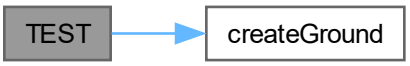
4.43.2.11 TEST() [11/16]

```
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:



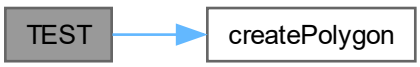
4.43.2.12 TEST() [12/16]

```
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:



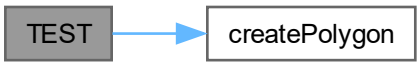
4.43.2.13 TEST() [13/16]

```
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:



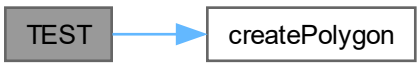
4.43.2.14 TEST() [14/16]

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

Definition at line 178 of file ShapeTest.cc.

```
00178                                          {
00179     b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
00180     float x = 0.0f, y = 0.0f, density = 0.0f, friction = 0.5f;
00181     std::vector<b2Vec2> vertices = {b2Vec2(-1.0f, -1.0f), b2Vec2(1.0f, -1.0f), b2Vec2(0.0f, 1.0f),
00182                                     b2Vec2(-1.0f, -1.0f), b2Vec2(1.0f, -1.0f), b2Vec2(0.0f, 1.0f),
00183                                     b2Vec2(-1.0f, -1.0f), b2Vec2(1.0f, -1.0f)};
00184     sf::Color color = sf::Color::Red;
00185     ASSERT_THROW(createPolygon(world, x, y, vertices, density, friction, color);
00186                  , std::invalid_argument);
00187 }
```

Here is the call graph for this function:



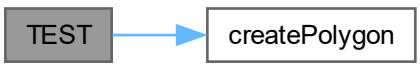
4.43.2.15 TEST() [15/16]

```
TEST (
              CreatePolygonTest ,
              InvalidFrictionTest  )
```

Definition at line 189 of file ShapeTest.cc.

```
00189                                          {
00190     b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
00191     float x = 0.0f, y = 0.0f, density = 1.0f, friction = -0.5f;
00192     std::vector<b2Vec2> vertices = {b2Vec2(-1.0f, -1.0f), b2Vec2(1.0f, -1.0f), b2Vec2(0.0f, 1.0f),
00193                                     b2Vec2(-1.0f, -1.0f), b2Vec2(1.0f, -1.0f), b2Vec2(0.0f, 1.0f),
00194                                     b2Vec2(-1.0f, -1.0f), b2Vec2(1.0f, -1.0f)};
00195     sf::Color color = sf::Color::Red;
00196     ASSERT_THROW(createPolygon(world, x, y, vertices, density, friction, color);
00197                  , std::invalid_argument);
00198 }
```

Here is the call graph for this function:



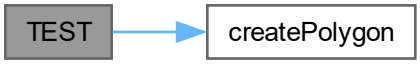
4.43.2.16 TEST() [16/16]

```
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.

```
00001
00009 #include <gtest/gtest.h>
00010 #include "../src/Shape.h"
00011
00012 TEST(CreateBoxTest, BasicTest) {
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 }
00027
00028 TEST(CreateBoxTest, InvalidWidthTest) {
00029     // Test with invalid input (negative width and height)
00030     b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
00031     float x = 0.0f, y = 0.0f, width = -10.0f, height = 10.0f, density = 1.0f, friction = 0.5f;
00032     sf::Color color = sf::Color::Red;
00033
00034     ASSERT_THROW(createBox(world, x, y, width, height, density, friction, color),
00035                  std::invalid_argument);
00036 }
00037
00038 TEST(CreateBoxTest, InvalidHeightTest) {
00039     // Test with invalid input (negative width and height)
00040     b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
00041     float x = 0.0f, y = 0.0f, width = 10.0f, height = 0.0f, density = 1.0f, friction = 0.5f;
00042     sf::Color color = sf::Color::Red;
00043
00044     ASSERT_THROW(createBox(world, x, y, width, height, density, friction, color),
00045                  std::invalid_argument);
```

