ZPR 23L Projekt - EvoRacer

Generated by Doxygen 1.9.7

| 1 Class Index | 1 |
|--|--------|
| 1.1 Class List | 1 |
| 2 File Index | 3 |
| 2.1 File List | 3 |
| 3 Class Documentation | 5 |
| 3.1 Box Struct Reference | 5 |
| 3.1.1 Detailed Description | 5 |
| 3.1.2 Member Data Documentation | 6 |
| 3.1.2.1 body | 6 |
| 3.1.2.2 color | 6 |
| 3.1.2.3 height | 6 |
| 3.1.2.4 width | 6 |
| 3.2 Car Class Reference | 6 |
| 3.2.1 Detailed Description | 8 |
| 3.2.2 Constructor & Destructor Documentation | 8 |
| 3.2.2.1 Car() | 8 |
| 3.2.3 Member Function Documentation | 9 |
| 3.2.3.1 getBackWheel() | 9 |
| 3.2.3.2 getBody() | 10 |
| 3.2.3.3 getBodyColor() | 10 |
| 3.2.3.4 getFrontWheel() | 10 |
| 3.2.3.5 getPosX() | 11 |
| 3.2.3.6 getPosXVec() | 11 |
| 3.2.3.7 getPosY() | 11 |
| 3.2.3.8 getPosYVec() | 12 |
| 3.2.3.9 getVelocity() | 12 |
| 3.2.3.10 getVelocityVec() | 12 |
| 3.2.3.11 getVelX() | 13 |
| 3.2.3.12 getVelY() | 13 |
| 3.2.3.13 setCollisionFilter() | 13 |
| 3.3 CarConfig Class Reference | 14 |
| 3.3.1 Detailed Description | 15 |
| 3.3.2 Member Data Documentation | 15 |
| 3.3.2.1 CAR_TORQUE | 15 |
| 3.3.2.2 CAR_VERTICES | 15 |
| 3.3.2.3 MAX_JOINT_LENGTH | 15 |
| 3.4 Chromosome Struct Reference | 15 |
| 3.4.1 Detailed Description | 16 |
| 3.4.2 Member Data Documentation | 17 |
| 3.4.2.1 bodyDensity | 17 |
| 3.4.2.2 bodyLengths | 17 |
| · · · | |

| 3.4.2.3 fitness | 17 |
|--|----|
| 3.4.2.4 wheelDensity | 17 |
| 3.4.2.5 wheelRadius | 17 |
| 3.5 Circle Struct Reference | 18 |
| 3.5.1 Detailed Description | 18 |
| 3.5.2 Member Data Documentation | 18 |
| 3.5.2.1 body | 18 |
| 3.5.2.2 color | 18 |
| 3.5.2.3 radius | 19 |
| 3.6 Config Class Reference | 19 |
| 3.6.1 Detailed Description | 20 |
| 3.6.2 Member Data Documentation | 20 |
| 3.6.2.1 BACK_WHEEL_POS | 20 |
| 3.6.2.2 CATEGORY_BITS | 20 |
| 3.6.2.3 DEBUG | 20 |
| 3.6.2.4 DEG_PER_RAD | 20 |
| 3.6.2.5 FRICTION | 20 |
| 3.6.2.6 FRONT_WHEEL_POS | 20 |
| 3.6.2.7 GENERATION_TIME | 21 |
| 3.6.2.8 GRAVITATIONAL_ACCELERATION | 21 |
| 3.6.2.9 GROUND_PARTS_RENDERED | 21 |
| 3.6.2.10 MASK_BITS | 21 |
| 3.6.2.11 MAX_FPS | 21 |
| 3.6.2.12 Pl | 21 |
| 3.6.2.13 PPM | 21 |
| 3.6.2.14 SAVE_FILE_NAME | 22 |
| 3.6.2.15 SAVE_TO_FILE | 22 |
| 3.6.2.16 VELOCITY_ARRAY_SIZE | 22 |
| 3.6.2.17 WINDOW_HEIGHT | 22 |
| 3.6.2.18 WINDOW_WIDTH | 22 |
| 3.7 EvolutionaryAlgorithm Class Reference | 23 |
| 3.7.1 Detailed Description | 23 |
| 3.7.2 Constructor & Destructor Documentation | 24 |
| 3.7.2.1 EvolutionaryAlgorithm() | 24 |
| 3.7.3 Member Function Documentation | 24 |
| 3.7.3.1 exportPopulation() | 24 |
| 3.7.3.2 getGeneration() | 25 |
| 3.7.3.3 getPopulation() | 25 |
| 3.7.3.4 getPopulationSize() | 26 |
| 3.7.3.5 mutate() | 26 |
| 3.7.3.6 nextGeneration() | 28 |
| 3.7.3.7 setFitness() | 29 |

| 3.7.3.8 tournamentSelection() | 29 |
|---|----|
| 3.8 EvolutionaryAlgorithmConfig Class Reference | 30 |
| 3.8.1 Detailed Description | 31 |
| 3.8.2 Member Data Documentation | 31 |
| 3.8.2.1 INITIAL_BODY_DENSITY_MEAN | 31 |
| 3.8.2.2 INITIAL_BODY_DENSITY_VARIANCE | 31 |
| 3.8.2.3 INITIAL_BODY_LENGTH_MEAN | 31 |
| 3.8.2.4 INITIAL_BODY_LENGTH_VARIANCE | 32 |
| 3.8.2.5 INITIAL_WHEEL_DENSITY_MEAN | 32 |
| 3.8.2.6 INITIAL_WHEEL_DENSITY_VARIANCE | 32 |
| 3.8.2.7 INITIAL_WHEEL_RADIUS_MEAN | 32 |
| 3.8.2.8 INITIAL_WHEEL_RADIUS_VARIANCE | 32 |
| 3.8.2.9 MAX_BODY_DENSITY | 32 |
| 3.8.2.10 MAX_BODY_LENGTH | 32 |
| 3.8.2.11 MAX_WHEEL_DENSITY | 33 |
| 3.8.2.12 MAX_WHEEL_RADIUS | 33 |
| 3.8.2.13 MIN_BODY_DENSITY | 33 |
| 3.8.2.14 MIN_BODY_LENGTH | 33 |
| 3.8.2.15 MIN_WHEEL_DENSITY | 33 |
| 3.8.2.16 MIN_WHEEL_RADIUS | 33 |
| 3.8.2.17 MUTATION_FACTOR_BODY_DENSITY | 33 |
| 3.8.2.18 MUTATION_FACTOR_BODY_LENGTHS | 34 |
| 3.8.2.19 MUTATION_FACTOR_WHEEL_DENSITY | 34 |
| 3.8.2.20 MUTATION_FACTOR_WHEEL_RADIUS | 34 |
| 3.8.2.21 MUTATION_RATE_BODY_DENSITY | 34 |
| 3.8.2.22 MUTATION_RATE_BODY_LENGTHS | 34 |
| 3.8.2.23 MUTATION_RATE_WHEEL_DENSITY | 34 |
| 3.8.2.24 MUTATION_RATE_WHEEL_RADIUS | 34 |
| 3.8.2.25 POPULATION_SIZE | 35 |
| 3.8.2.26 TOURNAMENT_SIZE | 35 |
| 3.9 MapGenConfig Class Reference | 35 |
| 3.9.1 Detailed Description | 36 |
| 3.9.2 Member Data Documentation | 36 |
| 3.9.2.1 BG_SPRITES_COUNT | 36 |
| 3.9.2.2 CAR_STARTING_X | 36 |
| 3.9.2.3 CAR_STARTING_Y | 36 |
| 3.9.2.4 GENERATE_DISTANCE | 36 |
| 3.9.2.5 GROUND_DEGREE_DEVIATION | 36 |
| 3.9.2.6 GROUND_LEG_LENGTH | 37 |
| 3.9.2.7 GROUND_PART_LENGTH | 37 |
| 3.9.2.8 GROUND_STARTING_X | 37 |
| 3.9.2.9 GROUND_STARTING_Y | 37 |

| | 3.9.2.10 MAX_GROUND_DEGREE | 37 |
|---|---|----|
| | 3.10 Polygon Struct Reference | 38 |
| | 3.10.1 Detailed Description | 38 |
| | 3.10.2 Member Data Documentation | 39 |
| | 3.10.2.1 body | 39 |
| | 3.10.2.2 color | 39 |
| | 3.10.2.3 vertices | 39 |
| 4 | File Documentation | 41 |
| | 4.1 config/CarConfig.h File Reference | 41 |
| | 4.1.1 Detailed Description | 41 |
| | 4.2 CarConfig.h | 42 |
| | 4.3 config/Config.h File Reference | 42 |
| | 4.3.1 Detailed Description | 42 |
| | 4.4 Config.h | 43 |
| | 4.5 config/EvolutionaryAlgorithmConfig.h File Reference | 43 |
| | 4.5.1 Detailed Description | 44 |
| | 4.6 EvolutionaryAlgorithmConfig.h | 44 |
| | 4.7 config/MapGenConfig.h File Reference | 45 |
| | 4.7.1 Detailed Description | 45 |
| | 4.8 MapGenConfig.h | 46 |
| | 4.9 src/Car.cc File Reference | 46 |
| | 4.9.1 Detailed Description | 46 |
| | 4.9.2 Function Documentation | 47 |
| | 4.9.2.1 createVertices() | 47 |
| | 4.10 Car.cc | 47 |
| | 4.11 src/Car.h File Reference | 48 |
| | 4.11.1 Detailed Description | 49 |
| | 4.11.2 Typedef Documentation | 50 |
| | 4.11.2.1 b2WorldPtr | 50 |
| | 4.11.3 Function Documentation | 50 |
| | 4.11.3.1 createVertices() | 50 |
| | 4.12 Car.h | 50 |
| | 4.13 src/EvolutionaryAlgorithm.cc File Reference | 51 |
| | 4.13.1 Detailed Description | 51 |
| | 4.14 EvolutionaryAlgorithm.cc | 52 |
| | 4.15 src/EvolutionaryAlgorithm.h File Reference | 54 |
| | 4.15.1 Detailed Description | 55 |
| | 4.16 EvolutionaryAlgorithm.h | 55 |
| | 4.17 src/GUI.cc File Reference | 56 |
| | 4.17.1 Detailed Description | 56 |
| | 4.17.2 Function Documentation | 57 |

| 4.17.2.1 renderPositionPlot() . | | | 57 |
|-----------------------------------|------|------|----|
| 4.17.2.2 renderVelocityPlot() . | | | 58 |
| 4.18 GUI.cc | | | 59 |
| 4.19 src/GUI.h File Reference | | | 60 |
| 4.19.1 Detailed Description | | | 61 |
| 4.19.2 Function Documentation | | | 61 |
| 4.19.2.1 renderPositionPlot() . | | | 61 |
| 4.19.2.2 renderVelocityPlot() . | | | 62 |
| 4.20 GUI.h | | | 63 |
| 4.21 src/Main.cc File Reference | | | 63 |
| 4.21.1 Detailed Description | | | 64 |
| 4.21.2 Typedef Documentation | | | 65 |
| 4.21.2.1 b2WorldPtr | | | 65 |
| 4.21.3 Function Documentation | | | 65 |
| 4.21.3.1 main() | | | 65 |
| 4.21.4 Variable Documentation | | | 67 |
| 4.21.4.1 world | | | 67 |
| 4.22 Main.cc | | | 68 |
| 4.23 src/Render.cc File Reference | | | 69 |
| 4.23.1 Detailed Description | | | 70 |
| 4.23.2 Function Documentation | | | 70 |
| 4.23.2.1 render() | | | 70 |
| 4.23.2.2 renderCar() | | | 72 |
| 4.23.2.3 renderCircle() | | | 73 |
| 4.23.2.4 renderCircleDebug() . | | | 73 |
| 4.23.2.5 renderPolygon() | | | 74 |
| 4.23.2.6 renderPolygonDebug(| | | 75 |
| 4.24 Render.cc | | | 75 |
| 4.25 src/Render.h File Reference | | | 77 |
| 4.25.1 Detailed Description | | | 78 |
| 4.25.2 Function Documentation | | | 78 |
| 4.25.2.1 render() | | | 78 |
| 4.25.2.2 renderCar() | | | 79 |
| 4.25.2.3 renderCircle() | | | 80 |
| 4.25.2.4 renderCircleDebug() . | | | 81 |
| 4.25.2.5 renderPolygon() | | | 81 |
| 4.25.2.6 renderPolygonDebug(| | | 82 |
| 4.26 Render.h | | | 83 |
| 4.27 src/Shape.cc File Reference | | | 83 |
| 4.27.1 Detailed Description | | | 84 |
| 4.27.2 Function Documentation | | | 84 |
| 4.27.2.1 createBox() | | | 84 |

| 4.27.2.2 createCircle() | 85 |
|--------------------------------------|-------|
| 4.27.2.3 createGround() | . 86 |
| 4.27.2.4 createPolygon() | 87 |
| 4.28 Shape.cc | . 88 |
| 4.29 src/Shape.h File Reference | 90 |
| 4.29.1 Detailed Description | 91 |
| 4.29.2 Typedef Documentation | 91 |
| 4.29.2.1 b2WorldPtr | 91 |
| 4.29.3 Function Documentation | 91 |
| 4.29.3.1 createBox() | 91 |
| 4.29.3.2 createCircle() | 92 |
| 4.29.3.3 createGround() | 93 |
| 4.29.3.4 createPolygon() | 94 |
| 4.30 Shape.h | 95 |
| 4.31 src/UserInput.cc File Reference | 96 |
| 4.31.1 Detailed Description | 97 |
| 4.31.2 Function Documentation | 97 |
| 4.31.2.1 handleEvents() | 97 |
| 4.31.2.2 handleUserInput() | 98 |
| 4.32 UserInput.cc | . 98 |
| 4.33 src/UserInput.h File Reference | 99 |
| 4.33.1 Detailed Description | . 100 |
| 4.33.2 Function Documentation | . 100 |
| 4.33.2.1 handleEvents() | . 100 |
| 4.33.2.2 handleUserInput() | . 101 |
| 4.34 UserInput.h | . 102 |
| 4.35 src/Utility.cc File Reference | . 102 |
| 4.35.1 Detailed Description | . 103 |
| 4.35.2 Function Documentation | . 103 |
| 4.35.2.1 applyAirResistance() | . 103 |
| 4.35.2.2 generateCar() | . 104 |
| 4.35.2.3 generateGround() | . 105 |
| 4.35.2.4 getFurthestCarPos() | . 106 |
| 4.35.2.5 getIdxOfGrndClstToLoc() | . 107 |
| 4.35.2.6 getNextGroundPartDegree() | . 108 |
| 4.35.2.7 getRootDir() | . 108 |
| 4.35.2.8 loadBGSprite() | . 109 |
| 4.35.2.9 loadBGSprites() | . 110 |
| 4.35.2.10 loadBGTextures() | . 111 |
| 4.35.2.11 removeCars() | . 111 |
| 4.35.2.12 setlcon() | . 112 |
| 4.35.2.13 SFMLColorToImVec4() | 113 |

| 4.36 Utility.cc |
|---|
| 4.37 src/Utility.h File Reference |
| 4.37.1 Detailed Description |
| 4.37.2 Typedef Documentation |
| 4.37.2.1 b2WorldPtr |
| 4.37.3 Function Documentation |
| 4.37.3.1 applyAirResistance() |
| 4.37.3.2 generateCar() |
| 4.37.3.3 generateGround() |
| 4.37.3.4 getFurthestCarPos() |
| 4.37.3.5 getIdxOfGrndClstToLoc() |
| 4.37.3.6 getNextGroundPartDegree() |
| 4.37.3.7 getRootDir() |
| 4.37.3.8 loadBGSprite() |
| 4.37.3.9 loadBGSprites() |
| 4.37.3.10 loadBGTextures() |
| 4.37.3.11 removeCars() |
| 4.37.3.12 setlcon() |
| 4.37.3.13 SFMLColorToImVec4() |
| 4.38 Utility.h |
| 4.39 test/CarTest.cc File Reference |
| 4.39.1 Detailed Description |
| 4.39.2 Function Documentation |
| 4.39.2.1 TEST() |
| 4.40 CarTest.cc |
| 4.41 test/EvolutionaryAlgorithmTest.cc File Reference |
| 4.41.1 Detailed Description |
| 4.41.2 Function Documentation |
| 4.41.2.1 TEST() [1/4] |
| 4.41.2.2 TEST() [2/4] |
| 4.41.2.3 TEST() [3/4] |
| 4.41.2.4 TEST() [4/4] |
| 4.42 EvolutionaryAlgorithmTest.cc |
| 4.43 test/ShapeTest.cc File Reference |
| 4.43.1 Detailed Description |
| 4.43.2 Function Documentation |
| 4.43.2.1 TEST() [1/16] |
| 4.43.2.2 TEST() [2/16] |
| 4.43.2.3 TEST() [3/16] |
| 4.43.2.4 TEST() [4/16] |
| 4.43.2.5 TEST() [5/16] |
| 4.43.2.6 TEST() [6/16] |

Index

| 4.43.2.7 TEST() [7/16] | 139 |
|---|-----|
| 4.43.2.8 TEST() [8/16] | 140 |
| 4.43.2.9 TEST() [9/16] | 140 |
| 4.43.2.10 TEST() [10/16] | 141 |
| 4.43.2.11 TEST() [11/16] | 141 |
| 4.43.2.12 TEST() [12/16] | 142 |
| 4.43.2.13 TEST() [13/16] | 142 |
| 4.43.2.14 TEST() [14/16] | 143 |
| 4.43.2.15 TEST() [15/16] | 143 |
| 4.43.2.16 TEST() [16/16] | 144 |
| 4.44 ShapeTest.cc | 144 |
| 4.45 test/UtilityTest.cc File Reference | 146 |
| 4.45.1 Function Documentation | 147 |
| 4.45.1.1 TEST() [1/10] | 147 |
| 4.45.1.2 TEST() [2/10] | 148 |
| 4.45.1.3 TEST() [3/10] | 148 |
| 4.45.1.4 TEST() [4/10] | 149 |
| 4.45.1.5 TEST() [5/10] | 150 |
| 4.45.1.6 TEST() [6/10] | 150 |
| 4.45.1.7 TEST() [7/10] | 151 |
| 4.45.1.8 TEST() [8/10] | 151 |
| 4.45.1.9 TEST() [9/10] | 152 |
| 4.45.1.10 TEST() [10/10] | 152 |
| 4.46 UtilityTest.cc | |
| | |

155

Chapter 1

Class Index

1.1 Class List

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

| Config | 1 |
|--------------------------|---|
| omosome | 1 |
| le | 1 |
| fig | 1 |
| lutionaryAlgorithm | 2 |
| lutionaryAlgorithmConfig | 3 |
| GenConfig | 3 |
| gon | 3 |

2 Class Index

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

| config/CarConfig.h | |
|--|-----|
| This file contains all the constant values for the car class | 41 |
| config/Config.h | |
| This file contains all the constant values used in the program | 42 |
| config/EvolutionaryAlgorithmConfig.h | |
| This file contains all the constant values used in the evolutionary algorithm | 43 |
| config/MapGenConfig.h | |
| This file contains all the constant values used in the map generation algorithm | 45 |
| src/Car.cc | |
| Creates a car with a polygon (car's body) and two circles (front and back wheels) | 46 |
| src/Car.h | |
| Header file for the Car class | 48 |
| src/EvolutionaryAlgorithm.cc | |
| Implementation file for EvolutionaryAlgorithm class, Algorithm used for evolving the cars | 51 |
| src/EvolutionaryAlgorithm.h | E 4 |
| Header file for EvolutionaryAlgorithm class | 54 |
| File containing GUI functions | 56 |
| src/GUI.h | 50 |
| Header for a file containing GUI functions | 60 |
| src/Main.cc | 00 |
| Main file for the project, contains the main loop | 63 |
| src/Render.cc | - |
| This file contains the render function, which is responsible for rendering all the shapes in the | |
| world | 69 |
| src/Render.h | |
| Header file for render function | 77 |
| src/Shape.cc | |
| This file contains functions for creating Box2D objects | 83 |
| src/Shape.h | |
| Header file for functions for creating Box2D objects | 90 |
| src/UserInput.cc | |
| File containing user input functions | 96 |
| src/UserInput.h | |
| Header for a file containing user input functions | 99 |

File Index

| src/Utility.cc | |
|---|----|
| File containing utility functions |)2 |
| src/Utility.h | |
| Header for a file containing utility functions | Ę |
| test/CarTest.cc | |
| This file contains tests for functions from src/Car.h | 28 |
| test/EvolutionaryAlgorithmTest.cc | |
| This file contains tests for functions from src/EvolutionaryAlgorithm.h | 31 |
| test/ShapeTest.cc | |
| This file contains tests for functions from src/Shape.h | 35 |
| test/UtilityTest.cc | 16 |

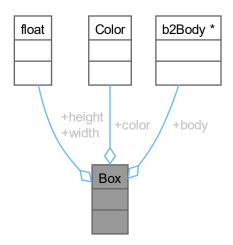
Chapter 3

Class Documentation

3.1 Box Struct Reference

#include <Shape.h>

Collaboration diagram for Box:



Public Attributes

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

3.1.1 Detailed Description

Definition at line 20 of file Shape.h.

3.1.2 Member Data Documentation

3.1.2.1 body

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

Definition at line 24 of file Shape.h.

3.1.2.2 color

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

Definition at line 23 of file Shape.h.

3.1.2.3 height

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

Definition at line 22 of file Shape.h.

3.1.2.4 width

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

Definition at line 21 of file Shape.h.

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

• src/Shape.h

3.2 Car Class Reference

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

3.2 Car Class Reference 7

Collaboration diagram for Car:

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

Public Member Functions

- Car (const b2WorldPtr &world, float x, float y, const Chromosome &chromosome, sf::Color bodyColor, sf::← Color wheelColor)
- Polygon * getBody ()
- Circle * getFrontWheel ()
- Circle * getBackWheel ()
- float getPosX () const
- float getPosY () const
- std::vector< float > * getVelX ()
- std::vector< float > * getVelY ()
- std::vector< float > * getPosXVec ()

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

- · sf::Color getBodyColor () const
- b2Vec2 getVelocityVec () const
- float getVelocity () const
- · void setCollisionFilter (b2Filter filter) const

3.2.1 Detailed Description

Definition at line 23 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 )
Definition at line 12 of file Car.cc.
                                                            {
00014
           // Create a polygon (octagon)
00015
00016
           auto vertices = createVertices(chromosome.bodyLengths);
00017
00018
          bodv =
00019
               createPolygon(world, x, y, vertices, chromosome.bodyDensity, Config::FRICTION, bodyColor);
00020
00021
00022
           frontWheel_ = createCircle(world, x, y, chromosome.wheelRadius.first,
00023
                                        chromosome.wheelDensity.first, Config::FRICTION, wheelColor);
00024
00025
           // Create another circle
           backWheel_ = createCircle(world, x, y, chromosome.wheelRadius.second,
00026
00027
                                       chromosome.wheelDensity.second, Config::FRICTION, wheelColor);
00028
00029
          b2DistanceJointDef jointDef2;
           jointDef2.bodyA = body_.body;
jointDef2.bodyB = frontWheel_.body;
jointDef2.localAnchorA = vertices[Config::BACK_WHEEL_POS];
jointDef2.localAnchorB = b2Vec2(0.0f, 0.0f);
00030
00031
00032
00033
00034
           jointDef2.maxLength = CarConfig::MAX_JOINT_LENGTH;
00035
           jointDef2.collideConnected = false;
00036
           world->CreateJoint(&jointDef2);
00037
           jointDef2.bodyA = body_.body;
00038
00039
           jointDef2.bodyB = backWheel_.body;
           jointDef2.localAnchorA = vertices[Config::FRONT_WHEEL_POS];
jointDef2.localAnchorB = b2Vec2(0.0f, 0.0f);
00040
00041
           jointDef2.maxLength = CarConfig::MAX_JOINT_LENGTH;
00042
00043
           iointDef2.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;
           filter.maskBits = Config::MASK_BITS;
00050
           this->setCollisionFilter(filter);
00051
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;
```

3.2 Car Class Reference 9

```
00059 velY_ = v_values;

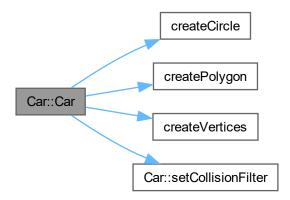
00060

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 ( )
Definition at line 69 of file Car.cc.
00069 { return &backWheel_; }
```

Here is the caller graph for this function:

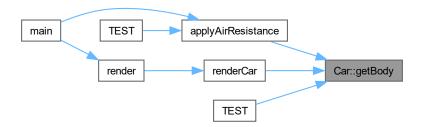


3.2.3.2 getBody()

```
Polygon * Car::getBody ( )

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

Definition at line 83 of file Car.cc.
00083 { return body_.color; }
```

Here is the caller graph for this function:

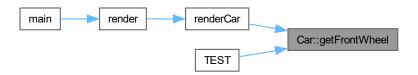


3.2.3.4 getFrontWheel()

```
Circle * Car::getFrontWheel ( )
Definition at line 67 of file Car.cc.
00067 { return &frontWheel_; }
```

3.2 Car Class Reference

Here is the caller graph for this function:



3.2.3.5 getPosX()

```
float Car::getPosX ( ) const

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 ( )
Definition at line 79 of file Car.cc.
00079 { return &posX_; }
```

3.2.3.7 getPosY()

```
float Car::getPosY ( ) const

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 ( )
Definition at line 81 of file Car.cc.
00081 { return &posY_; }

3.2.3.9 getVelocity()
```

```
float Car::getVelocity ( ) const

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

Definition at line 85 of file Car.cc.
00085 { return body_.body->GetLinearVelocity(); }
```

Here is the caller graph for this function:



3.2 Car Class Reference

3.2.3.11 getVeIX()

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

Definition at line 75 of file Car.cc.
00075 { return &velX_; }
```

Here is the caller graph for this function:



3.2.3.12 getVeIY()

```
std::vector< float > * Car::getVelY ( )
Definition at line 77 of file Car.cc.
00077 { return &velY_; }
```

Here is the caller graph for this function:



3.2.3.13 setCollisionFilter()

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);
```

Here is the caller graph for this function:



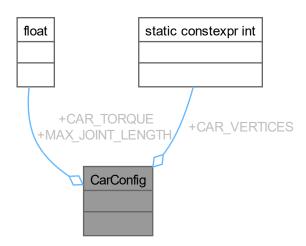
The documentation for this class was generated from the following files:

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

3.3 CarConfig Class Reference

#include <CarConfig.h>

Collaboration diagram for CarConfig:



Static Public Attributes

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

3.3.1 Detailed Description

Definition at line 12 of file CarConfig.h.

3.3.2 Member Data Documentation

3.3.2.1 CAR_TORQUE

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

Definition at line 16 of file CarConfig.h.

3.3.2.2 CAR_VERTICES

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

Definition at line 21 of file CarConfig.h.

3.3.2.3 MAX_JOINT_LENGTH

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

Definition at line 19 of file CarConfig.h.

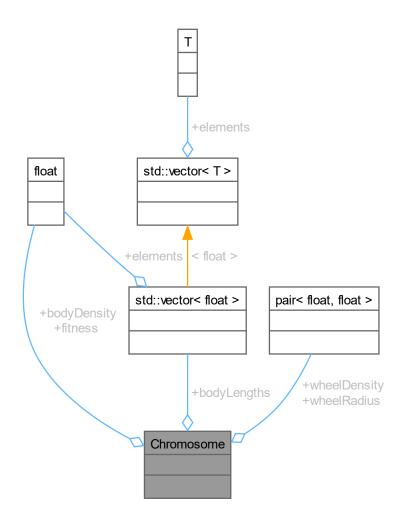
The documentation for this class was generated from the following file:

· config/CarConfig.h

3.4 Chromosome Struct Reference

#include <EvolutionaryAlgorithm.h>

Collaboration diagram for Chromosome:



Public Attributes

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

3.4.1 Detailed Description

Definition at line 21 of file EvolutionaryAlgorithm.h.

3.4.2 Member Data Documentation

3.4.2.1 bodyDensity

float Chromosome::bodyDensity

Definition at line 23 of file EvolutionaryAlgorithm.h.

3.4.2.2 bodyLengths

std::vector<float> Chromosome::bodyLengths

Definition at line 22 of file EvolutionaryAlgorithm.h.

3.4.2.3 fitness

float Chromosome::fitness

Definition at line 26 of file EvolutionaryAlgorithm.h.

3.4.2.4 wheelDensity

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

Definition at line 25 of file EvolutionaryAlgorithm.h.

3.4.2.5 wheelRadius

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

Definition at line 24 of file EvolutionaryAlgorithm.h.

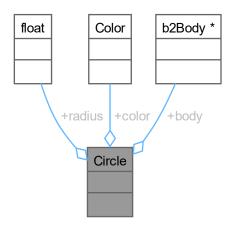
The documentation for this struct was generated from the following file:

• src/EvolutionaryAlgorithm.h

3.5 Circle Struct Reference

#include <Shape.h>

Collaboration diagram for Circle:



Public Attributes

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

3.5.1 Detailed Description

Definition at line 27 of file Shape.h.

3.5.2 Member Data Documentation

3.5.2.1 body

b2Body* Circle::body {}

Definition at line 30 of file Shape.h.

3.5.2.2 color

sf::Color Circle::color

Definition at line 29 of file Shape.h.

3.5.2.3 radius

float Circle::radius {}

Definition at line 28 of file Shape.h.

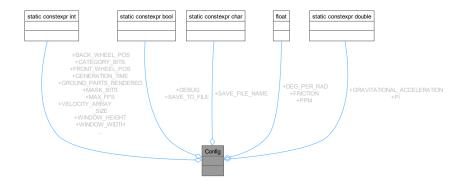
The documentation for this struct was generated from the following file:

• 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

Definition at line 26 of file Config.h.

3.6.2.8 GRAVITATIONAL_ACCELERATION

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

Definition at line 38 of file Config.h.

3.6.2.9 GROUND_PARTS_RENDERED

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

Definition at line 20 of file Config.h.

3.6.2.10 MASK_BITS

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

Definition at line 46 of file Config.h.

3.6.2.11 MAX_FPS

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

Definition at line 18 of file Config.h.

3.6.2.12 PI

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

Definition at line 42 of file Config.h.

3.6.2.13 PPM

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

Definition at line 29 of file Config.h.

3.6.2.14 SAVE_FILE_NAME

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

Definition at line 24 of file Config.h.

3.6.2.15 **SAVE_TO_FILE**

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

Definition at line 23 of file Config.h.

3.6.2.16 VELOCITY_ARRAY_SIZE

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

Definition at line 40 of file Config.h.

3.6.2.17 WINDOW_HEIGHT

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

Definition at line 15 of file Config.h.

3.6.2.18 WINDOW_WIDTH

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

Definition at line 14 of file Config.h.

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

· config/Config.h

3.7 EvolutionaryAlgorithm Class Reference

#include <EvolutionaryAlgorithm.h>

Collaboration diagram for EvolutionaryAlgorithm:

EvolutionaryAlgorithm

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

Public Member Functions

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

3.7.1 Detailed Description

Definition at line 29 of file EvolutionaryAlgorithm.h.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 EvolutionaryAlgorithm()

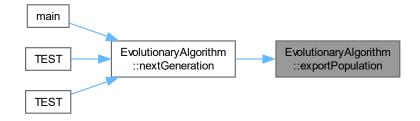
3.7.3 Member Function Documentation

3.7.3.1 exportPopulation()

```
int EvolutionaryAlgorithm::exportPopulation ( )
Definition at line 189 of file Evolutionary Algorithm.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;
               chromosomeJson["bodyLengths"] = chromosome.bodyLengths;
chromosomeJson["bodyDensity"] = chromosome.bodyDensity;
00197
00198
               chromosomeJson["wheelRadius"] = {chromosome.wheelRadius.first,
00199
00200
                                                   chromosome.wheelRadius.second};
               chromosomeJson["wheelDensity"] = {chromosome.wheelDensity.first,
00201
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;
```

Here is the caller graph for this function:

00219 }



3.7.3.2 getGeneration()

```
int EvolutionaryAlgorithm::getGeneration ( ) const [inline]
Definition at line 45 of file EvolutionaryAlgorithm.h.
00045 { return generation_; }
```

Here is the caller graph for this function:

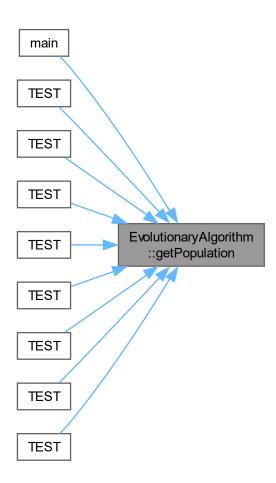


3.7.3.3 getPopulation()

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

Definition at line 40 of file EvolutionaryAlgorithm.h. 00040 { return population_; }

Here is the caller graph for this function:



3.7.3.4 getPopulationSize()

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

Definition at line 46 of file EvolutionaryAlgorithm.h.

```
00046 { return populationSize_; }
```

3.7.3.5 mutate()

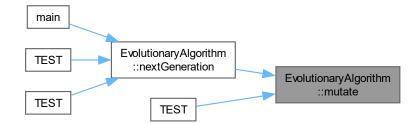
```
void EvolutionaryAlgorithm::mutate ( )
```

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) {</pre>
                      length += dist(gen) * EvolutionaryAlgorithmConfig::MUTATION_FACTOR_BODY_LENGTHS;
00093
                      length = std::max(length, EvolutionaryAlgorithmConfig::MIN_BODY_LENGTH);
00094
00095
                      length = std::min(length, EvolutionaryAlgorithmConfig::MAX_BODY_LENGTH);
00096
                  }
00097
              }
00098
00099
              // Mutate bodyDensity
00100
              if (dist(gen) < EvolutionaryAlgorithmConfig::MUTATION_RATE_BODY_DENSITY) {</pre>
                  chrom.bodyDensity +=
00101
                      dist(gen) * EvolutionaryAlgorithmConfig::MUTATION_FACTOR_BODY_DENSITY;
00102
00103
00104
                  chrom.bodyDensity =
00105
                      std::max(chrom.bodyDensity, EvolutionaryAlgorithmConfig::MIN_BODY_DENSITY);
                  chrom.bodyDensity =
00106
                      std::min(chrom.bodyDensity, EvolutionaryAlgorithmConfig::MAX_BODY_DENSITY);
00107
00108
              }
00109
00110
              // Mutate wheelRadius
00111
              if (dist(gen) < EvolutionaryAlgorithmConfig::MUTATION_RATE_WHEEL_RADIUS) {</pre>
00112
                  chrom.wheelRadius.first +=
                      dist(gen) * EvolutionaryAlgorithmConfig::MUTATION_FACTOR_WHEEL_RADIUS;
00113
                  chrom.wheelRadius.first =
00114
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) {</pre>
                  chrom.wheelRadius.second +=
00120
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) {</pre>
00130
                  chrom.wheelDensity.first +=
00131
                      dist(gen) * EvolutionaryAlgorithmConfig::MUTATION_FACTOR_WHEEL_DENSITY;
                  chrom.wheelDensity.first =
00132
                      std::max(chrom.wheelDensity.first, EvolutionaryAlgorithmConfig::MIN_WHEEL_DENSITY);
00133
00134
                  chrom.wheelDensity.first
00135
                      std::min(chrom.wheelDensity.first, EvolutionaryAlgorithmConfig::MAX_WHEEL_DENSITY);
00136
00137
              if (dist(gen) < EvolutionaryAlgorithmConfig::MUTATION_RATE_WHEEL_DENSITY) {</pre>
                  chrom.wheelDensity.second +=
00138
                      dist(gen) * EvolutionaryAlgorithmConfig::MUTATION_FACTOR_WHEEL_DENSITY;
00139
00140
                  chrom.wheelDensitv.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:



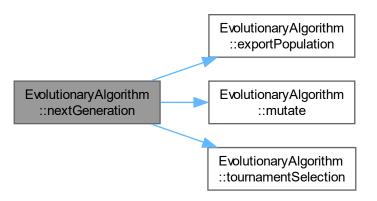
3.7.3.6 nextGeneration()

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

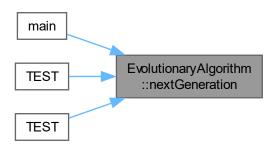
Definition at line 180 of file EvolutionaryAlgorithm.cc.

```
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()

Definition at line 44 of file EvolutionaryAlgorithm.h.
00044 { population_[index].fitness = fitness; }

Here is the caller graph for this function:

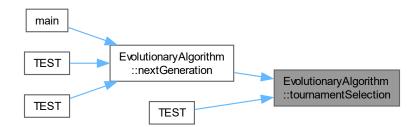


3.7.3.8 tournamentSelection()

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

```
Definition at line 169 of file EvolutionaryAlgorithm.cc.
```

Here is the caller graph for this function:



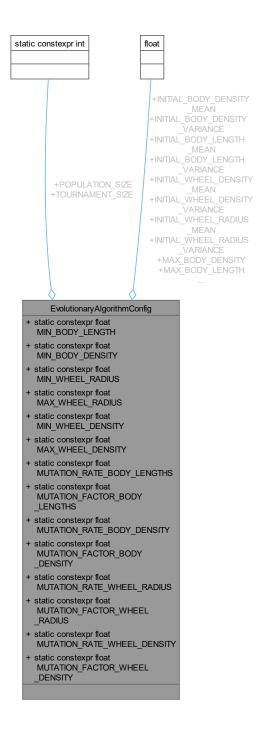
The documentation for this class was generated from the following files:

- 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 = 15

- static constexpr float MIN_BODY_LENGTH = 1.0f
- static constexpr float MAX_BODY_LENGTH = 5.0f
- static constexpr float MIN_BODY_DENSITY = 10.0f
- static constexpr float MAX BODY DENSITY = 1000.0f
- static constexpr float MIN_WHEEL_RADIUS = 2.0f
- static constexpr float MAX_WHEEL_RADIUS = 40.0f
- static constexpr float MIN_WHEEL_DENSITY = 10.0f
- static constexpr float MAX_WHEEL_DENSITY = 1000.0f
- static constexpr float INITIAL_BODY_LENGTH_MEAN = 3.0f
- static constexpr float INITIAL_BODY_LENGTH_VARIANCE = 1.0f
- static constexpr float INITIAL BODY DENSITY MEAN = 100.0f
- static constexpr float INITIAL_BODY_DENSITY_VARIANCE = 100.0f
- static constexpr float INITIAL_WHEEL_RADIUS_MEAN = 25.0f
- static constexpr float INITIAL_WHEEL_RADIUS_VARIANCE = 10.0f
- static constexpr float INITIAL_WHEEL_DENSITY_MEAN = 100.0f
- static constexpr float INITIAL_WHEEL_DENSITY_VARIANCE = 100.0f
- static constexpr float MUTATION_RATE_BODY_LENGTHS = 0.1f
- static constexpr float MUTATION_FACTOR_BODY_LENGTHS = 0.5f
- static constexpr float MUTATION_RATE_BODY_DENSITY = 0.2f
- static constexpr float MUTATION_FACTOR_BODY_DENSITY = 20.0f
- static constexpr float MUTATION_RATE_WHEEL_RADIUS = 0.3f
- static constexpr float MUTATION_FACTOR_WHEEL_RADIUS = 2.0f
- static constexpr float MUTATION_RATE_WHEEL_DENSITY = 0.1f
- static constexpr float MUTATION_FACTOR_WHEEL_DENSITY = 20.0f
- static constexpr int TOURNAMENT_SIZE = 3

3.8.1 Detailed Description

Definition at line 12 of file EvolutionaryAlgorithmConfig.h.

3.8.2 Member Data Documentation

3.8.2.1 INITIAL_BODY_DENSITY_MEAN

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

Definition at line 34 of file EvolutionaryAlgorithmConfig.h.

3.8.2.2 INITIAL_BODY_DENSITY_VARIANCE

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

Definition at line 35 of file EvolutionaryAlgorithmConfig.h.

3.8.2.3 INITIAL BODY LENGTH MEAN

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

Definition at line 31 of file EvolutionaryAlgorithmConfig.h.

3.8.2.4 INITIAL_BODY_LENGTH_VARIANCE

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

Definition at line 32 of file EvolutionaryAlgorithmConfig.h.

3.8.2.5 INITIAL_WHEEL_DENSITY_MEAN

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

Definition at line 40 of file EvolutionaryAlgorithmConfig.h.

3.8.2.6 INITIAL_WHEEL_DENSITY_VARIANCE

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

Definition at line 41 of file EvolutionaryAlgorithmConfig.h.

3.8.2.7 INITIAL WHEEL RADIUS MEAN

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

Definition at line 37 of file EvolutionaryAlgorithmConfig.h.

3.8.2.8 INITIAL_WHEEL_RADIUS_VARIANCE

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

Definition at line 38 of file EvolutionaryAlgorithmConfig.h.

3.8.2.9 MAX_BODY_DENSITY

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

Definition at line 22 of file EvolutionaryAlgorithmConfig.h.

3.8.2.10 MAX_BODY_LENGTH

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

Definition at line 19 of file EvolutionaryAlgorithmConfig.h.

3.8.2.11 MAX_WHEEL_DENSITY

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

Definition at line 28 of file EvolutionaryAlgorithmConfig.h.

3.8.2.12 MAX_WHEEL_RADIUS

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

Definition at line 25 of file EvolutionaryAlgorithmConfig.h.

3.8.2.13 MIN_BODY_DENSITY

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

Definition at line 21 of file EvolutionaryAlgorithmConfig.h.

3.8.2.14 MIN_BODY_LENGTH

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

Definition at line 18 of file EvolutionaryAlgorithmConfig.h.

3.8.2.15 MIN_WHEEL_DENSITY

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

Definition at line 27 of file EvolutionaryAlgorithmConfig.h.

3.8.2.16 MIN_WHEEL_RADIUS

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

Definition at line 24 of file EvolutionaryAlgorithmConfig.h.

3.8.2.17 MUTATION_FACTOR_BODY_DENSITY

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

Definition at line 49 of file EvolutionaryAlgorithmConfig.h.

3.8.2.18 MUTATION_FACTOR_BODY_LENGTHS

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

Definition at line 46 of file EvolutionaryAlgorithmConfig.h.

3.8.2.19 MUTATION FACTOR WHEEL DENSITY

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

Definition at line 55 of file EvolutionaryAlgorithmConfig.h.

3.8.2.20 MUTATION_FACTOR_WHEEL_RADIUS

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

Definition at line 52 of file EvolutionaryAlgorithmConfig.h.

3.8.2.21 MUTATION RATE BODY DENSITY

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

Definition at line 48 of file EvolutionaryAlgorithmConfig.h.

3.8.2.22 MUTATION_RATE_BODY_LENGTHS

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

Definition at line 45 of file EvolutionaryAlgorithmConfig.h.

3.8.2.23 MUTATION_RATE_WHEEL_DENSITY

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

Definition at line 54 of file EvolutionaryAlgorithmConfig.h.