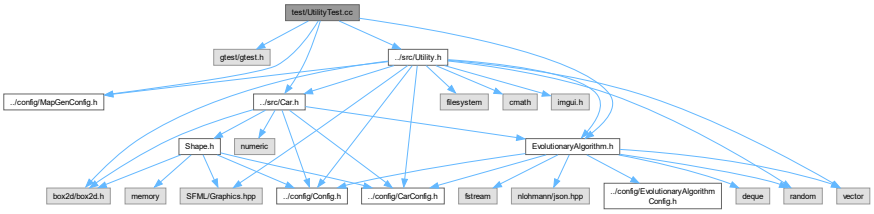
```
00046 }
00047
00048 TEST(CreateBoxTest, InvalidDensityTest) {
00049     // Test with invalid input (negative width and height)
00050     b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
00051     float x = 1.0f, y = 1.0f, width = 10.0f, height = 1.0f, 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 }
00057
00058 TEST(CreateBoxTest, InvalidFrictionTest) {
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 }
00067
00068 TEST(CreateGroundTest, BasicTest) {
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 }
00083
00084 TEST(CreateGroundTest, InvalidVerticesTest) {
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 }
00093
00094 TEST(CreateCircleTest, BasicTest) {
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 }
00108
00109 TEST(CreateCircleTest, InvalidRadiusTest) {
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 }
00117
00118 TEST(CreateCircleTest, InvalidDensityTest) {
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 }
00126
00127 TEST(CreateCircleTest, InvalidFrictionTest) {
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 }
00135
00136 TEST(CreatePolygonTest, BasicTest) {
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 }
00157
00158 TEST(CreatePolygonTest, EmptyVerticesTest) {
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 }
00166
00167 TEST(CreatePolygonTest, TooMuchVerticesTest) {
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 }
00177
00178 TEST(CreatePolygonTest, InvalidDensityTest) {
00179     b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
00180     float x = 0.0f, y = 0.0f, density = 0.0f, friction = 0.5f;
00181     std::vector<b2Vec2> vertices = {b2Vec2(-1.0f, -1.0f), b2Vec2(1.0f, -1.0f), b2Vec2(0.0f, 1.0f),
00182                                     b2Vec2(-1.0f, -1.0f), b2Vec2(1.0f, -1.0f), b2Vec2(0.0f, 1.0f),
00183                                     b2Vec2(-1.0f, -1.0f), b2Vec2(1.0f, -1.0f)};
00184     sf::Color color = sf::Color::Red;
00185     ASSERT_THROW(createPolygon(world, x, y, vertices, density, friction, color);
00186                  , std::invalid_argument);
00187 }
00188
00189 TEST(CreatePolygonTest, InvalidFrictionTest) {
00190     b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
00191     float x = 0.0f, y = 0.0f, density = 1.0f, friction = -0.5f;
00192     std::vector<b2Vec2> vertices = {b2Vec2(-1.0f, -1.0f), b2Vec2(1.0f, -1.0f), b2Vec2(0.0f, 1.0f),
00193                                     b2Vec2(-1.0f, -1.0f), b2Vec2(1.0f, -1.0f), b2Vec2(0.0f, 1.0f),
00194                                     b2Vec2(-1.0f, -1.0f), b2Vec2(1.0f, -1.0f)};
00195     sf::Color color = sf::Color::Red;
00196     ASSERT_THROW(createPolygon(world, x, y, vertices, density, friction, color);
00197                  , std::invalid_argument);
00198 }
```

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

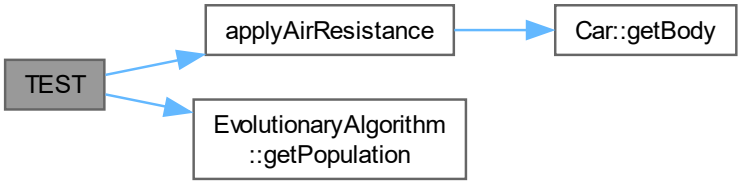
4.45.1.1 TEST() [1/10]

```
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:



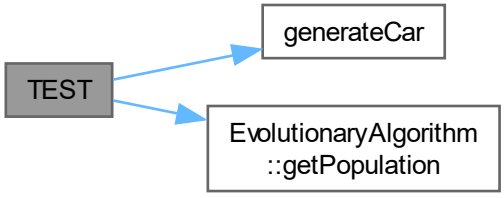
4.45.1.2 TEST() [2/10]