3.8.2.24 MUTATION_RATE_WHEEL_RADIUS

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

Definition at line 51 of file EvolutionaryAlgorithmConfig.h.

3.8.2.25 POPULATION_SIZE

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

Definition at line 15 of file EvolutionaryAlgorithmConfig.h.

3.8.2.26 TOURNAMENT_SIZE

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

Definition at line 58 of file EvolutionaryAlgorithmConfig.h.

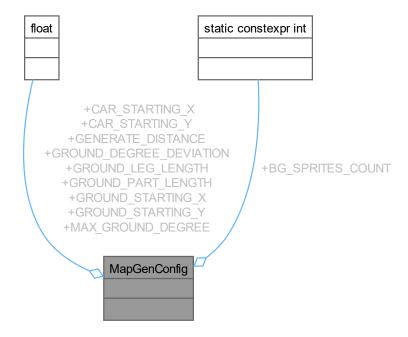
The documentation for this class was generated from the following file:

· config/EvolutionaryAlgorithmConfig.h

3.9 MapGenConfig Class Reference

#include <MapGenConfig.h>

Collaboration diagram for MapGenConfig:



Static Public Attributes

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

3.9.1 Detailed Description

Definition at line 12 of file MapGenConfig.h.

3.9.2 Member Data Documentation

3.9.2.1 BG_SPRITES_COUNT

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

Definition at line 19 of file MapGenConfig.h.

3.9.2.2 CAR_STARTING_X

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

Definition at line 21 of file MapGenConfig.h.

3.9.2.3 CAR_STARTING_Y

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

Definition at line 22 of file MapGenConfig.h.

3.9.2.4 GENERATE_DISTANCE

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

Definition at line 14 of file MapGenConfig.h.

3.9.2.5 GROUND_DEGREE_DEVIATION

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

Definition at line 25 of file MapGenConfig.h.

3.9.2.6 GROUND_LEG_LENGTH

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

Definition at line 17 of file MapGenConfig.h.

3.9.2.7 GROUND_PART_LENGTH

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

Definition at line 18 of file MapGenConfig.h.

3.9.2.8 GROUND_STARTING_X

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

Definition at line 15 of file MapGenConfig.h.

3.9.2.9 GROUND_STARTING_Y

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

Definition at line 16 of file MapGenConfig.h.

3.9.2.10 MAX_GROUND_DEGREE

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

Definition at line 26 of file MapGenConfig.h.

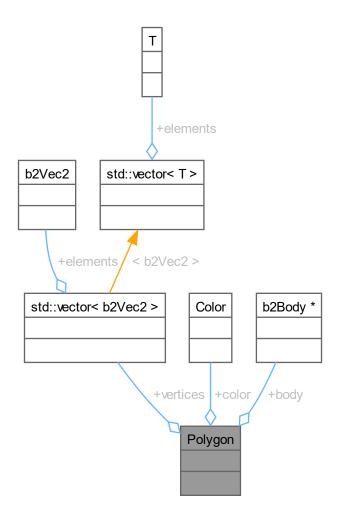
The documentation for this class was generated from the following file:

• config/MapGenConfig.h

3.10 Polygon Struct Reference

#include <Shape.h>

Collaboration diagram for Polygon:



Public Attributes

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

3.10.1 Detailed Description

Definition at line 33 of file Shape.h.

3.10.2 Member Data Documentation

3.10.2.1 body

b2Body* Polygon::body

Definition at line 36 of file Shape.h.

3.10.2.2 color

sf::Color Polygon::color

Definition at line 35 of file Shape.h.

3.10.2.3 vertices

std::vector<b2Vec2> Polygon::vertices

Definition at line 34 of file Shape.h.

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

• src/Shape.h

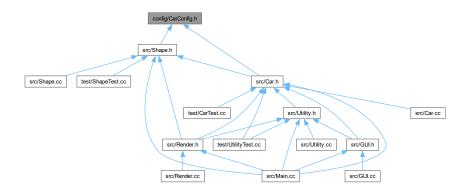
Chapter 4

File Documentation

4.1 config/CarConfig.h File Reference

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

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



Classes

• class CarConfig

4.1.1 Detailed Description

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

Author

Mateusz Krakowski

Date

2023-06-06

Definition in file CarConfig.h.

4.2 CarConfig.h

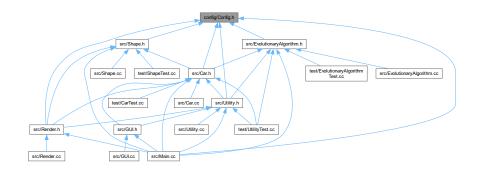
Go to the documentation of this file.

```
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
static constexpr float CAR_TORQUE = 2000.0f;
00015
00016
00017
00018
            static constexpr float MAX_JOINT_LENGTH = 0.01f;
00019
            // Number of vertices in a car's body polygon
static constexpr int CAR_VERTICES = 8;
00020
00021
00022
00023 };
00024
00025 #endif // CARCONFIG_H
```

4.3 config/Config.h File Reference

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

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



Classes

· class Config

4.3.1 Detailed Description

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

Authors

Jakub Marcowski, Mateusz Krakowski

Date

2023-06-06

Definition in file Config.h.

4.4 Config.h 43

4.4 Config.h

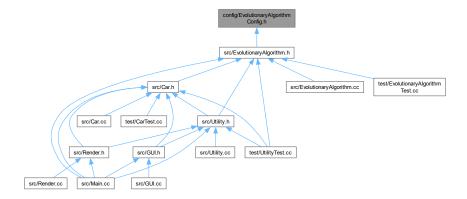
Go to the documentation of this file.

```
00001
00009 #ifndef CONFIG H
00010 #define CONFIG_H
00011
00012 class Config {
00013
          public:
           static constexpr int WINDOW_WIDTH = 1280;
static constexpr int WINDOW_HEIGHT = 720;
00014
00015
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
            // Exporting to file
00022
           static constexpr bool SAVE_TO_FILE = true;
static constexpr char SAVE_FILE_NAME[] = "evoRacerOutput.json";
00023
00024
00025
00026
            static constexpr int GENERATION_TIME =
            3000; // in frames, about 60 frames per second => 50 seconds // Pixels per meter. Box2D uses metric units, so we need PPM for conversion purposes static constexpr float PPM = 30.0F;
00027
00028
00029
00030
00031
            // SFML uses degrees for angles while Box2D uses radians
00032
            static constexpr float DEG_PER_RAD = 57.2957795F;
00033
           // Draw debug geometry
static constexpr bool DEBUG = true;
00034
00035
00036
00037
            // Physics
00038
            static constexpr double GRAVITATIONAL_ACCELERATION = -9.81f;
           static constexpr float FRICTION = 0.3f;
static constexpr int VELOCITY_ARRAY_SIZE = 1000;
00039
00040
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 };
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

2023-06-06

Definition in file EvolutionaryAlgorithmConfig.h.

4.6 EvolutionaryAlgorithmConfig.h

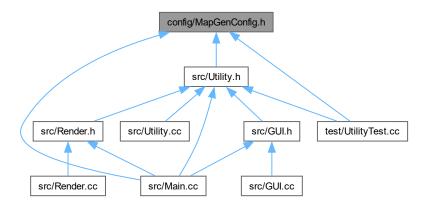
Go to the documentation of this file.

```
00001
00009 #ifndef EVOLUTIONARY_ALGORITHM_CONFIG_H
00010 #define EVOLUTIONARY_ALGORITHM_CONFIG_H
00012 class EvolutionaryAlgorithmConfig {
        public:
00013
00014
          // \ {\tt Evolutionary \ algorithm \ parameters}
          static constexpr int POPULATION_SIZE = 15;
00015
00016
          // Boundaries for the chromosomes
00018
          static constexpr float MIN_BODY_LENGTH = 1.0f;
00019
          static constexpr float MAX_BODY_LENGTH = 5.0f;
00020
          static constexpr float MIN BODY DENSITY = 10.0f;
00021
00022
          static constexpr float MAX_BODY_DENSITY = 1000.0f;
00023
00024
          static constexpr float MIN_WHEEL_RADIUS = 2.0f;
00025
          static constexpr float MAX_WHEEL_RADIUS = 40.0f;
00026
          static constexpr float MIN_WHEEL_DENSITY = 10.0f;
00027
00028
          static constexpr float MAX_WHEEL_DENSITY = 1000.0f;
00029
00030
          // Population initialization hyper parameters
00031
          static constexpr float INITIAL_BODY_LENGTH_MEAN = 3.0f;
00032
          static constexpr float INITIAL_BODY_LENGTH_VARIANCE = 1.0f;
00033
          static constexpr float INITIAL_BODY_DENSITY_MEAN = 100.0f;
00034
00035
          static constexpr float INITIAL_BODY_DENSITY_VARIANCE = 100.0f;
00036
00037
          static constexpr float INITIAL_WHEEL_RADIUS_MEAN = 25.0f;
00038
          static constexpr float INITIAL_WHEEL_RADIUS_VARIANCE = 10.0f;
00039
          static constexpr float INITIAL_WHEEL_DENSITY_MEAN = 100.0f;
00040
00041
          static constexpr float INITIAL_WHEEL_DENSITY_VARIANCE = 100.0f;
00042
00043
          // Mutation hyper parameters
00044
          static constexpr float MUTATION_RATE_BODY_LENGTHS = 0.1f;
static constexpr float MUTATION_FACTOR_BODY_LENGTHS = 0.5f;
00045
00046
00047
00048
          static constexpr float MUTATION_RATE_BODY_DENSITY = 0.2f;
          static constexpr float MUTATION_FACTOR_BODY_DENSITY = 20.0f;
00049
00050
          static constexpr float MUTATION_RATE_WHEEL_RADIUS = 0.3f;
00051
00052
          static constexpr float MUTATION_FACTOR_WHEEL_RADIUS = 2.0f;
00053
          static constexpr float MUTATION_RATE_WHEEL_DENSITY = 0.1f;
00054
          static constexpr float MUTATION_FACTOR_WHEEL_DENSITY = 20.0f;
00055
00056
00057
          // Selection hyper parameters
          static constexpr int TOURNAMENT_SIZE = 3; // Has to be equal to or lesser than POPULATION_SIZE
00058
00059 };
00061 #endif // EVOLUTIONARY_ALGORITHM_CONFIG_H
```

4.7 config/MapGenConfig.h File Reference

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

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



Classes

· class MapGenConfig

4.7.1 Detailed Description

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

Author

Mateusz Krakowski

Date

2023-06-06

Definition in file MapGenConfig.h.

4.8 MapGenConfig.h

Go to the documentation of this file.

```
00009 #ifndef MAPGENCONFIG_H
00010 #define MAPGENCONFIG_H
00011
00012 class MapGenConfig {
        public:
00013
          static constexpr float GENERATE_DISTANCE = 666.0;
00014
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;
          static constexpr float GROUND_PART_LENGTH = 1.5;
00018
00019
          static constexpr int BG_SPRITES_COUNT = 5;
00020
          static constexpr float CAR_STARTING_X = 250.0;
00021
00022
          static constexpr float CAR_STARTING_Y = 650.0;
00023
00024
          \ensuremath{//} Change the mapgen behaviour here
          static constexpr float GROUND_DEGREE_DEVIATION = 12.0f;
static constexpr float MAX_GROUND_DEGREE = 50.0f;
00025
00026
00027
00028
00029 };
00030
00031 #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)

4.9.1 Detailed Description

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

Authors

Jakub Marcowski, Mateusz Krakowski

Date

2023-06-06

Definition in file Car.cc.

4.10 Car.cc 47

4.9.2 Function Documentation

4.9.2.1 createVertices()

```
std::vector< b2Vec2 > createVertices (
                std::vector< float > lengths )
Definition at line 95 of file Car.cc.
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++) {</pre>
00101
               angles.push_back(360.0f / lengths.size());
00102
00103
           ^{\prime}/^{\prime} so that the wheels are set properly (that is - parallel to the ground)
           float angle = ((180.0f + (angles.back() / 2)) / 180.0f) * Config::PI;
00104
00105
00106
           for (int i = 0; i < lengths.size(); i++) {
               vertices.emplace_back(lengths[i] * cos(angle), lengths[i] * sin(angle));
angle += (angles[i] / 180.0f) * Config::PI;
00107
00108
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
           frontWheel_ = createCircle(world, x, y, chromosome.wheelRadius.first,
00022
00023
                                        chromosome.wheelDensity.first, Config::FRICTION, wheelColor);
00024
00025
           // Create another circle
00026
           backWheel_ = createCircle(world, x, y, chromosome.wheelRadius.second,
                                       chromosome.wheelDensity.second, Config::FRICTION, wheelColor);
00027
00028
00029
           b2DistanceJointDef jointDef2;
           jointDef2.bodyA = body_.body;
jointDef2.bodyB = frontWheel_.body;
00030
00031
           jointDef2.localAnchorA = vertices[Config::BACK_WHEEL_POS];
jointDef2.localAnchorB = b2Vec2(0.0f, 0.0f);
00032
00033
           jointDef2.maxLength = CarConfig::MAX_JOINT_LENGTH;
00034
00035
           iointDef2.collideConnected = false;
00036
           world->CreateJoint(&jointDef2);
00037
```

```
jointDef2.bodyA = body_.body;
00039
           jointDef2.bodyB = backWheel_.body;
           jointDef2.localAnchorA = vertices[Config::FRONT_WHEEL_POS];
jointDef2.localAnchorB = b2Vec2(0.0f, 0.0f);
00040
00041
           jointDef2.maxLength = CarConfig::MAX_JOINT_LENGTH;
jointDef2.collideConnected = false;
00042
00043
00044
           world->CreateJoint(&jointDef2);
00045
00046
           // Make cars pass through each-other
00047
           // by setting collision filtering
           b2Filter filter;
00048
           filter.categoryBits = Config::CATEGORY_BITS;
00049
00050
           filter.maskBits = Config::MASK_BITS;
00051
           this->setCollisionFilter(filter);
00052
          std::vector<float> v_axis(Config::VELOCITY_ARRAY_SIZE);
std::vector<float> v_values(Config::VELOCITY_ARRAY_SIZE);
00053
00054
00055
00056
           std::iota(std::begin(v_axis), std::end(v_axis), 1);
00057
00058
           velX_ = v_axis;
00059
           velY_ = v_values;
00060
          posX_ = v_axis;
posY_ = v_values;
00061
00062
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_; }
08000
00081 std::vector<float>* Car::getPosYVec() { return &posY_; }
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());
for (int i = 0; i < lengths.size(); i++) {</pre>
00100
               angles.push_back(360.0f / lengths.size());
00101
00102
          // so that the wheels are set properly (that is - parallel to the ground) float angle = ((180.0f + (angles.back() / 2)) / 180.0f) * Config::PI;
00103
00104
00105
00106
           for (int i = 0; i < lengths.size(); i++) {</pre>
               vertices.emplace_back(lengths[i] * cos(angle), lengths[i] * sin(angle));
00107
00108
               angle += (angles[i] / 180.0f) * Config::PI;
00109
00110
           return vertices;
00111 }
```

4.11 src/Car.h File Reference

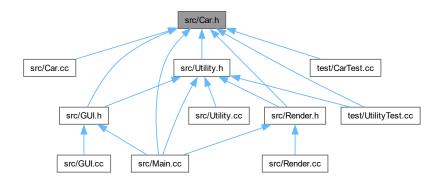
Header file for the Car class.

```
#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

Typedefs

typedef std::shared_ptr< b2World > b2WorldPtr

Functions

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

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

Here is the caller graph for this function:



4.12 Car.h

Go to the documentation of this file.

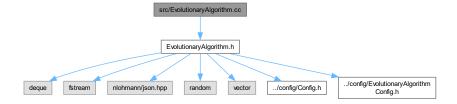
```
00001
00009 #ifndef CAR_H
00010 #define CAR_H
00011
00012 #include <numeric>
00013
00014 #include "box2d/box2d.h"
00015
00016 #include "../config/Config.h"
00017 #include "../config/CarConfig.h"
00018 #include "Shape.h"
00019 #include "EvolutionaryAlgorithm.h"
00020
```

```
00021 typedef std::shared_ptr<b2World> b2WorldPtr;
00023 class Car {
00024
       private:
         b2WorldPtr world_;
00025
         Polygon body_;
00026
         Circle frontWheel_;
00028
         Circle backWheel_;
00029
         std::vector<float> velX_;
00030
         std::vector<float> velY ;
00031
         std::vector<float> posX_;
00032
         std::vector<float> posY_;
00033
00034
       public:
00035
         Car(const b2WorldPtr& world, float x, float y, const Chromosome& chromosome,
00036
             sf::Color bodyColor, sf::Color wheelColor);
00037
00038
         Polygon* getBody();
         Circle* getFrontWheel();
00039
00040
         Circle* getBackWheel();
00041
         float getPosX() const;
00042
         float getPosY() const;
00043
         std::vector<float>* getVelX();
         std::vector<float>* getVelY();
00044
00045
         std::vector<float>* getPosXVec();
00046
         std::vector<float>* getPosYVec();
00047
          sf::Color getBodyColor() const;
00048
         b2Vec2 getVelocityVec() const;
00049
         float getVelocity() const;
          void setCollisionFilter(b2Filter filter) const;
00050
00051 };
00052
00053 std::vector<b2Vec2> createVertices(std::vector<float> lengths);
00054
00055 #endif
```

4.13 src/EvolutionaryAlgorithm.cc File Reference

Implementation file for Evolutionary Algorithm 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());
          std::normal_distribution<float> dist(0.0, 1.0);
00022
00023
          // add variation and mean
00024
          for (int i = 0; i < populationSize_; ++i) {</pre>
00025
              Chromosome chrom;
00026
              // TODO: change the 8
00027
              for (int p = 0; p < 8; ++p) {
                  float length = dist(gen) * EvolutionaryAlgorithmConfig::INITIAL_BODY_LENGTH_VARIANCE +
00028
                  EvolutionaryAlgorithmConfig::MITIAL_BODY_LENGTH_MEAN;
length = std::max(length, EvolutionaryAlgorithmConfig::MIN_BODY_LENGTH);
00029
00030
                  length = std::min(length, EvolutionaryAlgorithmConfig::MAX_BODY_LENGTH);
00031
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.bodvDensitv =
00043
                  std::min(chrom.bodyDensity, EvolutionaryAlgorithmConfig::MAX_BODY_DENSITY);
00044
00045
              // initialize wheelRadius
00046
00047
              chrom.wheelRadius.first =
                  dist(gen) * EvolutionaryAlgorithmConfig::INITIAL_WHEEL_RADIUS_VARIANCE +
00048
                  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 +
                  EvolutionaryAlgorithmConfig::INITIAL_WHEEL_RADIUS_MEAN;
00056
00057
              chrom.wheelRadius.second =
00058
                  std::max(chrom.wheelRadius.second, EvolutionaryAlgorithmConfig::MIN_WHEEL_RADIUS);
              chrom.wheelRadius.second =
00059
00060
                  std::min(chrom.wheelRadius.second, EvolutionaryAlgorithmConfig::MAX_WHEEL_RADIUS);
00061
00062
              // Initialize wheelDensity
00063
00064
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
                  \verb|std::min(chrom.wheelDensity.second, EvolutionaryAlgorithmConfig::MAX\_WHEEL\_DENSITY)|;\\
00079
08000
              population_.push_back(chrom);
00081
          }
00082 }
00083
00084 void EvolutionaryAlgorithm::mutate() {
00085
         std::random device rd;
00086
          std::mt19937 gen(rd());
          std::normal_distribution<float> dist(0.0, 1.0);
00088
00089
          for (auto& chrom : population_) {
00090
              // Mutate bodyLengths
```

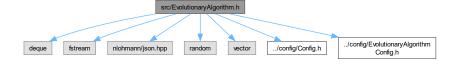
```
00091
              for (auto& length : chrom.bodyLengths) {
00092
                  if (dist(gen) < EvolutionaryAlgorithmConfig::MUTATION_RATE_BODY_LENGTHS) {</pre>
00093
                      length += dist(gen) * EvolutionaryAlgorithmConfig::MUTATION_FACTOR_BODY_LENGTHS;
                      length = std::max(length, EvolutionaryAlgorithmConfig::MIN_BODY_LENGTH);
00094
00095
                      length = std::min(length, EvolutionaryAlgorithmConfig::MAX_BODY_LENGTH);
00096
                  }
00097
              }
00098
              // Mutate bodyDensity
00099
00100
              if (dist(gen) < EvolutionaryAlgorithmConfig::MUTATION_RATE_BODY_DENSITY) {</pre>
                  chrom.bodyDensity +=
00101
                      dist(gen) * EvolutionaryAlgorithmConfig::MUTATION_FACTOR_BODY_DENSITY;
00102
00103
00104
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
              if (dist(gen) < EvolutionaryAlgorithmConfig::MUTATION_RATE_WHEEL_RADIUS) {</pre>
00111
                  chrom.wheelRadius.first +=
00112
00113
                      dist(gen) * EvolutionaryAlgorithmConfig::MUTATION_FACTOR_WHEEL_RADIUS;
                  chrom.wheelRadius.first =
00114
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) {</pre>
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) {</pre>
00130
                 chrom.wheelDensity.first +=
                      dist(gen) * EvolutionaryAlgorithmConfig::MUTATION_FACTOR_WHEEL_DENSITY;
00131
                  chrom.wheelDensity.first =
00132
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) {</pre>
00138
                  chrom.wheelDensity.second +=
                     dist(gen) * EvolutionaryAlgorithmConfig::MUTATION_FACTOR_WHEEL_DENSITY;
00139
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) {</pre>
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) {</pre>
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)</pre>
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
00186
           ++generation_;
00187 }
00188
00189 int EvolutionaryAlgorithm::exportPopulation() {
          nlohmann::json jsonData;
jsonData["generation"] = generation_;
00190
00191
00192
00193
           std::deque<nlohmann::json> populationData;
00194
00195
           for (const auto& chromosome : population_) {
00196
               nlohmann::json chromosomeJson;
00197
               chromosomeJson["bodyLengths"] = chromosome.bodyLengths;
               chromosomeJson["bodyDensity"] = chromosome.bodyDensity;
chromosomeJson["wheelRadius"] = {chromosome.wheelRadius.first,
00198
00199
00200
                                                   chromosome.wheelRadius.second};
               chromosomeJson["wheelDensity"] = {chromosome.wheelDensity.first,
00201
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 }
```

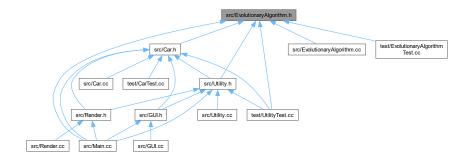
4.15 src/EvolutionaryAlgorithm.h File Reference

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 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.

```
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
00021 struct Chromosome {
00022
            std::vector<float> bodyLengths;
            float bodyDensity;
std::pair<float, float> wheelRadius;
std::pair<float, float> wheelDensity;
float fitness;
00023
00024
00025
00026
00027 };
```

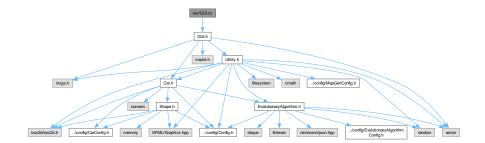
```
00028
00029 class EvolutionaryAlgorithm {
          private:
00030
00031
            unsigned long int generation_;
            unsigned long int populationSize_; std::vector<Chromosome> population_;
00032
00033
00034
            bool saveToFile_ = false;
00035
            Chromosome tournament();
00036
            void initializePopulation();
00037
00038
           public:
            explicit EvolutionaryAlgorithm(int populationSize, bool saveToFile = false);
00039
            std::vector<Chromosome> getPopulation() { return population_; }
00041
            void mutate();
00042
            void tournamentSelection();
00043
            void nextGeneration();
            void setFitness(int index, float fitness) { population_[index].fitness = fitness; }
int getGeneration() const { return generation_; }
int getPopulationSize() const { return populationSize_; }
00044
00045
00047
            int exportPopulation();
00048 };
00049
00050 #endif
```

4.17 src/GUI.cc File Reference

File containing GUI functions.

```
#include "GUI.h"
```

Include dependency graph for GUI.cc:



Functions

- $\bullet \ \ \mathsf{void} \ \mathsf{renderVelocityPlot} \ (\mathsf{std} :: \mathsf{vector} < \mathsf{Car} > \& \mathsf{cars}, \ \mathsf{bool} \ \mathsf{paused}) \\$
 - Renders velocity plot.
- void renderPositionPlot (std::vector< Car > &cars, bool paused)

Renders position plot.

4.17.1 Detailed Description

File containing GUI functions.

Author

Jakub Marcowski

Date

2023-06-06

Definition in file GUI.cc.

4.17.2 Function Documentation

4.17.2.1 renderPositionPlot()

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

Renders position plot.

Parameters

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

Definition at line 38 of file GUI.cc.

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

Here is the call graph for this function:



Here is the caller graph for this function:



4.17.2.2 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.

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

4.18 GUI.cc 59

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();
          if (ImPlot::BeginPlot("Velocity")) {
00014
00015
              ImPlot::SetupLegend(ImPlotLocation_West, ImPlotLegendFlags_Outside);
              for (int i = 0; i < cars.size(); ++i) {
    char i_str[11]; // 10 digits + null
    sprintf(i_str, "%d", i);</pre>
00016
00017
00018
00019
00020
                  if (!paused) {
                      cars[i].getVelX()->push_back(cars[i].getVelX()->back() + 1);
00021
00022
                      cars[i].getVelY()->push_back(cars[i].getVelocity());
00023
                 00024
00025
00026
                  std::vector<float> v_values_crop = std::vector<float>(
    cars[i].getVelY()->end() - Config::VELOCITY_ARRAY_SIZE, cars[i].getVelY()->end());
00027
                  ImPlot::PushStyleColor(ImPlotCol_Line, SFMLColorToImVec4(cars[i].getBodyColor()));
00028
                  ImPlot::PlotLine(i_str, &(v_axis_crop[0]), &(v_values_crop[0]),
00030
                                   Config::VELOCITY_ARRAY_SIZE);
00031
                  ImPlot::PopStyleColor();
00032
              ImPlot::EndPlot();
00033
00034
00035
          ImGui::End();
00036 }
00037
00041
          if (ImPlot::BeginPlot("Position")) {
00042
              ImPlot::SetupLegend(ImPlotLocation_West, ImPlotLegendFlags_Outside);
00043
              for (int i = 0; i < cars.size(); ++i) {</pre>
                 char i_str[11];
00044
                  sprintf(i_str, "%d", i);
00045
00046
00047
00048
                      cars[i].getPosXVec()->push_back(cars[i].getPosXVec()->back() + 1);
```

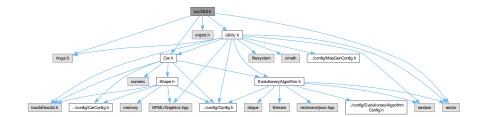
```
cars[i].getPosYVec()->push_back(cars[i].getBody()->body->GetPosition().x);
00050
00051
               std::vector<float> v_axis_crop =
                  00052
00053
                                  cars[i].getPosXVec()->end());
               std::vector<float> v_values_crop =
00054
00055
                  std::vector<float>(cars[i].getPosYVec()->end() - Config::VELOCITY_ARRAY_SIZE,
00056
                                  cars[i].getPosYVec()->end());
00057
               ImPlot:: PushStyleColor(ImPlotCol\_Line, \ \ \underline{SFMLColorToImVec4} \ (cars[i].getBodyColor())); \\
               00058
00059
00060
               ImPlot::PopStyleColor();
00061
00062
           ImPlot::EndPlot();
00063
00064
        ImGui::End();
00065 3
```

4.19 src/GUI.h File Reference

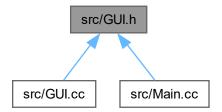
Header for a file containing GUI functions.

```
#include <vector>
#include "imgui.h"
#include "implot.h"
#include "Car.h"
#include "Utility.h"
```

Include dependency graph for GUI.h:



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



Functions

- void render VelocityPlot (std::vector< $\mbox{Car} > \mbox{\&cars},$ bool paused)
 - Renders velocity plot.
- void renderPositionPlot (std::vector< Car > &cars, bool paused)

Renders position plot.

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 renderPositionPlot()

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

Renders position plot.

Parameters

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

Definition at line 38 of file GUI.cc.

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

Here is the call graph for this function:



Here is the caller graph for this function:



4.19.2.2 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
         ImGui::Begin("Cars' Velocity");
00012
00013
         ImPlot::SetNextAxesToFit();
00014
         if (ImPlot::BeginPlot("Velocity")) {
00015
            ImPlot::SetupLegend(ImPlotLocation_West, ImPlotLegendFlags_Outside);
            for (int i = 0; i < cars.size(); ++i) {
   char i_str[11]; // 10 digits + null
   sprintf(i_str, "%d", i);</pre>
00016
00017
00018
00019
00020
                if (!paused) {
00021
                    cars[i].getVelX()->push_back(cars[i].getVelX()->back() + 1);
00022
                    cars[i].getVelY()->push_back(cars[i].getVelocity());
00023
00024
                00025
00026
                std::vector<float> v_values_crop = std::vector<float>(
00027
                    cars[i].getVelY()->end() - Config::VELOCITY_ARRAY_SIZE, cars[i].getVelY()->end());
                ImPlot::PushStyleColor(ImPlotCol_Line, SFMLColorToImVec4(cars[i].getBodyColor()));
00028
                00029
00030
                ImPlot::PopStyleColor();
00031
```

4.20 GUI.h 63

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 "Utility.h"
00019
00025 void renderVelocityPlot(std::vector<Car>& cars, bool paused);
00026
00033 void renderPositionPlot(std::vector<Car>& cars, bool paused);
00034
00035 #endif // GUI_H
```

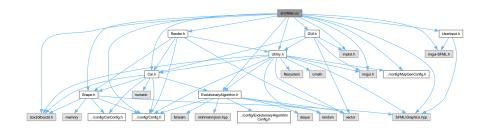
4.21 src/Main.cc File Reference

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 "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 ( )
```

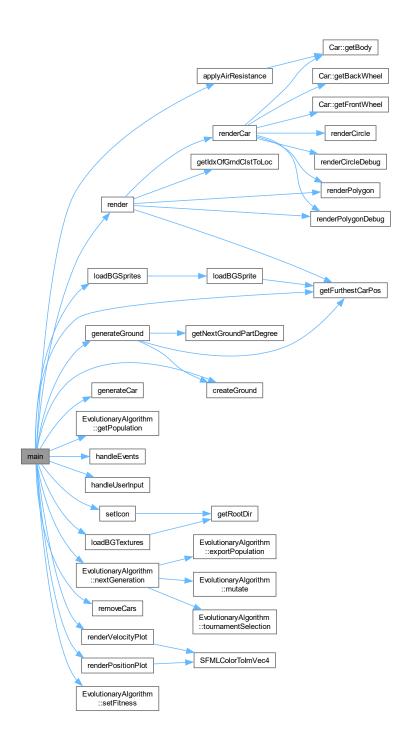
PROGRAM LOOP

```
Definition at line 30 of file Main.cc.
```

```
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
          ImGui::SFML::Init(w);
00040
00041
          ImPlot::CreateContext();
00042
00043
          // Change imgui.ini location
          ImGui::GetIO().IniFilename = "./imgui.ini";
00044
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<br/>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
                                   // Should we pause the simulation?
          bool paused = false;
00064
          bool pauseCheck = true; // Should we check if the user wants to flip `paused`?
00065
          bool nextGen = false;
                                  // Should we generate the next generation?
00066
          bool nextGCheck = true; // Should we check if the user wants to flip `nextGen`?
00067
00068
          bool focus = true;
                                   // Is the window in focus? (used to prevent input when not in focus)
          int timer = 0;
00069
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()) {
              // Update the world, standard arguments
08000
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) {</pre>
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
               ImGui::SFML::Update(w, deltaClock.restart());
00105
00106
00107
               ImGui::PushStyleColor(ImGuiCol_WindowBg, ImVec4(0.071f, 0.141f, 0.137f, 0.5f));
00108
               ImGui::SetNextWindowSize(ImVec2(340, 340), ImGuiCond_FirstUseEver);
00109
               {\tt ImGui::SetNextWindowPos(ImVec2(10, 10), ImGuiCond\_FirstUseEver);}
00110
00111
               renderVelocityPlot(cars, paused);
00112
00113
               ImGui::SetNextWindowSize(ImVec2(340, 340), ImGuiCond_FirstUseEver);
00114
               ImGui::SetNextWindowPos(ImVec2(10, 360), ImGuiCond_FirstUseEver);
00115
               renderPositionPlot(cars, paused);
00116
00117
               ImGui::PopStvleColor();
00118
00119
               generateGround(world, &groundVector, cars);
00120
00121
               ImGui::SFML::Render(w);
00122
00123
               w.display();
00124
               // Attach camera to the car's body
00126
               sf::View cameraView =
00127
                  sf::View(sf::Vector2f(getFurthestCarPos(cars).x * Config::PPM,
                             Config::WINDOW_HEIGHT - getFurthestCarPos(cars).y * Config::PPM), sf::Vector2f(Config::WINDOW_WIDTH, Config::WINDOW_HEIGHT));
00128
00129
00130
               w.setView(cameraView):
00131
               // If the camera moves, shift backgrounds accordingly to create a parallax effect for (int i = 0; i < 5; ++i) {
00132
00133
00134
                   sprites[i].setPosition(
                       cameraView.getCenter().x * (1.0 - 0.2 * i) - Config::WINDOW_WIDTH * (1.4 - 0.1 * i), cameraView.getCenter().y - Config::WINDOW_HEIGHT / 2.0);
00135
00136
00137
               }
00138
00139
               if (!paused) {
00140
                   for (auto& car : cars) {
00141
                       car.getFrontWheel()->body->ApplyTorque(-CarConfig::CAR_TORQUE, false);
                        car.getBackWheel()->body->ApplyTorque(-CarConfig::CAR_TORQUE, false);
00142
00143
                        applyAirResistance(car);
00144
                   }
00145
               }
00146
               \ensuremath{//} Display FPS in window title
00147
               w.setTitle("EvoRacer, FPS: " + std::to_string((int))ImGui::GetIO().Framerate));
00148
00149
00150
               handleEvents(w, pauseCheck, nextGCheck, focus);
00151
              handleUserInput(w, paused, pauseCheck, nextGen, nextGCheck, focus);
00152
          }
00153
          ImPlot::DestroyContext();
00154
00155
          ImGui::SFML::Shutdown();
00156
00157
          return 0;
00158 }
```

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));
00030 int main() {
00031
         sf::ContextSettings settings;
00032
          settings.antialiasingLevel = 8;
00033
          // Setup SFML window
00034
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
          ImGui::SFML::Init(w);
00040
00041
          ImPlot::CreateContext();
00042
00043
          // Change imqui.ini location
          ImGui::GetIO().IniFilename = "./imgui.ini";
00044
00045
00046
          // Containers to hold objects we create
00047
          std::vector<Polygon> groundVector;
00048
          std::vector<Car> cars;
00049
00050
00051
          std::vector<br/>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
00065
                                   // Should we generate the next generation?
00066
          bool nextGen = false;
          bool nextGCheck = true; // Should we check if the user wants to flip `nextGen`?
00068
                                   // Is the window in focus? (used to prevent input when not in focus)
          bool focus = true;
00069
          int timer = 0;
00070
00071
          // Set window icon
00072
         set Icon (w):
00073
00074
          auto textures = loadBGTextures();
00075
          auto sprites = loadBGSprites(textures, cars);
00076
00077
          sf::Clock deltaClock;
00079
          while (w.isOpen()) {
08000
              // Update the world, standard arguments
00081
              if (!paused) {
00082
                  world->Step(1 / 60.0f, 6, 3);
00083
                  ++timer;
00084
                  if (timer >= Config::GENERATION_TIME) {
                      nextGen = true;
00085
00086
                  }
00087
              }
00088
              if (nextGen) {
00089
00090
                  nextGen = false;
```

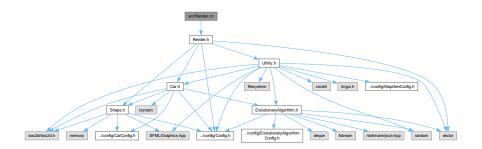
```
for (int i = 0; i < cars.size(); ++i)</pre>
00092
                       ea.setFitness(i, cars[i].getPosX());
00093
00094
                   ea.nextGeneration();
00095
                  removeCars(world, &cars);
for (const Chromosome& chromosome : ea.getPopulation()) {
00096
                       cars.push_back(generateCar(world, chromosome));
00097
00098
                   timer = 0;
00099
00100
              }
00101
00102
              // Render everything
00103
              render(w, sprites, groundVector, cars);
00104
00105
               ImGui::SFML::Update(w, deltaClock.restart());
00106
              ImGui::PushStyleColor(ImGuiCol WindowBg, ImVec4(0.071f, 0.141f, 0.137f, 0.5f));
00107
00108
00109
              ImGui::SetNextWindowSize(ImVec2(340, 340), ImGuiCond_FirstUseEver);
00110
              ImGui::SetNextWindowPos(ImVec2(10, 10), ImGuiCond_FirstUseEver);
00111
              renderVelocityPlot(cars, paused);
00112
              ImGui::SetNextWindowSize(ImVec2(340, 340), ImGuiCond_FirstUseEver);
00113
00114
              ImGui::SetNextWindowPos(ImVec2(10, 360), ImGuiCond_FirstUseEver);
00115
              renderPositionPlot(cars, paused);
00116
00117
              ImGui::PopStyleColor();
00118
00119
              generateGround(world, &groundVector, cars);
00120
00121
              ImGui::SFML::Render(w);
00122
00123
              w.display();
00124
00125
               // Attach camera to the car's body
00126
              sf::View cameraView =
                  sf::View(sf::Vector2f(getFurthestCarPos(cars).x * Config::PPM,
00127
                                          Config::WINDOW_HEIGHT - getFurthestCarPos(cars).y * Config::PPM),
00129
                            sf::Vector2f(Config::WINDOW_WIDTH, Config::WINDOW_HEIGHT));
00130
              w.setView(cameraView);
00131
00132
              // If the camera moves, shift backgrounds accordingly to create a parallax effect
              for (int i = 0; i < 5; ++i) {
00133
00134
                   sprites[i].setPosition(
                       cameraView.getCenter().x * (1.0 - 0.2 * i) - Config::WINDOW_WIDTH * (1.4 - 0.1 * i),
cameraView.getCenter().y - Config::WINDOW_HEIGHT / 2.0);
00135
00136
00137
              }
00138
00139
              if (!paused) {
00140
                   for (auto& car : cars) {
00141
                       car.getFrontWheel()->body->ApplyTorque(-CarConfig::CAR_TORQUE, false);
00142
                       car.getBackWheel()->body->ApplyTorque(-CarConfig::CAR_TORQUE, false);
00143
                       applyAirResistance(car);
00144
                   }
00145
              }
00146
              // Display FPS in window title
00148
              w.setTitle("EvoRacer, FPS: " + std::to_string((int)ImGui::GetIO().Framerate));
00149
00150
              handleEvents(w, pauseCheck, nextGCheck, focus);
00151
              handleUserInput(w, paused, pauseCheck, nextGen, nextGCheck, focus);
00152
          }
00153
00154
          ImPlot::DestroyContext();
00155
          ImGui::SFML::Shutdown();
00156
00157
          return 0;
00158 }
```

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)
- void renderCircleDebug (sf::RenderWindow &w, Circle *circle)
- void renderPolygon (sf::RenderWindow &w, Polygon *polygon)
- void renderPolygonDebug (sf::RenderWindow &w, Polygon *polygon)
- void renderCar (sf::RenderWindow &w, Car car)
- void render (sf::RenderWindow &w, const std::vector < sf::Sprite > &BGs, std::vector < Polygon > &ground ←
 Vector, std::vector < Car > &cars)

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

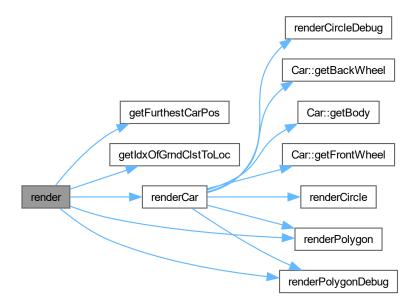
4.23.2.1 render()

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;
          int centerIndex = getIdxOfGrndClstToLoc(groundVector, getFurthestCarPos(cars).x);
00107
00108
          int groundEndIndex = groundVector.size();
00109
         if (centerIndex - Config::GROUND_PARTS_RENDERED / 2 > 0) {
    groundBeginIndex = centerIndex - Config::GROUND_PARTS_RENDERED / 2;
00110
00111
00112
          if (centerIndex + Config::GROUND_PARTS_RENDERED / 2 < groundEndIndex) {</pre>
00113
              groundEndIndex = centerIndex + Config::GROUND_PARTS_RENDERED / 2;
00114
00115
         00116
00117
00118
00119
          for (Polygon ground : groundSlice) {
00120
             renderPolygon(w, &ground);
              if (Config::DEBUG) {
00121
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 \ge 0; --i) {
           renderCar(w, cars[i]);
00128
00129
00130 }
```

Here is the call graph for this function:



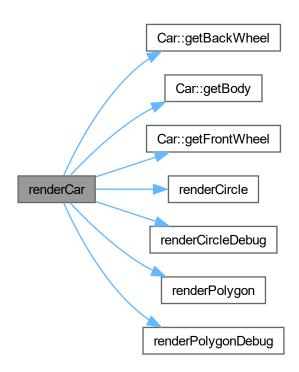
Here is the caller graph for this function:



4.23.2.2 renderCar()

```
void renderCar (
                       sf::RenderWindow & w,
                       Car car )
Definition at line 88 of file Render.cc.
00088
               renderPolygon(w, car.getBody());
renderCircle(w, car.getFrontWheel());
renderCircle(w, car.getBackWheel());
00089
00090
00091
00092
               if (Config::DEBUG) {
                    renderPolygonDebug(w, car.getBody());
renderCircleDebug(w, car.getFrontWheel());
renderCircleDebug(w, car.getBackWheel());
00093
00094
00095
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 )
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()

Here is the caller graph for this function:



4.23.2.5 renderPolygon()

```
void renderPolygon (
    sf::RenderWindow & w,
    Polygon * 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
00060
          // Flip the polygon along the {\tt X} axis
          convex.scale(1, -1);
00061
          convex.setFillColor(polygon->color);
00062
00063
          w.draw(convex);
00064 }
```

Here is the caller graph for this function:



4.24 Render.cc 75

4.23.2.6 renderPolygonDebug()

```
void renderPolygonDebug (
             sf::RenderWindow & w,
             Polygon * polygon )
Definition at line 66 of file Render.cc.
00066
00067
         // Draw the polygon's center
         sf::CircleShape circ;
00068
00069
         circ.setRadius(5);
00070
         circ.setOrigin(5, 5);
         00071
00072
00073
         circ.setFillColor(sf::Color::Blue);
00074
         w.draw(circ);
00075
00076
         // Draw the polygon's vertices
         for (int i = 0; i < polygon->vertices.size(); ++i) {
    circ.setRadius(2);
00077
00078
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.