```
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]

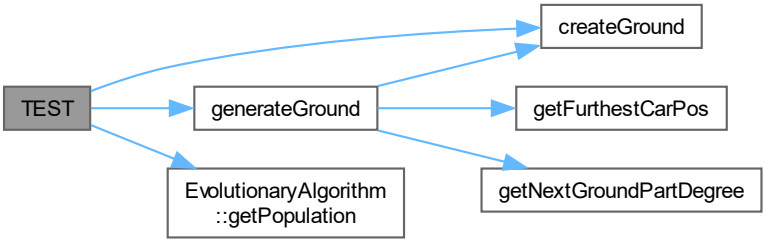
```
TEST (
            UtilityTest ,
            generateGroundTest  )
```

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:



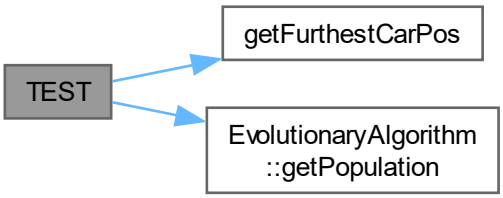
4.45.1.4 TEST() [4/10]

```
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:



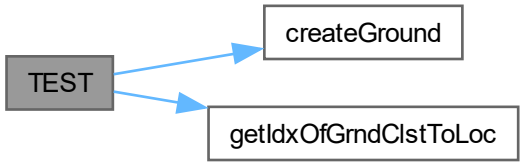
4.45.1.5 TEST() [5/10]

```
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:

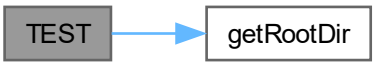
4.45.1.7 TEST() [7/10]

```
TEST (
            UtilityTest ,
            getRootDirTest  )
```

Definition at line 137 of file UtilityTest.cc.

```
00137 { EXPECT_NO_THROW(getRootDir()); }
```

Here is the call graph for this function:



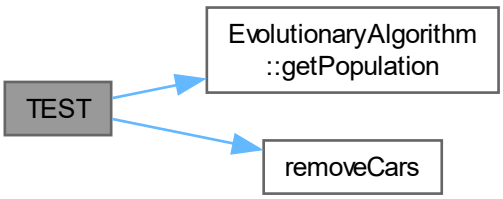
4.45.1.8 TEST() [8/10]

```
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.

```
00001
00009 #include <gtest/gtest.h>
00010
00011 #include "../config/MapGenConfig.h"
00012 #include "../src/Car.h"
00013 #include "../src/EvolutionaryAlgorithm.h"
00014 #include "../src/Utility.h"
00015
00016 TEST(UtilityTest, applyAirResistanceTest) {
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 }
00028
00029 TEST(UtilityTest, generateGroundTest) {
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 }
00057
00058 TEST(UtilityTest, getNextGroundPartDegreeTest) { EXPECT_NO_THROW(getNextGroundPartDegree()); }
00059
00060 TEST(UtilityTest, generateCarTest) {
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 }
00071
00072 TEST(UtilityTest, getFurthestCarPosTest) {
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 }
00089
```

```
00090 TEST(UtilityTest, SFMLColorToImVec4) {
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 }
00099 
00100 TEST(UtilityTest, getIdxOfGrndClstToLocTest) {
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 }
00119 
00120 TEST(UtilityTest, removeCarsTest) {
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 }
00136 
00137 TEST(UtilityTest, getRootDirTest) { EXPECT_NO_THROW(getRootDir()); }
00138 
00139 TEST(UtilityTest, setIconTest) {
00140     sf::RenderWindow window(sf::VideoMode(800, 600), "SFML window");
00141     EXPECT_NO_THROW(setIcon(window));
00142 }
```

Index