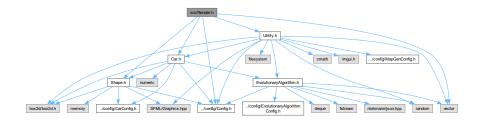
```
00001
00010 #include "Render.h"
00011
00012 void renderCircle(sf::RenderWindow &w, Circle *circle) {
00013
         sf::CircleShape circ;
00014
00015
          circ.setPosition(circle->body->GetPosition().x * Config::PPM,
00016
                           Config::WINDOW_HEIGHT - (circle->body->GetPosition().y * 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(
```

```
circle->body->GetPosition().x * Config::PPM +
00037
                       circle->radius * cos(circle->body->GetAngle()),
                   Config::WINDOW_HEIGHT - (circle->body->GetPosition().y * Config::PPM +
00038
                                              circle->radius * sin(circle->body->GetAngle()))));
00039
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,
                               Config::WINDOW_HEIGHT - (polygon->body->GetPosition().y * Config::PPM));
00047
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,
                                                 polygon->vertices[i].y * Config::PPM));
00054
00055
00056
00057
          convex.setRotation(-1 * polygon->body->GetAngle() * Config::DEG_PER_RAD);
00058
00059
          // Flip the polygon along the \ensuremath{\mathbf{X}} axis
00060
          convex.scale(1, -1);
00061
          convex.setFillColor(polygon->color);
00062
          w.draw(convex);
00063
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
          \verb|circ.setPosition(polygon->body->GetPosition().x * Config::PPM,|\\
                             Config::WINDOW_HEIGHT - (polygon->body->GetPosition().y * Config::PPM));
00072
          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) {
              circ.setRadius(2);
00078
00079
              circ.setOrigin(2, 2);
08000
              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) {
         renderPolygon(w, car.getBody());
renderCircle(w, car.getFrontWheel());
renderCircle(w, car.getBackWheel());
if (Config::DEBUG) {
00089
00090
00091
00092
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) {
          w.clear();
00101
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
          if (centerIndex - Config::GROUND_PARTS_RENDERED / 2 > 0) {
00110
              groundBeginIndex = centerIndex - Config::GROUND_PARTS_RENDERED / 2;
00111
00112
00113
          if (centerIndex + Config::GROUND_PARTS_RENDERED / 2 < groundEndIndex) {</pre>
              groundEndIndex = centerIndex + Config::GROUND_PARTS_RENDERED / 2;
00114
00115
00116
          std::vector<Polygon> groundSlice(groundVector.begin() + groundBeginIndex,
                                              groundVector.begin() + groundEndIndex);
00117
00118
00119
          for (Polygon ground : groundSlice) {
00120
              renderPolygon(w, &ground);
00121
              if (Config::DEBUG) {
00122
                   renderPolygonDebug(w, &ground);
```

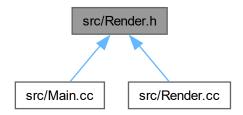
4.25 src/Render.h File Reference

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)
- void renderCircleDebug (sf::RenderWindow &w, Circle *circle)
- void renderPolygon (sf::RenderWindow &w, Polygon *polygon)
- void renderPolygonDebug (sf::RenderWindow &w, Polygon *polygon)
- void renderCar (sf::RenderWindow &w, Car car)
- void render (sf::RenderWindow &w, const std::vector< sf::Sprite > &BGs, std::vector< Polygon > &ground, std::vector< Car > &cars)

4.25.1 Detailed Description

Header file for render function.

Authors

Jakub Marcowski, Mateusz Krakowski

Date

2023-06-06

Definition in file Render.h.

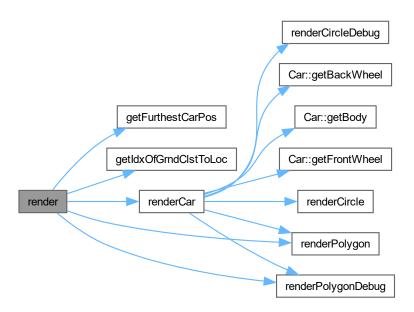
4.25.2 Function Documentation

4.25.2.1 render()

void render (

```
sf::RenderWindow & w,
                 const std::vector< sf::Sprite > & BGs,
                 std::vector< Polygon > & ground,
                 std::vector< Car > & cars )
Definition at line 99 of file Render.cc.
00100
                                                                                          {
00101
            w.clear();
            for (const sf::Sprite &BG : BGs) {
00102
00103
                w.draw(BG);
00104
00105
           int groundBeginIndex = 0;
int centerIndex = getIdxOfGrndClstToLoc(groundVector, getFurthestCarPos(cars).x);
int groundEndIndex = groundVector.size();
00106
00107
00108
00109
00110
            if (centerIndex - Config::GROUND_PARTS_RENDERED / 2 > 0) {
00111
                groundBeginIndex = centerIndex - Config::GROUND_PARTS_RENDERED / 2;
00112
            if (centerIndex + Config::GROUND_PARTS_RENDERED / 2 < groundEndIndex) {
    groundEndIndex = centerIndex + Config::GROUND_PARTS_RENDERED / 2;</pre>
00113
00114
00115
00116
            std::vector<Polygon> groundSlice(groundVector.begin() + groundBeginIndex,
00117
                                                    groundVector.begin() + groundEndIndex);
00118
00119
            for (Polygon ground : groundSlice) {
                renderPolygon(w, &ground);
if (Config::DEBUG) {
00120
00121
00122
                     renderPolygonDebug(w, &ground);
00123
00124
00125
           // new cars should be rendered behind the old ones for (int i = cars.size() - 1; i >= 0; --i) {
00126
00127
00128
               renderCar(w, cars[i]);
00129
00130 }
```

Here is the call graph for this function:



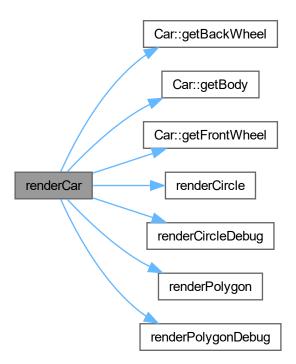
Here is the caller graph for this function:



4.25.2.2 renderCar()

```
void renderCar (
                      sf::RenderWindow & w,
                      Car car )
Definition at line 88 of file Render.cc.
               renderPolygon(w, car.getBody());
renderCircle(w, car.getFrontWheel());
00089
00090
00091
               renderCircle(w, car.getBackWheel());
if (Config::DEBUG) {
00092
                    renderPolygonDebug(w, car.getBody());
renderCircleDebug(w, car.getFrontWheel());
renderCircleDebug(w, car.getBackWheel());
00093
00094
00095
00096
               }
00097 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.25.2.3 renderCircle()

Here is the caller graph for this function:



4.25.2.4 renderCircleDebug()

Definition at line 28 of file Render.cc.

```
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,
                                  Config::WINDOW_HEIGHT - (circle->body->GetPosition().y * Config::PPM))),
00034
00035
               sf::Vertex(sf::Vector2f(
00036
                   circle->body->GetPosition().x * Config::PPM +
                   circle->radius * cos(circle->body->GetAngle()),
Config::WINDOW_HEIGHT - (circle->body->GetPosition().y * Config::PPM +
00037
00038
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()

Definition at line 43 of file Render.cc.

```
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());
           for (int i = 0; i < polygon->vertices.size(); ++i) {
    convex.setPoint(i, sf::Vector2f(polygon->vertices[i].x * Config::PPM,
00052
00053
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 convex.scale(1, -1);
00060
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 )
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
for (int i = 0; i < polygon->vertices.size(); ++i) {
00077
00078
             circ.setRadius(2);
00079
             circ.setOrigin(2, 2);
             08000
00081
00082
00083
             circ.setFillColor(sf::Color::White);
00084
             w.draw(circ);
00085
00086 }
```

4.26 Render.h 83

Here is the caller graph for this function:



4.26 Render.h

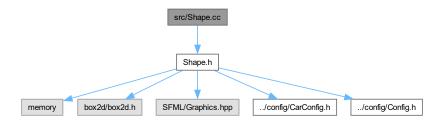
Go to the documentation of this file.

```
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
00019 void renderCircle(sf::RenderWindow &w, Circle *circle);
00020
00021 void renderCircleDebug(sf::RenderWindow &w, Circle *circle);
00022
00023 void renderPolygon(sf::RenderWindow &w, Polygon *polygon);
00024
00025 void renderPolygonDebug(sf::RenderWindow &w, Polygon *polygon);
00026
00027 void renderCar(sf::RenderWindow &w, Car car);
00028
00029 void render(sf::RenderWindow &w, const std::vector<sf::Sprite> &BGs, std::vector<Polygon> &ground,
00030
                  std::vector<Car> &cars);
00031
00032 #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)

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

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 (

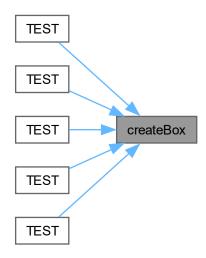
const b2WorldPtr & world,
float x,
float y,
float width,
float height,
float density,
float friction,
sf::Color color )

Definition at line 11 of file Shape.cc.
```

```
00013
          // Argument validation
00014
          if (width <= 0) {</pre>
              throw std::invalid_argument("Invalid width parameter");
00015
00016
          } else if (height <= 0.0f) {
              throw std::invalid_argument("Invalid height parameter");
00017
00018
          } else if (density \leftarrow 0.0f) {
00019
              throw std::invalid_argument("Invalid density parameter");
00020
          } else if (friction <= 0.0f) {</pre>
              throw std::invalid_argument("Invalid friction parameter");
00021
00022
          ,
// Body definition
00023
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;
           fixtureDef.density = density;
fixtureDef.friction = friction;
00034
00035
00036
           fixtureDef.shape = &boxShape;
00037
00038
            // Now we have a body for our Box object
00039
           b2Body* boxBody = world->CreateBody(&boxBodyDef);
           // Lastly, assign the fixture
boxBody->CreateFixture(&fixtureDef);
00040
00041
00042
00043
           return Box{width, height, color, boxBody};
00044 }
```

Here is the caller graph for this function:

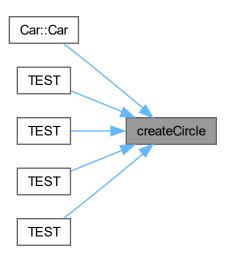


4.27.2.2 createCircle()

```
Circle createCircle (
              const b2WorldPtr & world,
               float x,
               float y,
              float radius,
              float density,
               float friction,
              sf::Color color )
Definition at line 65 of file Shape.cc.
00066
00067
          // Argument validation
          if (radius <= 0.0f) {
00068
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) {</pre>
00073
00074
              throw std::invalid_argument("Invalid friction parameter");
00075
00076
          b2BodyDef boxBodyDef;
          boxBodyDef.position.Set(x / Config::PPM, y / Config::PPM);
```

```
boxBodyDef.type = b2_dynamicBody;
00079
08000
          b2CircleShape circleShape;
          circleShape.m_radius = radius / Config::PPM;
00081
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:

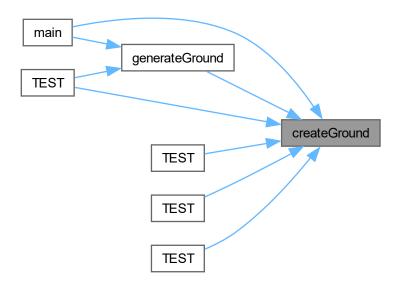


4.27.2.3 createGround()

```
Polygon createGround (
               const b2WorldPtr & world,
               float x,
               float y,
               const std::vector< b2Vec2 > & vertices,
               sf::Color color )
Definition at line 46 of file Shape.cc.
00047
00048
           // Argument validation
          if (vertices.size() < 3) {</pre>
00049
               throw std::invalid_argument("Invalid number of vertices");
00050
00051
00052
00053
          b2BodyDef groundBodyDef;
00054
          groundBodyDef.position.Set(x / Config::PPM, y / Config::PPM);
00055
00056
          b2PolygonShape groundPolygon;
groundPolygon.Set(vertices.data(), static_cast<int32>(vertices.size()));
00057
00058
00059
          b2Body* groundBody = world->CreateBody(&groundBodyDef);
```

```
00060 groundBody->CreateFixture(&groundPolygon, 0.0f);
00061
00062 return Polygon{vertices, color, groundBody};
00063 }
```

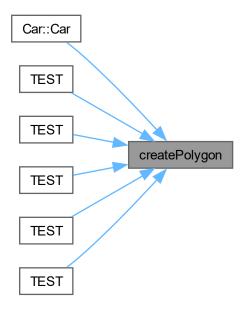
Here is the caller graph for this function:



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 )
Definition at line 95 of file Shape.cc.
00097
            // Argument validation
           if (vertices.size() < 3 || vertices.size() > CarConfig::CAR_VERTICES) {
00098
           throw std::invalid_argument("Invalid vertices size");
} else if (density <= 0.0f) {</pre>
00099
00100
           throw std::invalid_argument("Invalid density parameter");
} else if (friction <= 0.0f) {
00101
00102
00103
                throw std::invalid_argument("Invalid friction parameter");
00104
           b2BodyDef boxBodyDef;
00105
           boxBodyDef.position.Set(x / Config::PPM, y / Config::PPM);
boxBodyDef.type = b2_dynamicBody;
00106
00107
00108
00109
           b2PolygonShape boxShape;
00110
           boxShape.Set(vertices.data(), vertices.size());
00111
           b2FixtureDef fixtureDef;
fixtureDef.density = density;
fixtureDef.friction = friction;
00112
00113
00114
00115
           fixtureDef.shape = &boxShape;
```

Here is the caller graph for this function:



4.28 Shape.cc

Go to the documentation of this file.

```
00009 #include "Shape.h"
00010
00011 Box createBox(const b2WorldPtr& world, float x, float y, float width, float height, float density,
                     float friction, sf::Color color) {
00012
00013
           // Argument validation
          if (width <= 0) {
00014
00015
               throw std::invalid_argument("Invalid width parameter");
00016
          } else if (height <= 0.0f) {</pre>
               throw std::invalid_argument("Invalid height parameter");
00017
          } else if (density <= 0.0f) {</pre>
00018
          throw std::invalid_argument("Invalid density parameter");
} else if (friction <= 0.0f) {</pre>
00019
00020
               throw std::invalid_argument("Invalid friction parameter");
00021
00022
           // Body definition
00023
           b2BodyDef boxBodyDef;
00024
          boxBodyDef.position.Set(x / Config::PPM, y / Config::PPM);
boxBodyDef.type = b2_dynamicBody;
00025
00026
00027
00028
           // Shape definition
00029
           b2PolygonShape boxShape;
00030
00031
           boxShape.SetAsBox(width / 2 / Config::PPM, height / 2 / Config::PPM);
00032
           // Fixture definition
00033
           b2FixtureDef fixtureDef;
00034
           fixtureDef.density = density;
```

4.28 Shape.cc 89

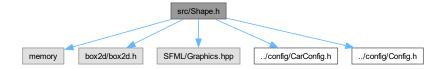
```
fixtureDef.friction = friction;
                 fixtureDef.shape = &boxShape;
00036
00037
00038
                  // Now we have a body for our Box object
                 b2Body* boxBody = world->CreateBody(&boxBodyDef);
00039
                  // Lastly, assign the fixture
00040
                 boxBody->CreateFixture(&fixtureDef);
00041
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, \ float \ y, \ const \ std::vector < b2Vec2 > \& \ vertices, \ float \ y, 
00047
                                               sf::Color color) {
                  // Argument validation
00048
00049
                 if (vertices.size() < 3) {</pre>
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;
                 groundPolygon.Set(vertices.data(), static_cast<int32>(vertices.size()));
00057
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, float friction, sf::Color color) {
00067
                 // Argument validation
00068
                 if (radius <= 0.0f) {</pre>
00069
                        throw std::invalid_argument("Invalid width parameter");
00070
                 } else if (density <= 0.0f) {
                throw std::invalid_argument("Invalid density parameter");
} else if (friction <= 0.0f) {</pre>
00071
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;
                 fixtureDef.friction = friction;
00085
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) {</pre>
00101
                 throw std::invalid_argument("Invalid density parameter");
} else if (friction <= 0.0f) {</pre>
00102
00103
                       throw std::invalid_argument("Invalid friction parameter");
00104
00105
                 b2BodyDef boxBodyDef;
                 boxBodyDef.position.Set(x / Config::PPM, y / Config::PPM);
00106
00107
                 boxBodyDef.type = b2_dynamicBody;
00108
00109
                 b2PolygonShape boxShape;
00110
                 boxShape.Set(vertices.data(), vertices.size());
00111
00112
                 b2FixtureDef fixtureDef;
                 fixtureDef.density = density;
fixtureDef.friction = friction;
00113
00114
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 }
```

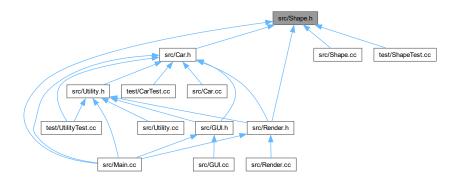
4.29 src/Shape.h File Reference

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 Circle
- struct Polygon

Typedefs

 $\bullet \ \ typedef \ std::shared_ptr < b2World > b2WorldPtr \\$

Functions

- Box createBox (const b2WorldPtr &world, float x, float y, float width, float height, float density, float friction, sf::Color color)
- Polygon createGround (const b2WorldPtr &world, float x, float y, const std::vector< b2Vec2 > &vertices, sf::Color color)
- Circle createCircle (const b2WorldPtr &world, float x, float y, float radius, float density, float friction, sf::Color color)
- Polygon createPolygon (const b2WorldPtr &world, float x, float y, std::vector < b2Vec2 > vertices, float density, float friction, sf::Color color)

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 39 of file Shape.h.

4.29.3 Function Documentation

4.29.3.1 createBox()

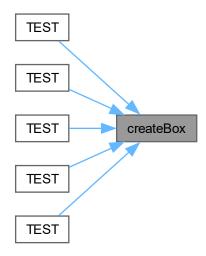
```
Box createBox (

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

Definition at line 11 of file Shape.cc.
00012
00013 // Argument validation
00014 if (width <= 0) {
```

```
throw std::invalid_argument("Invalid width parameter");
00016
           } else if (height <= 0.0f) {
                throw std::invalid_argument("Invalid height parameter");
00017
           } else if (density <= 0.0f) {</pre>
00018
           throw std::invalid_argument("Invalid density parameter");
} else if (friction <= 0.0f) {</pre>
00019
00020
00021
               throw std::invalid_argument("Invalid friction parameter");
00022
00023
            // Body definition
           b2BodyDef boxBodyDef;
00024
           boxBodyDef boxBodyDef;
boxBodyDef.position.Set(x / Config::PPM, y / Config::PPM);
boxBodyDef.type = b2_dynamicBody;
00025
00026
00027
00028
            // Shape definition
00029
           b2PolygonShape boxShape;
           boxShape.SetAsBox(width / 2 / Config::PPM, height / 2 / Config::PPM);
00030
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
b2Body* boxBody = world->CreateBody(&boxBodyDef);
00039
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:

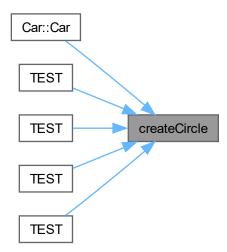


4.29.3.2 createCircle()

Definition at line 65 of file Shape.cc.

```
00067
           // Argument validation
00068
           if (radius <= 0.0f) {
          throw std::invalid_argument("Invalid width parameter");
} else if (density <= 0.0f) {</pre>
00069
00070
          throw std::invalid_argument("Invalid density parameter");
} else if (friction <= 0.0f) {</pre>
00071
00072
00073
              throw std::invalid_argument("Invalid friction parameter");
00074
00075
00076
          b2BodyDef boxBodyDef;
           boxBodyDef.position.Set(x / Config::PPM, y / Config::PPM);
00077
00078
           boxBodyDef.type = b2_dynamicBody;
00079
08000
           b2CircleShape circleShape;
           circleShape.m_radius = radius / Config::PPM;
00081
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:

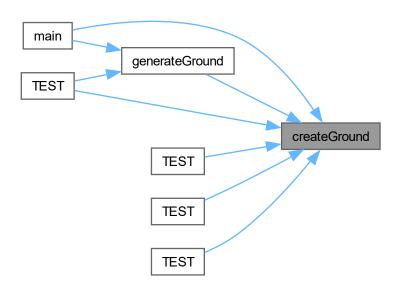


4.29.3.3 createGround()

Definition at line 46 of file Shape.cc.

```
00048
          // Argument validation
00049
          if (vertices.size() < 3) {</pre>
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:



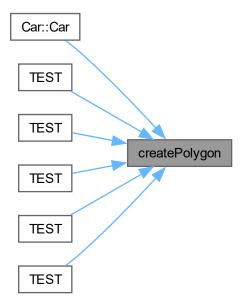
4.29.3.4 createPolygon()

```
Polygon createPolygon (
                  const b2WorldPtr & world,
                  float x,
                  float y,
                  std::vector< b2Vec2 > vertices,
                  float density,
                  float friction,
                  sf::Color color )
Definition at line 95 of file Shape.cc.
00096
00097
             // Argument validation
            if (vertices.size() < 3 || vertices.size() > CarConfig::CAR_VERTICES) {
   throw std::invalid_argument("Invalid vertices size");
} else if (density <= 0.0f) {</pre>
00098
00099
00100
00101
                throw std::invalid_argument("Invalid density parameter");
00102
            } else if (friction <= 0.0f) {</pre>
```

4.30 Shape.h 95

```
throw std::invalid_argument("Invalid friction parameter");
00104
00105
          b2BodyDef boxBodyDef;
          boxBodyDef.position.Set(x / Config::PPM, y / Config::PPM);
00106
00107
          boxBodyDef.type = b2_dynamicBody;
00108
00109
          b2PolygonShape boxShape;
00110
          boxShape.Set(vertices.data(), vertices.size());
00111
          b2FixtureDef fixtureDef;
00112
          fixtureDef.density = density;
fixtureDef.friction = friction;
00113
00114
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.

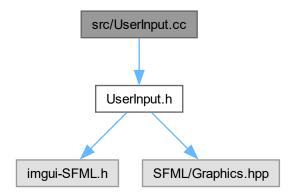
```
00001
00009 #ifndef SHAPE_H
00010 #define SHAPE_H
00011
00012 #include <memory>
00013
00014 #include "box2d/box2d.h"
00015 #include "SFML/Graphics.hpp"
00016
00017 #include "../config/CarConfig.h"
00018 #include "../config/Config.h"
00019
00020 struct Box {
00021 float width{};
```

```
float height{};
 00023
                                 sf::Color color;
 00024
                                b2Body* body{};
00025 };
00026
00027 struct Circle {
 00028
                               float radius{};
 00029
                                 sf::Color color;
 00030
                               b2Body* body{};
 00031 };
00032
 00033 struct Polygon {
 00034
                               std::vector<b2Vec2> vertices;
 00035
                                 sf::Color color;
 00036
                                b2Body* body;
 00037 };
 00038
 00039 typedef std::shared_ptr<b2World> b2WorldPtr;
 00041 Box createBox(const b2WorldPtr& world, float x, float y, float width, float height, float density,
 00042
                                                                 float friction, sf::Color color);
 00043
 00044 \ \ Polygon \ \ createGround (const \ b2WorldPtr\& \ world, \ float \ x, \ float \ y, \ const \ std::vector < b2Vec2>\& \ vertices, \ float \ y, \ float \ 
00045
                                                                                         sf::Color color);
 00046
 00047 Circle createCircle(const b2WorldPtr& world, float x, float y, float radius, float density,
 00048
                                                                                     float friction, sf::Color color);
00049
00050 Polygon createPolygon(const b2WorldPtr& world, float x, float y, std::vector<b2Vec2> vertices, 00051 float density, float friction, sf::Color color);
00052
00053 #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)
- void handleEvents (sf::RenderWindow &w, bool &pause_check, bool &nxt_g_check, bool &focus)

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 )
```

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
               if (event.type == sf::Event::LostFocus) {
00045
00046
                   focus = false;
00047
00048
               if (focus) {
00049
                   ImGui::SFML::ProcessEvent(event);
00050
                   // Close window : exit
                   if (event.type == sf::Event::Closed) {
   w.close();
00051
00052
00053
00054
                   if (event.type == sf::Event::KeyReleased) {
                        // Allow user to toggle pause again
if (event.key.code == sf::Keyboard::P || event.key.code == sf::Keyboard::Space) {
00055
00056
00057
                            pause_check = true;
00058
                        // 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 }
```

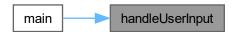
Here is the caller graph for this function:



4.31.2.2 handleUserInput()

```
void handleUserInput (
              sf::RenderWindow & w,
              bool & paused,
              bool & pause_check,
              bool & next_gen,
              bool & nxt_g_check,
              bool & focus )
Definition at line 11 of file UserInput.cc.
00013
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.

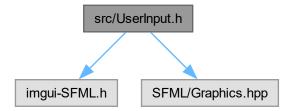
```
00001
00009 #include "UserInput.h"
00010
00011 void handleUserInput(sf::RenderWindow &w, bool &paused, bool &pause_check, bool &next_gen,
00012
                           bool &nxt_g_check, bool &focus) {
00013
          if (focus) {
              if (sf::Keyboard::isKeyPressed(sf::Keyboard::Q) ||
00014
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
               if (sf::Keyboard::isKeyPressed(sf::Keyboard::N)) {
00028
00029
                   // Generate the next generation
00030
                   if (nxt_g_check) {
                       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) {
    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
                   if (event.type == sf::Event::KeyReleased) {
00054
                       // Allow user to toggle pause again
if (event.key.code == sf::Keyboard::P || event.key.code == sf::Keyboard::Space) {
00055
00056
00057
                            pause_check = true;
00058
00059
                       \ensuremath{//} Allow user to generate the next generation again
                       if (event.key.code == sf::Keyboard::N) {
00060
                            nxt_g_check = true;
00061
00062
00063
                   }
00064
             }
00065
00066 }
```

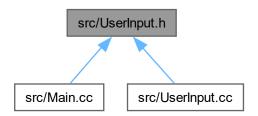
4.33 src/UserInput.h File Reference

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)
- void handleEvents (sf::RenderWindow &w, bool &pause check, bool &nxt g check, bool &focus)

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 )
```

Definition at line 38 of file UserInput.cc.

```
00043
                     focus = true;
00044
00045
                 if (event.type == sf::Event::LostFocus) {
00046
                     focus = false;
00047
00048
                 if (focus) {
                     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) {
                           // Allow user to toggle pause again
if (event.key.code == sf::Keyboard::P || event.key.code == sf::Keyboard::Space) {
00055
00056
00057
                               pause_check = true;
00058
                          // Allow user to generate the next generation again
if (event.key.code == sf::Keyboard::N) {
    nxt_g_check = true;
00059
00060
00061
00062
00063
00064
                }
00065
            }
00066 }
```



4.33.2.2 handleUserInput()

void handleUserInput (

```
sf::RenderWindow & w,
               bool & paused,
               bool & pause_check,
               bool & next_gen,
               bool & nxt_g_check,
               bool & focus )
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;
                      pause_check = false;
00025
00026
00027
00028
              if (sf::Keyboard::isKeyPressed(sf::Keyboard::N)) {
00029
                  // Generate the next generation
00030
                  if (nxt_g_check) {
00031
                      next_gen = true;
                      nxt_g_check = false;
00032
00033
00034
              }
```

```
00035 }
00036 }
```

Here is the caller graph for this function:



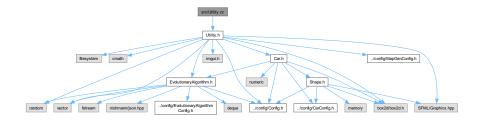
4.34 UserInput.h

Go to the documentation of this file.

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)

- float getNextGroundPartDegree ()
- Car generateCar (const b2WorldPtr &world, const Chromosome &chromosome)
- ImVec4 SFMLColorTolmVec4 (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 ()
- void setIcon (sf::RenderWindow &window)
- std::vector< sf::Texture * > loadBGTextures ()
- sf::Sprite loadBGSprite (sf::Texture *texture, const std::vector< Car > &cars)
- std::vector< sf::Sprite > loadBGSprites (std::vector< sf::Texture * > textures, const std::vector< Car > &cars)

4.35.1 Detailed Description

File containing utility functions.

Authors

Jakub Marcowski, Mateusz Krakowski

Date

2023-06-06

Definition in file Utility.cc.

4.35.2 Function Documentation

4.35.2.1 applyAirResistance()

Simplified air drag.

Parameters

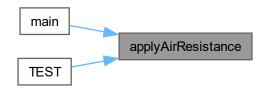
```
car | Car to apply air resistance to.
```

Definition at line 11 of file Utility.cc.

Here is the call graph for this function:

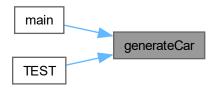


Here is the caller graph for this function:



4.35.2.2 generateCar()

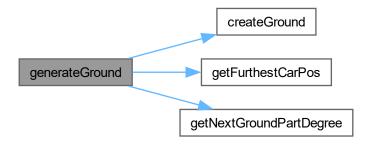
```
Car generateCar (
               const b2WorldPtr & world,
               const Chromosome & chromosome )
Definition at line 66 of file Utility.cc.
00066
00067
          std::random_device rd;
00068
          std::mt19937 gen(rd());
00069
          std::uniform_int_distribution<> rgb_value(50, 200);
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));
00073
00074
00075
          return {world,
                  MapGenConfig::CAR_STARTING_X,
MapGenConfig::CAR_STARTING_Y,
00076
00077
                  chromosome,
00078
                  bodyColor,
00079
                   wheelColor};
00080 }
```



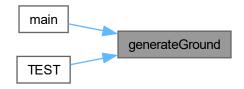
4.35.2.3 generateGround()

```
void generateGround (
                 const b2WorldPtr & world,
                 std::vector< Polygon > * groundVector,
                 const std::vector< Car > & cars )
Definition at line 27 of file Utility.cc.
00028
00029
            Polygon lastGround = groundVector->back();
00030
            if (lastGround.vertices[1].x * Config::PPM <</pre>
00031
                getFurthestCarPos(cars).x * Config::PPM + MapGenConfig::GENERATE_DISTANCE) {
00032
                float degree = getNextGroundPartDegree();
00033
                float angle_in_radians = degree * (M_PI / 180.0f);
00034
                float delta_x = MapGenConfig::GROUND_PART_LENGTH * cos(angle_in_radians);
float delta_y = -MapGenConfig::GROUND_PART_LENGTH * sin(angle_in_radians);
00035
00036
00037
00038
                std::vector<b2Vec2> groundVertices = {
00039
                     b2Vec2(lastGround.vertices[1].x, lastGround.vertices[1].y),
                     b2Vec2(lastGround.vertices[1].x + delta_x, lastGround.vertices[1].y + delta_y), b2Vec2(lastGround.vertices[2].x + delta_x, lastGround.vertices[2].y + delta_y)};
00040
00041
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()

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

Parameters

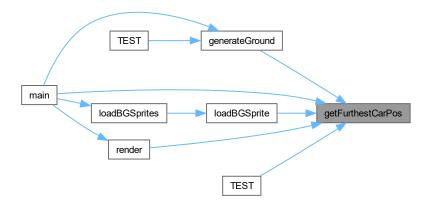
```
cars Vector of cars.
```

Returns

b2Vec2 Position of the furthest car.

Definition at line 86 of file Utility.cc.

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



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

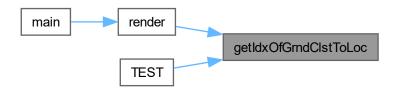
```
cars
       Vector of cars.
```

Returns

int Index of the ground element.

```
Definition at line 100 of file Utility.cc. _{\tt 00100}
00101
            int index = 0;
for (int i = 0; i < ground.size(); ++i) {</pre>
00102
00103
                 if (ground[i].vertices[0].x - x > 0) {
00104
00105
00106
                 index = i;
00107
00108
            return index;
00109 }
```

Here is the caller graph for this function:

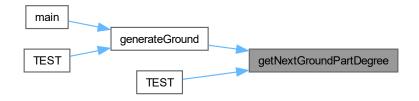


4.35.2.6 getNextGroundPartDegree()

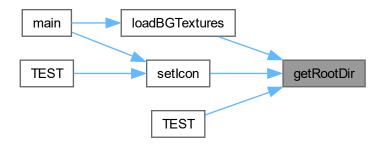
```
float getNextGroundPartDegree ( )
```

```
Definition at line 51 of file Utility.cc.
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;
          } else if (degree < -MapGenConfig::MAX_GROUND_DEGREE) {</pre>
00059
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()



4.35.2.8 loadBGSprite()

Here is the call graph for this function:



Here is the caller graph for this function:



4.35.2.9 loadBGSprites()

Here is the call graph for this function:



Here is the caller graph for this function:



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

texture->loadFromFile(BGPath);

4.35.2.10 loadBGTextures()

```
Definition at line 134 of file Utility.cc.
00135
            std::vector<sf::Texture*> textures;
00136
            for (int i = 0; i < MapGenConfig::BG_SPRITES_COUNT; ++i) {</pre>
                std::string BGPath =
    (getRootDir() / ("../resources/background_img_" + std::to_string(i) + ".png")).string();
auto* texture = new sf::Texture();
00137
00138
```

00141 texture->setRepeated(true); 00142 textures.push_back(texture); 00143 return textures; 00144 00145 }

00139 00140

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 )
```

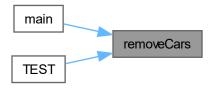
Deletes all cars from the world and the Car vector.

Parameters

| world | d | A shared pointer to the box2d world. |
|-------|---|--------------------------------------|
| cars | | A vector of cars. |

Definition at line 111 of file Utility.cc.

Here is the caller graph for this function:

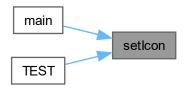


4.35.2.12 setIcon()

Definition at line 126 of file Utility.cc.

Here is the call graph for this function:





4.35.2.13 SFMLColorTolmVec4()

Transforms a SFML color into an ImGUI color.

Parameters

```
color SFML color.
```

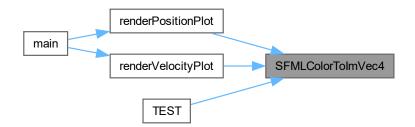
Returns

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.

```
00009 #include "Utility.h"
00010
00011 void applyAirResistance(Car car) {
          // F = V^2 * k
// k 1/2 * * A * C_d 3.4
00012
00013
           // = 1.293 \text{ kg/m}^3
00014
          // A = ? (let's assume 5 m^2)
// C_d = ? (let's assume 1.05)
00015
00016
00018
00019
           car.getBody()->body->ApplyForceToCenter(
               00020
00021
00022
                       -1.84 * car.getBody()->body->GetLinearVelocity().y *
                           abs(car.getBody()->body->GetLinearVelocity().y)),
00023
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 \star (M_PI / 180.0f);
00034
00035
               float delta_x = MapGenConfig::GROUND_PART_LENGTH * cos(angle_in_radians);
               float delta_y = -MapGenConfig::GROUND_PART_LENGTH * sin(angle_in_radians);
00036
00037
00038
               std::vector<b2Vec2> groundVertices =
                   b2Vec2(lastGround.vertices[1].x, lastGround.vertices[1].y),
b2Vec2(lastGround.vertices[1].x + delta_x, lastGround.vertices[1].y + delta_y),
b2Vec2(lastGround.vertices[2].x + delta_x, lastGround.vertices[2].y + delta_y));
00039
00040
00041
00042
00043
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
          }
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);
           if (degree > MapGenConfig::MAX_GROUND_DEGREE) {
   degree = MapGenConfig::MAX_GROUND_DEGREE;
00057
00058
           } else if (degree < -MapGenConfig::MAX_GROUND_DEGREE) {</pre>
00059
00060
               degree = -MapGenConfig::MAX GROUND DEGREE;
00061
           }
00062
00063
           return degree;
00064 }
00065
00066 Car generateCar(const b2WorldPtr& world, const Chromosome& chromosome) {
          std::random_device rd;
00068
           std::mt19937 gen(rd());
00069
           std::uniform_int_distribution<> rgb_value(50, 200);
00070
          sf::Color bodyColor = sf::Color(rgb_value(gen), rgb_value(gen), rgb_value(gen));
sf::Color wheelColor = sf::Color(rgb_value(gen), rgb_value(gen), rgb_value(gen));
00071
00072
00073
00074
00075
                    MapGenConfig::CAR_STARTING_X,
00076
                    MapGenConfig::CAR_STARTING_Y,
00077
                    chromosome,
00078
                    bodvColor,
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) {
```

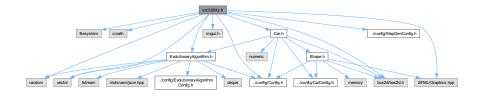
```
float currentCarX = car.getBody()->body->GetPosition().x;
00091
             float currentCarY = car.getBody()->body->GetPosition().y;
              if (currentCarX > furthestCarX) {
00092
                 furthestCarX = currentCarX;
furthestCarY = currentCarY;
00093
00094
00095
             }
00097
          return {furthestCarX, furthestCarY};
00098 }
00099
00100 int getIdxOfGrndClstToLoc(std::vector<Polygon> ground, float x) {
00101
         int index = 0;
          for (int i = 0; i < ground.size(); ++i) {</pre>
00102
00103
             if (ground[i].vertices[0].x - x > 0) {
00104
00105
             index = i:
00106
00107
00108
          return index;
00109 }
00110
00111 void removeCars(const b2WorldPtr& world, std::vector<Car>* cars) {
00112
       for (auto car : *cars) {
             world->DestroyBody(car.getBody()->body);
00113
             world->DestroyBody(car.getBackWheel()->body);
00114
00115
             world->DestroyBody(car.getFrontWheel()->body);
00116
00117
         cars->clear();
00118 }
00119
00120 std::filesystem::path getRootDir() {
         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) {</pre>
00137
             std::string BGPath =
                 (getRootDir() / ("../resources/background_img_" + std::to_string(i) + ".png")).string();
00138
             auto* texture = new sf::Texture();
00139
00140
             texture->loadFromFile(BGPath);
00141
             texture->setRepeated(true);
00142
             textures.push_back(texture);
00143
00144
         return textures;
00145 }
00147 sf::Sprite loadBGSprite(sf::Texture* texture, const std::vector<Car>& cars) {
00148
         sf::Sprite sprite(*texture);
         00149
00150
         sprite.setTextureRect(sf::IntRect(0, 0, 256 * Config::WINDOW_WIDTH, Config::WINDOW_HEIGHT));
00151
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,
                                            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) {</pre>
00163
             sprites.push_back(loadBGSprite(textures[i], cars));
00164
00165
         return sprites;
00166 }
```

4.37 src/Utility.h File Reference

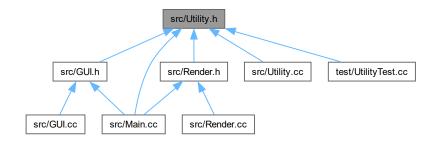
Header for a file containing utility functions.

```
#include <filesystem>
#include <cmath>
#include <random>
#include <vector>
#include "box2d/box2d.h"
#include "imgui.h"
#include "SFML/Graphics.hpp"
#include "EvolutionaryAlgorithm.h"
#include "../config/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 > *boxes, const std::vector< Car > &cars)
- float getNextGroundPartDegree ()
- Car generateCar (const b2WorldPtr &world, const Chromosome &chromosome)
- ImVec4 SFMLColorTolmVec4 (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 ()
- void setlcon (sf::RenderWindow &window)
- std::vector< sf::Texture * > loadBGTextures ()
- sf::Sprite loadBGSprite (sf::Texture *texture, const std::vector< Car > &cars)
- std::vector< sf::Sprite > loadBGSprites (std::vector< sf::Texture * > textures, const std::vector< Car > &cars)

4.37.1 Detailed Description

Header for a file containing utility functions.

Authors

Jakub Marcowski, Mateusz Krakowski

Date

2023-06-06

Definition in file Utility.h.

4.37.2 Typedef Documentation

4.37.2.1 b2WorldPtr

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

Definition at line 26 of file Utility.h.

4.37.3 Function Documentation

4.37.3.1 applyAirResistance()

Simplified air drag.

Parameters

car | Car to apply air resistance to.

Definition at line 11 of file Utility.cc.

```
// F = V^2 * k

// k 1/2 * * A * C_d 3.4

// = 1.293 kg/m^3

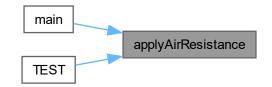
// A = ? (let's assume 5 m^2)

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

Here is the call graph for this function:

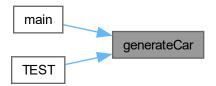


Here is the caller graph for this function:



4.37.3.2 generateCar()

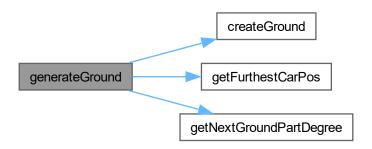
```
Car generateCar (
               const b2WorldPtr & world,
               const Chromosome & chromosome )
Definition at line 66 of file Utility.cc.
00066
00067
          std::random_device rd;
00068
          std::mt19937 gen(rd());
00069
          std::uniform_int_distribution<> rgb_value(50, 200);
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,
```



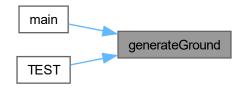
4.37.3.3 generateGround()

```
void generateGround (
                const b2WorldPtr & world,
                std::vector< Polygon > * boxes,
                const std::vector< Car > & cars )
Definition at line 27 of file Utility.cc.
00028
           Polygon lastGround = groundVector->back();
00029
           if (lastGround.vertices[1].x * Config::PPM <
   getFurthestCarPos(cars).x * Config::PPM + MapGenConfig::GENERATE_DISTANCE) {</pre>
00030
00031
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),
                    b2Vec2(lastGround.vertices[1].x + delta_x, lastGround.vertices[1].y + delta_y),
b2Vec2(lastGround.vertices[2].x + delta_x, lastGround.vertices[2].y + delta_y)};
00040
00041
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 ( {\tt const\ std::vector} < {\tt Car} \ > \ \& \ {\it cars} \ )
```

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

Parameters

```
cars Vector of cars.
```

Returns

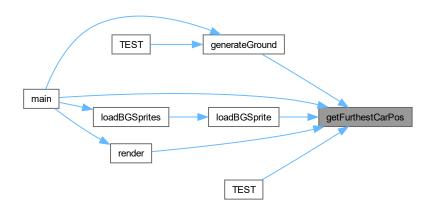
b2Vec2 Position of the furthest car.

Definition at line 86 of file Utility.cc.

```
00086

00087 float furthestCarX = 0;

00088 float furthestCarY = 0;
```



4.37.3.5 getIdxOfGrndClstToLoc()

"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

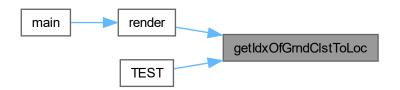
int Index of the ground element.

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:



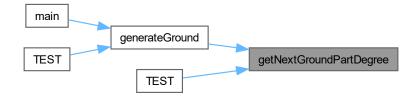
4.37.3.6 getNextGroundPartDegree()

```
float getNextGroundPartDegree ( )
```

Definition at line 51 of file Utility.cc.

```
00052
           std::random_device rd;
           std::mt19937 gen(rd());
00053
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) {
                degree = MapGenConfig::MAX_GROUND_DEGREE;
00058
           } else if (degree < -MapGenConfig::MAX_GROUND_DEGREE) {
   degree = -MapGenConfig::MAX_GROUND_DEGREE;</pre>
00059
00060
00061
00062
00063
           return degree;
00064 }
```

Here is the caller graph for this function:



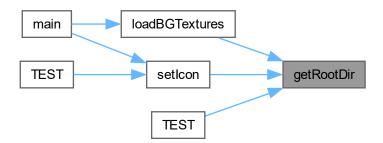
4.37.3.7 getRootDir()

00123

00124 }

Here is the caller graph for this function:

return dirPath;



4.37.3.8 loadBGSprite()

```
sf::Sprite loadBGSprite (
              sf::Texture * texture,
              const std::vector< Car > & cars )
Definition at line 147 of file Utility.cc.
00147
00148
          sf::Sprite sprite(*texture);
         00149
00150
00151
          sprite.setTextureRect(sf::IntRect(0, 0, 256 * Config::WINDOW_WIDTH, Config::WINDOW_HEIGHT));
00152
         sprite.setPosition(
             sf::Vector2f(getFurthestCarPos(cars).x * Config::PPM, 0.5 * Config::WINDOW_HEIGHT) - sf::Vector2f(Config::WINDOW_WIDTH / 2.0f, Config::WINDOW_HEIGHT / 2.0f));
00153
00154
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()

Here is the call graph for this function:



Here is the caller graph for this function:



4.37.3.10 loadBGTextures()

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

Definition at line 134 of file Utility.cc.

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

Here is the call graph for this function:



Here is the caller graph for this function:



4.37.3.11 removeCars()

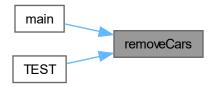
Deletes all cars from the world and the Car vector.

Parameters

| world | d | A shared pointer to the box2d world. |
|-------|---|--------------------------------------|
| cars | | A vector of cars. |

Definition at line 111 of file Utility.cc.

Here is the caller graph for this function:



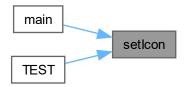
4.37.3.12 setIcon()

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

Definition at line 126 of file Utility.cc.

Here is the call graph for this function:





4.37.3.13 SFMLColorTolmVec4()

Transforms a SFML color into an ImGUI color.

Parameters

```
color SFML color.
```

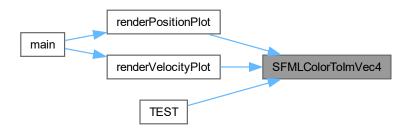
Returns

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/Config.h"
00023 #include "../config/MapGenConfig.h"
00024 #include "Car.h"
00025
00026 typedef std::shared_ptr<b2World> b2WorldPtr;
00027
00028 // TODO: think if some of these functions should be moved to other files
00029 // and whether or not some constants should be moved to config
00030
00035 void applyAirResistance(Car car);
00036
00037 void generateGround(const b2WorldPtr& world, std::vector<Polygon>* boxes,
                            const std::vector<Car>& cars);
00039
00040 float getNextGroundPartDegree();
00041
00042 Car generateCar(const b2WorldPtr& world, const Chromosome& chromosome);
00043
00050 ImVec4 SFMLColorToImVec4(sf::Color color);
00059 b2Vec2 getFurthestCarPos(const std::vector<Car>& cars);
00060
00069 int getIdxOfGrndClstToLoc(std::vector<Polygon> ground, float x);
00070
00077 void removeCars(const b2WorldPtr& world, std::vector<Car>* cars);
00078
00079 std::filesystem::path getRootDir();
08000
00081 void setIcon(sf::RenderWindow& window);
00082
00083 std::vector<sf::Texture*> loadBGTextures();
00085 sf::Sprite loadBGSprite(sf::Texture* texture, const std::vector<Car>& cars);
00086
00087 std::vector<sf::Sprite> loadBGSprites(std::vector<sf::Texture*> textures,
00088
                                                const std::vector<Car>& cars);
00089
00090 #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 (

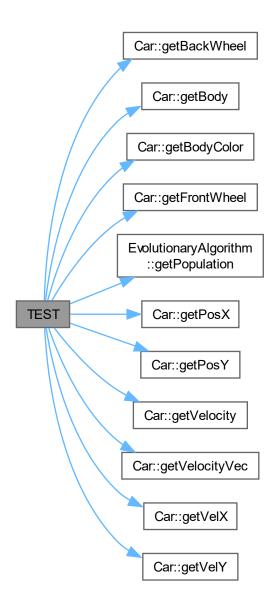
CreateCarTest ,

BasicTest )
```

```
Definition at line 13 of file CarTest.cc.
```

```
00014
            b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
           float x = 0.0f, y = 0.0f;
sf::Color bodyColor = sf::Color::Red;
sf::Color wheelColor = sf::Color::Blue;
00015
00016
00017
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
           EXPECT_EQ(car.getPosX(), x);
EXPECT_EQ(car.getPosY(), y);
EXPECT_EQ(car.getBodyColor(), bodyColor);
00023
00024
00025
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());
EXPECT_NO_THROW(car.getVelocityVec());
00031
            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.

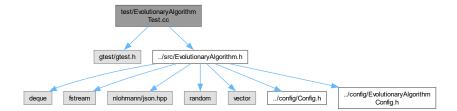
```
00001
00009 #include <gtest/gtest.h>
00010
00011 #include "../src/Car.h"
00012
00013 TEST(CreateCarTest, BasicTest) {
00014 b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
00015 float x = 0.0f, y = 0.0f;
00016 sf::Color bodyColor = sf::Color::Red;
00017 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);
EXPECT_EQ(car.getBodyColor(), bodyColor);
00025
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]

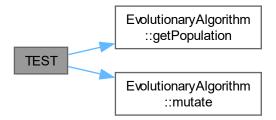
```
TEST (
                    EvolutionaryAlgorithmTest ,
                    MutationTest )
Definition at line 12 of file EvolutionaryAlgorithmTest.cc.
00013
              EvolutionaryAlgorithm evo(10);
              // mutate the population 50 times just to test it for (int i = 0; i < 50; ++i) {
00014
00015
                   evo.mutate();
00016
00017
00018
              std::vector<Chromosome> population = evo.getPopulation();
00019
              // assert that all body lengths are within the range
for (auto& chrom : population) {
    for (auto& length : chrom.bodyLengths) {
00020
00021
00022
                         ASSERT_GE (length, EvolutionaryAlgorithmConfig::MIN_BODY_LENGTH);
ASSERT_LE (length, EvolutionaryAlgorithmConfig::MAX_BODY_LENGTH);
00023
00024
00025
```

Here is the call graph for this function:

00026

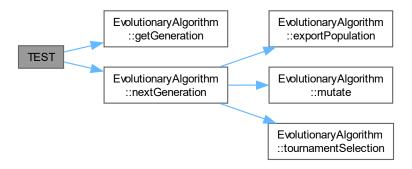
00027 }

}



4.41.2.2 TEST() [2/4]

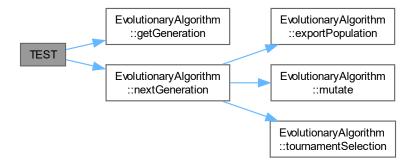
Definition at line 37 of file EvolutionaryAlgorithmTest.cc.



4.41.2.3 TEST() [3/4]

Definition at line 47 of file EvolutionaryAlgorithmTest.cc.

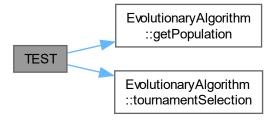
Here is the call graph for this function:



4.41.2.4 TEST() [4/4]

Definition at line 29 of file EvolutionaryAlgorithmTest.cc.

Here is the call graph for this function:



4.42 EvolutionaryAlgorithmTest.cc

Go to the documentation of this file.

```
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
00017
          std::vector<Chromosome> population = evo.getPopulation();
00018
00019
00020
          // assert that all body lengths are within the range
          for (auto& chrom : population) {
00022
              for (auto& length : chrom.bodyLengths) {
                  ASSERT_GE(length, EvolutionaryAlgorithmConfig::MIN_BODY_LENGTH);
00023
00024
                  ASSERT_LE(length, EvolutionaryAlgorithmConfig::MAX_BODY_LENGTH);
00025
00026
          }
00027 }
00028
{\tt 00029\ TEST} \ ({\tt EvolutionaryAlgorithmTest},\ {\tt 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();
          for (int i = 0; i < 49; ++i) {
```

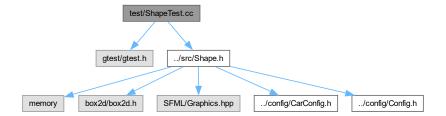
```
00041
              evo.nextGeneration();
00042
00043
00044
          ASSERT_EQ(evo.getGeneration(), 50);
00045 }
00046
00047 TEST (EvolutionaryAlgorithmTest, SaveToJsonTest) {
00048
          EvolutionaryAlgorithm evo(2, true);
00049
          evo.nextGeneration();
00050
          for (int i = 0; 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 Shape Test
```

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]

Definition at line 12 of file ShapeTest.cc.

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



4.43.2.2 TEST() [2/16]

```
TEST (
                  CreateBoxTest ,
                  InvalidDensityTest )
Definition at line 48 of file ShapeTest.cc.
00048
            // Test with invalid input (negative width and height)
b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
00050
            float x = 1.0f, y = 1.0f, width = 10.0f, height = 1.0f, density = -1.0f, friction = 0.5f; sf::Color color = sf::Color::Red;
00051
00052
00053
            {\tt ASSERT\_THROW(createBox(world,\ x,\ y,\ width,\ height,\ density,\ friction,\ color),}
00054
                            std::invalid_argument);
00055
00056 }
```

Here is the call graph for this function:



4.43.2.3 TEST() [3/16]



4.43.2.4 TEST() [4/16]

00044

00046 }

 ${\tt ASSERT_THROW(createBox(world,\ x,\ y,\ width,\ height,\ density,\ friction,\ color),}$

std::invalid_argument);

Here is the call graph for this function:



4.43.2.5 TEST() [5/16]

Definition at line 28 of file ShapeTest.cc.



4.43.2.6 TEST() [6/16]

```
TEST (
                  CreateCircleTest ,
                   BasicTest )
Definition at line 94 of file ShapeTest.cc.
            b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
float x = 10.0f, y = 20.0f, radius = 2.0f, density = 1.0f, friction = 0.5f;
sf::Color color(255, 255, 0);
00095
00096
00097
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);
             ASSERT_EQ(circle.body->GetType(), b2_dynamicBody);
00103
            ASSERT_EQ(circle.body->GetPosition(), b2Vec2(x / Config::PPM, y / Config::PPM));
ASSERT_EQ(circle.body->GetFixtureList()->GetDensity(), density);
00104
00105
00106
             ASSERT_EQ(circle.body->GetFixtureList()->GetFriction(), friction);
00107 }
```

Here is the call graph for this function:



4.43.2.7 TEST() [7/16]

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 }
```



4.43.2.8 TEST() [8/16]

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);
```

Here is the call graph for this function:



4.43.2.9 TEST() [9/16]

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);
```



4.43.2.10 TEST() [10/16]

```
TEST (
                CreateGroundTest ,
                BasicTest )
Definition at line 68 of file ShapeTest.cc.
00068
          b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
float x = 42.0f, y = 42.0f;
00069
00070
00071
           std::vector<b2Vec2> vertices = {b2Vec2(-25.0f, -5.0f), b2Vec2(25.0f, -5.0f),
                                              b2Vec2(25.0f, 5.0f), b2Vec2(-25.0f, 5.0f));
00072
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);
          ASSERT_EQ(ground.vertices, vertices);
ASSERT_EQ(ground.color, color);
00078
00079
00080
           ASSERT_EQ(ground.body->GetType(), b2_staticBody);
          ASSERT_EQ(ground.body->GetFixtureList()->GetDensity(), 0.0f);
00081
```

Here is the call graph for this function:

00082 }



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;
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 }
```



4.43.2.12 TEST() [12/16]

```
TEST (
                 CreatePolygonTest ,
                 BasicTest )
Definition at line 136 of file ShapeTest.cc.
           b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
float x = 0.0f, y = 0.0f, density = 1.0f, friction = 0.5f;
00137
00138
           std::vector<b2Vec2> vertices = {b2Vec2(-1.0f, -1.0f), b2Vec2(1.0f, -1.0f), b2Vec2(0.0f, 1.0f)}; sf::Color color = sf::Color::Red;
00139
00140
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(
                \label{linear_cast} $$\operatorname{dynamic\_cast}<\operatorname{const} b2PolygonShape*>(fixture->GetShape()), [](\operatorname{const} b2PolygonShape* s) $$\{\}$);
00151
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 (

00165 }



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
00187}
```

Here is the call graph for this function:



4.43.2.15 TEST() [15/16]

Definition at line 189 of file ShapeTest.cc.



4.43.2.16 TEST() [16/16]

00171 00172 00173

00174 00175

00176 }

```
TEST (
                       CreatePolygonTest .
                       TooMuchVerticesTest )
Definition at line 167 of file ShapeTest.cc.
00168
                b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
               float x = 0.0f, y = 0.0f, density = 1.0f, friction = 0.5f; std::vector<b2Vec2> vertices = {b2Vec2(-1.0f, -1.0f), b2Vec2(1.0f, -1.0f), b2Vec2(0.0f, 1.0f), b2Vec2(-1.0f, -1.0f), b2Vec2(1.0f, -1.0f), b2Vec2(0.0f, 1.0f), b2Vec2(-1.0f, -1.0f), b2Vec2(1.0f, -1.0f), b2Vec2(0.0f, 1.0f)};
00169
00170
```

ASSERT_THROW(createPolygon(world, x, y, vertices, density, friction, color);

Here is the call graph for this function:

sf::Color color = sf::Color::Red;

, std::invalid_argument);



ShapeTest.cc 4.44

Go to the documentation of this file.

```
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));
           float x = 0.0f, y = 0.0f, width = 10.0f, height = 20.0f, density = 1.0f, friction = 0.5f; sf::Color color = sf::Color::Red;
00014
00015
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);
           EXPECT_EQ(box.body->GetFixtureList()->GetDensity(), density);
00024
00025
           EXPECT_EQ(box.body->GetFixtureList()->GetFriction(), friction);
00026 }
00027
00028 TEST(CreateBoxTest, InvalidWidthTest) {
00029
           // Test with invalid input (negative width and height)
           b2WorldPtr world = std::make_shared<br/>b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));<br/>float x = 0.0f, y = 0.0f, width = -10.0f, height = 10.0f, density = 1.0f, friction = 0.5f;<br/>sf::Color color = sf::Color::Red;
00030
00031
00032
00033
00034
           {\tt 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));
           float x = 0.0f, y = 0.0f, width = 10.0f, height = 0.0f, density = 1.0f, friction = 0.5f;
00041
00042
           sf::Color color = sf::Color::Red;
00043
00044
           ASSERT_THROW(createBox(world, x, y, width, height, density, friction, color),
                          std::invalid_argument);
```

4.44 ShapeTest.cc 145

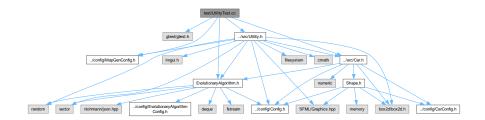
```
00046 }
00047
00048 TEST(CreateBoxTest, InvalidDensityTest) {
00049
           // Test with invalid input (negative width and height)
          b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION)); float x = 1.0f, y = 1.0f, width = 10.0f, height = 1.0f, density = -1.0f, friction = 0.5f; sf::Color color = sf::Color::Red;
00050
00051
00052
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));
           float x = 1.0f, y = 1.0f, width = 10.0f, height = 5.0f, density = 2.0f, friction = 0.0f;
00061
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)};
           sf::Color color = sf::Color::Blue;
00073
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);
          ASSERT_EQ(ground.body->GetType(), b2_staticBody);
ASSERT_EQ(ground.body->GetFixtureList()->GetDensity(), 0.0f);
00080
00081
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;
std::vector<b2Vec2> invalidVertices; // Empty vertices
00087
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));
           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 EO(circle.radius, radius);
           ASSERT_EQ(circle.color, color);
00102
           ASSERT_EQ(circle.body->GetType(), b2_dynamicBody);
00103
00104
           ASSERT_EQ(circle.body->GetPosition(), b2Vec2(x / Config::PPM, y / Config::PPM));
          ASSERT_EQ(circle.body->GetFixtureList()->GetDensity(), density);
ASSERT_EQ(circle.body->GetFixtureList()->GetFriction(), friction);
00105
00106
00107 }
00108
00109 TEST(CreateCircleTest, InvalidRadiusTest) {
00110
          b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
          float x = 10.0f, y = 20.0f, radius = 0.0f, density = 1.0f, friction = 0.5f; sf::Color color(255, 255, 0);
00111
00112
00113
00114
           ASSERT_THROW(createCircle(world, x, y, radius, density, friction, color),
00115
                        std::invalid_argument);
00116 }
00117
00121
          sf::Color color(255, 255, 0);
00122
00123
          {\tt ASSERT\_THROW} ({\tt createCircle} ({\tt world}, \ {\tt x}, \ {\tt y}, \ {\tt radius}, \ {\tt density}, \ {\tt friction}, \ {\tt 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, v, radius, density, friction, color),
```

```
00133
                                      std::invalid_argument);
00134 }
00135
00136 TEST(CreatePolygonTest, BasicTest) {
                b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION)); float x = 0.0f, y = 0.0f, density = 1.0f, friction = 0.5f;
00137
00138
                 std::vector<b2Vec2> vertices = {b2Vec2(-1.0f, -1.0f), b2Vec2(1.0f, -1.0f), b2Vec2(0.0f, 1.0f)};
                 sf::Color color = sf::Color::Red;
00140
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) {
                                   world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
00159
00160
                 float x = 0.0f, y = 0.0f, density = 1.0f, friction = 0.5f;
00161
                 std::vector<b2Vec2> vertices = {};
                 sf::Color color = sf::Color::Red;
00162
00163
                ASSERT_THROW(createPolygon(world, x, y, vertices, density, friction, color);
00164
                                     , std::invalid_argument);
00165 }
00166
00167 TEST(CreatePolygonTest, TooMuchVerticesTest) {
               00168
00169
00171
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
00180
00181
00182
00183
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) {
                b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
00190
00191
                 float x = 0.0f, y = 0.0f, density = 1.0f, friction = -0.5f;
                 {\tt std::vector < b2Vec2 > vertices = \{b2Vec2 (-1.0f, -1.0f), b2Vec2 (1.0f, -1.0f), b2Vec2 (0.0f, 1.0f), b2Vec2 (
00192
                                                                      b2Vec2(-1.0f, -1.0f), b2Vec2(1.0f, -1.0f), b2Vec2(0.0f, 1.0f), b2Vec2(-1.0f, -1.0f), b2Vec2(1.0f, -1.0f)};
00193
00194
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

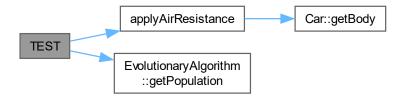
- TEST (UtilityTest, applyAirResistanceTest)
- TEST (UtilityTest, generateGroundTest)
- TEST (UtilityTest, getNextGroundPartDegreeTest)
- TEST (UtilityTest, generateCarTest)
- TEST (UtilityTest, getFurthestCarPosTest)
- TEST (UtilityTest, SFMLColorToImVec4)
- TEST (UtilityTest, getIdxOfGrndClstToLocTest)
- TEST (UtilityTest, removeCarsTest)
- TEST (UtilityTest, getRootDirTest)
- TEST (UtilityTest, setIconTest)

4.45.1 Function Documentation

4.45.1.1 TEST() [1/10]

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);
          Chromosome chromosome = ea.getPopulation()[0];
00023
00024
          Car car = Car(world, x, y, chromosome, bodyColor, wheelColor);
00025
00026
          EXPECT_NO_THROW(applyAirResistance(car));
00027 }
```



4.45.1.2 TEST() [2/10]

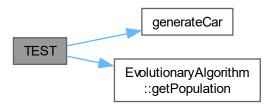
```
TEST (
                UtilityTest ,
                generateCarTest )
Definition at line 60 of file UtilityTest.cc.
00060
00061
           b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
           float x = 0.0f, y = 0.0f;
sf::Color bodyColor = sf::Color::Red;
00062
00063
00064
           sf::Color wheelColor = sf::Color::Blue;
00065
00066
           {\tt EvolutionaryAlgorithm\ ea = EvolutionaryAlgorithm\ (EvolutionaryAlgorithmConfig::POPULATION\_SIZE);}
00067
          Chromosome chromosome = ea.getPopulation()[0];
00068
```

Here is the call graph for this function:

EXPECT_NO_THROW(generateCar(world, chromosome));

00069

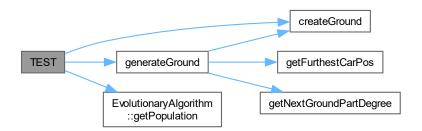
00070 }



4.45.1.3 TEST() [3/10]

```
Definition at line 29 of file UtilityTest.cc.
00029
00030
         b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
         float x = 0.0f, y = 0.0f;
sf::Color bodyColor = sf::Color::Red;
00031
00032
         sf::Color wheelColor = sf::Color::Blue;
00033
00034
00035
         {\tt 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
00043
00044
00045
00046
00047
00048
00049
00050
         std::vector<Polygon> groundVector = {createGround(world, MapGenConfig::GROUND_STARTING_X,
                                                           MapGenConfig::GROUND_STARTING_Y,
00051
                                                           groundVertecies, sf::Color(18, 36, 35))};
00052
00053
         EXPECT_NO_THROW(generateGround(world, &groundVector, cars));
```

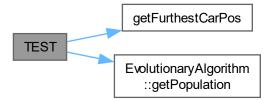
Here is the call graph for this function:



4.45.1.4 TEST() [4/10]

Definition at line 72 of file UtilityTest.cc.

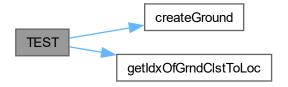
```
00072
           b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
float x = 0.0f, y = 0.0f;
sf::Color bodyColor = sf::Color::Red;
sf::Color wheelColor = sf::Color::Blue;
00073
00075
00076
00078
            {\tt EvolutionaryAlgorithm\ ea = EvolutionaryAlgorithm\ (EvolutionaryAlgorithmConfig::POPULATION\_SIZE);}
00079
            Chromosome chromosome = ea.getPopulation()[0];
08000
            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 }
```



4.45.1.5 TEST() [5/10]

```
TEST (
                  UtilityTest ,
                  getIdxOfGrndClstToLocTest )
Definition at line 100 of file UtilityTest.cc.
            b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
std::vector<b2Vec2> groundVertecies = {
   b2Vec2(MapGenConfig::GROUND_STARTING_X, MapGenConfig::GROUND_STARTING_Y),
   b2Vec2(MapGenConfig::GROUND_STARTING_X + MapGenConfig::GROUND_LEG_LENGTH,
00101
00102
00103
00104
                          MapGenConfig::GROUND_STARTING_Y),
00105
                 00106
00107
00108
00109
            Polygon ground =
                 createGround(world, MapGenConfig::GROUND_STARTING_X, MapGenConfig::GROUND_STARTING_Y,
00110
00111
                                  groundVertecies, sf::Color(18, 36, 35));
00112
00113
            std::vector<Polygon> groundVector;
00114
            groundVector.push_back(ground);
00115
            EXPECT_NO_THROW(getIdxOfGrndClstToLoc(groundVector, 0.0f));
EXPECT_EQ(getIdxOfGrndClstToLoc(groundVector, 0.0f), 0);
00116
00117
00118 }
```

Here is the call graph for this function:



4.45.1.6 TEST() [6/10]



4.45.1.7 TEST() [7/10]

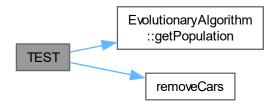
Here is the call graph for this function:



4.45.1.8 TEST() [8/10]

Definition at line 120 of file UtilityTest.cc.

```
00120
           b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION)); float x = 0.0f, y = 0.0f;
00121
00122
00123
           sf::Color bodyColor = sf::Color::Red;
00124
           sf::Color wheelColor = sf::Color::Blue;
00125
           EvolutionaryAlgorithm ea = EvolutionaryAlgorithm(EvolutionaryAlgorithmConfig::POPULATION_SIZE);
Chromosome chromosome = ea.getPopulation()[0];
00126
00127
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 }
```



4.45.1.9 TEST() [9/10]

Here is the call graph for this function:



4.45.1.10 TEST() [10/10]

Definition at line 90 of file UtilityTest.cc.

```
00090
00090
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);
```



4.46 UtilityTest.cc 153

4.46 UtilityTest.cc

Go to the documentation of this file.

```
00009 #include <gtest/gtest.h>
00010
00011 #include "../config/MapGenConfig.h"
00012 #include "../src/Car.h"
00013 #include "../src/EvolutionaryAlgorithm.h"
00013 #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);
          Chromosome chromosome = ea.getPopulation()[0];
00036
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
00046
00047
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);
          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));
          EXPECT_EQ(getFurthestCarPos(cars).x, x);
00087
          EXPECT_EQ(getFurthestCarPos(cars).y, y);
00088 }
00089
```

```
00090 TEST(UtilityTest, SFMLColorToImVec4) {
         sf::Color color = sf::Color::Red;
ImVec4 imVec4 = SFMLColorToImVec4(color);
00091
00092
00093
00094
          EXPECT_EQ(imVec4.x, 1.0f);
          EXPECT_EQ(imVec4.y, 0.0f);
00095
00096
          EXPECT_EQ(imVec4.z, 0.0f);
00097
          EXPECT_EQ(imVec4.w, 1.0f);
00098 }
00099
00100 TEST(UtilityTest, getIdxOfGrndClstToLocTest) {
         b2WorldPtr world = std::make_shared<b2World>(b2Vec2(0.0f, Config::GRAVITATIONAL_ACCELERATION));
00101
00102
          std::vector<b2Vec2> groundVertecies = {
              b2Vec2(MapGenConfig::GROUND_STARTING_X, MapGenConfig::GROUND_STARTING_Y),
00103
00104
              b2Vec2(MapGenConfig::GROUND_STARTING_X + MapGenConfig::GROUND_LEG_LENGTH,
             00105
00106
00107
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
          EvolutionaryAlgorithm ea = EvolutionaryAlgorithm(EvolutionaryAlgorithmConfig::POPULATION_SIZE);
00126
         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) {
          sf::RenderWindow window(sf::VideoMode(800, 600), "SFML window");
00141
          EXPECT_NO_THROW(setIcon(window));
00142 }
```

Index

| applyAirResistance | ManGanConfig 26 |
|------------------------|--|
| Utility.cc, 103 | MapGenConfig, 36 CAR TORQUE |
| Utility.h, 117 | CarConfig, 15 |
| Othity.ii, 117 | CAR_VERTICES |
| b2WorldPtr | CarConfig, 15 |
| Car.h, 50 | CarConfig, 14 |
| Main.cc, 65 | CAR TORQUE, 15 |
| Shape.h, 91 | CAR_VERTICES, 15 |
| Utility.h, 117 | MAX JOINT LENGTH, 15 |
| BACK_WHEEL_POS | CarTest.cc |
| Config, 20 | TEST, 129 |
| BG_SPRITES_COUNT | CATEGORY_BITS |
| MapGenConfig, 36 | Config, 20 |
| body | Chromosome, 15 |
| Box, 6 | bodyDensity, 17 |
| Circle, 18 | bodyLengths, 17 |
| Polygon, 39 | fitness, 17 |
| bodyDensity | wheelDensity, 17 |
| Chromosome, 17 | wheelRadius, 17 |
| bodyLengths | Circle, 18 |
| Chromosome, 17 | body, 18 |
| Box, 5 | color, 18 |
| body, 6 | radius, 18 |
| color, 6 | color |
| height, 6 | Box, 6 |
| width, 6 | Circle, 18 |
| , | Polygon, 39 |
| Car, 6 | Config, 19 |
| Car, 8 | BACK_WHEEL_POS, 20 |
| getBackWheel, 9 | CATEGORY BITS, 20 |
| getBody, 9 | DEBUG, 20 |
| getBodyColor, 10 | DEG_PER_RAD, 20 |
| getFrontWheel, 10 | FRICTION, 20 |
| getPosX, 11 | FRONT_WHEEL_POS, 20 |
| getPosXVec, 11 | GENERATION TIME, 20 |
| getPosY, 11 | GRAVITATIONAL ACCELERATION, 21 |
| getPosYVec, 11 | GROUND PARTS RENDERED, 21 |
| getVelocity, 12 | MASK_BITS, 21 |
| getVelocityVec, 12 | MAX_FPS, 21 |
| getVelX, 12 | PI, 21 |
| getVelY, 13 | PPM, 21 |
| setCollisionFilter, 13 | SAVE FILE NAME, 21 |
| Car.cc | SAVE TO FILE, 22 |
| createVertices, 47 | VELOCITY ARRAY SIZE, 22 |
| Car.h | WINDOW HEIGHT, 22 |
| b2WorldPtr, 50 | WINDOW_WIDTH, 22 |
| createVertices, 50 | config/CarConfig.h, 41, 42 |
| CAR_STARTING_X | config/Config.h, 42, 43 |
| MapGenConfig, 36 | config/EvolutionaryAlgorithmConfig.h, 43, 44 |
| CAR_STARTING_Y | · · · |

| config/MapGenConfig.h, 45, 46 | EvolutionaryAlgorithmTest.cc |
|--|------------------------------|
| createBox | TEST, 132, 133 |
| Shape.cc, 84 | exportPopulation |
| Shape.h, 91 | EvolutionaryAlgorithm, 24 |
| createCircle | Evolutionally agont mi, E1 |
| | fitness |
| Shape.cc, 85 | Chromosome, 17 |
| Shape.h, 92 | FRICTION |
| createGround | |
| Shape.cc, 86 | Config, 20 |
| Shape.h, 93 | FRONT_WHEEL_POS |
| createPolygon | Config, 20 |
| Shape.cc, 87 | |
| Shape.h, 94 | GENERATE_DISTANCE |
| createVertices | MapGenConfig, 36 |
| Car.cc, 47 | generateCar |
| Car.h, 50 | Utility.cc, 104 |
| Cal.ii, 50 | Utility.h, 118 |
| DEBUG | generateGround |
| | Utility.cc, 105 |
| Config, 20 | Utility.h, 119 |
| DEG_PER_RAD | - |
| Config, 20 | GENERATION_TIME |
| | Config, 20 |
| EvolutionaryAlgorithm, 23 | getBackWheel |
| EvolutionaryAlgorithm, 24 | Car, 9 |
| exportPopulation, 24 | getBody |
| getGeneration, 24 | Car, 9 |
| getPopulation, 25 | getBodyColor |
| getPopulationSize, 26 | Car, 10 |
| mutate, 26 | getFrontWheel |
| nextGeneration, 27 | Car, 10 |
| setFitness, 28 | getFurthestCarPos |
| | _ |
| tournamentSelection, 29 | Utility.cc, 106 |
| EvolutionaryAlgorithmConfig, 30 | Utility.h, 120 |
| INITIAL_BODY_DENSITY_MEAN, 31 | getGeneration |
| INITIAL_BODY_DENSITY_VARIANCE, 31 | EvolutionaryAlgorithm, 24 |
| INITIAL_BODY_LENGTH_MEAN, 31 | getIdxOfGrndClstToLoc |
| INITIAL_BODY_LENGTH_VARIANCE, 31 | Utility.cc, 107 |
| INITIAL_WHEEL_DENSITY_MEAN, 32 | Utility.h, 121 |
| INITIAL_WHEEL_DENSITY_VARIANCE, 32 | getNextGroundPartDegree |
| INITIAL WHEEL RADIUS MEAN, 32 | Utility.cc, 108 |
| INITIAL WHEEL RADIUS VARIANCE, 32 | Utility.h, 122 |
| MAX BODY DENSITY, 32 | getPopulation |
| MAX BODY LENGTH, 32 | Evolutionary Algorithm, 25 |
| MAX_BODT_EENOTH; 32 MAX_WHEEL_DENSITY, 32 | getPopulationSize |
| | |
| MAX_WHEEL_RADIUS, 33 | EvolutionaryAlgorithm, 26 |
| MIN_BODY_DENSITY, 33 | getPosX |
| MIN_BODY_LENGTH, 33 | Car, 11 |
| MIN_WHEEL_DENSITY, 33 | getPosXVec |
| MIN_WHEEL_RADIUS, 33 | Car, 11 |
| MUTATION_FACTOR_BODY_DENSITY, 33 | getPosY |
| MUTATION FACTOR BODY LENGTHS, 33 | Car, 11 |
| MUTATION FACTOR WHEEL DENSITY, 34 | getPosYVec |
| MUTATION FACTOR WHEEL RADIUS, 34 | Car, 11 |
| MUTATION RATE BODY DENSITY, 34 | getRootDir |
| MUTATION RATE BODY LENGTHS, 34 | Utility.cc, 108 |
| | - |
| MUTATION_RATE_WHEEL_DENSITY, 34 | Utility.h, 122 |
| MUTATION_RATE_WHEEL_RADIUS, 34 | getVelocity |
| POPULATION_SIZE, 34 | Car, 12 |
| TOURNAMENT_SIZE, 35 | getVelocityVec |
| | |

| Car, 12 | Utility.cc, 110 |
|---------------------------------|--|
| getVelX | Utility.h, 124 |
| Car, 12 | • ' |
| getVelY | main |
| Car, 13 | Main.cc, 65 |
| GRAVITATIONAL_ACCELERATION | Main.cc |
| Config, 21 | b2WorldPtr, 65 |
| GROUND_DEGREE_DEVIATION | main, 65 |
| MapGenConfig, 36 | world, 67 |
| GROUND_LEG_LENGTH | MapGenConfig, 35 |
| MapGenConfig, 36 | BG_SPRITES_COUNT, 36 CAR_STARTING_X, 36 |
| GROUND_PART_LENGTH | CAR STARTING_X, 36 |
| MapGenConfig, 37 | GENERATE DISTANCE, 36 |
| GROUND_PARTS_RENDERED | GROUND DEGREE DEVIATION, 36 |
| Config, 21 | GROUND_LEG_LENGTH, 36 |
| GROUND_STARTING_X | GROUND_PART_LENGTH, 37 |
| MapGenConfig, 37 | GROUND STARTING X, 37 |
| GROUND_STARTING_Y | GROUND STARTING Y, 37 |
| MapGenConfig, 37 GUI.cc | MAX GROUND DEGREE, 37 |
| renderPositionPlot, 57 | MASK BITS |
| render/VelocityPlot, 58 | Config, 21 |
| GUI.h | MAX_BODY_DENSITY |
| renderPositionPlot, 61 | EvolutionaryAlgorithmConfig, 32 |
| render/VelocityPlot, 62 | MAX_BODY_LENGTH |
| Torradi Volodity Flot, 62 | EvolutionaryAlgorithmConfig, 32 |
| handleEvents | MAX_FPS |
| UserInput.cc, 97 | Config, 21 |
| UserInput.h, 100 | MAX_GROUND_DEGREE |
| handleUserInput | MapGenConfig, 37 |
| UserInput.cc, 97 | MAX_JOINT_LENGTH |
| UserInput.h, 101 | CarConfig, 15 |
| height | MAX_WHEEL_DENSITY |
| Box, 6 | EvolutionaryAlgorithmConfig, 32 |
| | MAX_WHEEL_RADIUS |
| INITIAL_BODY_DENSITY_MEAN | EvolutionaryAlgorithmConfig, 33 |
| EvolutionaryAlgorithmConfig, 31 | MIN_BODY_DENSITY |
| INITIAL_BODY_DENSITY_VARIANCE | EvolutionaryAlgorithmConfig, 33 |
| EvolutionaryAlgorithmConfig, 31 | MIN_BODY_LENGTH |
| INITIAL_BODY_LENGTH_MEAN | EvolutionaryAlgorithmConfig, 33 |
| EvolutionaryAlgorithmConfig, 31 | MIN_WHEEL_DENSITY |
| INITIAL_BODY_LENGTH_VARIANCE | EvolutionaryAlgorithmConfig, 33 |
| EvolutionaryAlgorithmConfig, 31 | MIN_WHEEL_RADIUS |
| INITIAL_WHEEL_DENSITY_MEAN | EvolutionaryAlgorithmConfig, 33 |
| EvolutionaryAlgorithmConfig, 32 | mutate |
| INITIAL_WHEEL_DENSITY_VARIANCE | EvolutionaryAlgorithm, 26 |
| EvolutionaryAlgorithmConfig, 32 | MUTATION_FACTOR_BODY_DENSITY |
| INITIAL_WHEEL_RADIUS_MEAN | EvolutionaryAlgorithmConfig, 33 |
| EvolutionaryAlgorithmConfig, 32 | MUTATION_FACTOR_BODY_LENGTHS |
| INITIAL_WHEEL_RADIUS_VARIANCE | EvolutionaryAlgorithmConfig, 33 |
| EvolutionaryAlgorithmConfig, 32 | MUTATION_FACTOR_WHEEL_DENSITY |
| loadBGSprite | EvolutionaryAlgorithmConfig, 34 |
| Utility.cc, 109 | MUTATION_FACTOR_WHEEL_RADIUS |
| Utility.h, 123 | EvolutionaryAlgorithmConfig, 34 |
| loadBGSprites | MUTATION_RATE_BODY_DENSITY EvolutionaryAlgorithmConfig, 34 |
| Utility.cc, 110 | MUTATION_RATE_BODY_LENGTHS |
| Utility.h, 124 | EvolutionaryAlgorithmConfig, 34 |
| loadBGTextures | MUTATION_RATE_WHEEL_DENSITY |
| | MO IVION TIVIT MUEET DEMOULT |

| EvolutionaryAlgorithmConfig, 34 | GUI.cc, 58 |
|---------------------------------|---|
| MUTATION_RATE_WHEEL_RADIUS | GUI.h, 62 |
| EvolutionaryAlgorithmConfig, 34 | CAVE FILE NAME |
| novtConoration | SAVE_FILE_NAME |
| nextGeneration | Config, 21 |
| EvolutionaryAlgorithm, 27 | SAVE_TO_FILE |
| PI | Config, 22 |
| Config, 21 | setCollisionFilter |
| 5 , | Car, 13 |
| Polygon, 38 | setFitness |
| body, 39 | EvolutionaryAlgorithm, 28 |
| color, 39 | setIcon |
| vertices, 39 | Utility.cc, 112 |
| POPULATION_SIZE | Utility.h, 126 |
| EvolutionaryAlgorithmConfig, 34 | SFMLColorToImVec4 |
| PPM | Utility.cc, 113 |
| Config, 21 | Utility.h, 127 |
| and divine | Shape.cc |
| radius | createBox, 84 |
| Circle, 18 | createCircle, 85 |
| removeCars | createGround, 86 |
| Utility.cc, 111 | createPolygon, 87 |
| Utility.h, 125 | Shape.h |
| render | b2WorldPtr, 91 |
| Render.cc, 70 | createBox, 91 |
| Render.h, 78 | createCircle, 92 |
| Render.cc | createGround, 93 |
| render, 70 | createPolygon, 94 |
| renderCar, 72 | ShapeTest.cc |
| renderCircle, 73 | TEST, 136–143 |
| renderCircleDebug, 73 | src/Car.cc, 46, 47 |
| renderPolygon, 74 | src/Car.h, 48, 50 |
| renderPolygonDebug, 74 | src/EvolutionaryAlgorithm.cc, 51, 52 |
| Render.h | src/EvolutionaryAlgorithm.h, 54, 55 |
| render, 78 | src/GUI.cc, 56, 59 |
| renderCar, 79 | src/GUI.h, 60, 63 |
| renderCircle, 80 | src/Main.cc, 63, 68 |
| renderCircleDebug, 81 | src/Render.cc, 69, 75 |
| renderPolygon, 81 | src/Render.h, 77, 83 |
| renderPolygonDebug, 82 | src/Shape.cc, 83, 88 |
| renderCar | src/Shape.h, 90, 95 |
| Render.cc, 72 | src/UserInput.cc, 96, 98 |
| Render.h, 79 | src/UserInput.h, 99, 102 |
| renderCircle | src/Utility.cc, 102, 114 |
| Render.cc, 73 | src/Utility.h, 115, 128 |
| Render.h, 80 | 310/ Othity.11, 110, 120 |
| renderCircleDebug | TEST |
| Render.cc, 73 | CarTest.cc, 129 |
| Render.h, 81 | EvolutionaryAlgorithmTest.cc, 132, 133 |
| renderPolygon | ShapeTest.cc, 136-143 |
| Render.cc, 74 | UtilityTest.cc, 147–152 |
| Render.h, 81 | test/CarTest.cc, 128, 130 |
| renderPolygonDebug | test/EvolutionaryAlgorithmTest.cc, 131, 134 |
| Render.cc, 74 | test/ShapeTest.cc, 135, 144 |
| Render.h, 82 | test/UtilityTest.cc, 146, 153 |
| renderPositionPlot | TOURNAMENT_SIZE |
| GUI.cc, 57 | EvolutionaryAlgorithmConfig, 35 |
| GUI.h, 61 | tournamentSelection |
| renderVelocityPlot | EvolutionaryAlgorithm, 29 |

```
UserInput.cc
    handleEvents, 97
    handleUserInput, 97
UserInput.h
    handleEvents, 100
    handleUserInput, 101
Utility.cc
    applyAirResistance, 103
    generateCar, 104
    generateGround, 105
    getFurthestCarPos, 106
    getIdxOfGrndClstToLoc, 107
    getNextGroundPartDegree, 108
    getRootDir, 108
    loadBGSprite, 109
    loadBGSprites, 110
    loadBGTextures, 110
    removeCars, 111
    setIcon, 112
    SFMLColorToImVec4, 113
Utility.h
    applyAirResistance, 117
    b2WorldPtr, 117
    generateCar, 118
    generateGround, 119
    getFurthestCarPos, 120
    getIdxOfGrndClstToLoc, 121
    getNextGroundPartDegree, 122
    getRootDir, 122
    loadBGSprite, 123
    loadBGSprites, 124
    loadBGTextures, 124
    removeCars, 125
    setIcon, 126
    SFMLColorToImVec4, 127
UtilityTest.cc
    TEST, 147-152
VELOCITY_ARRAY_SIZE
    Config, 22
vertices
     Polygon, 39
wheelDensity
    Chromosome, 17
wheelRadius
    Chromosome, 17
width
    Box, 6
WINDOW_HEIGHT
    Config, 22
WINDOW_WIDTH
    Config, 22
world
    Main.cc, 67
```