Robot Simulation 2015

0.1.500

Generated by Doxygen 1.8.6

Tue Mar 3 2015 14:36:32

Contents

Chapter 1

My Personal Index Page

1.1 Introduction

This is the introduction.

1.2 Installation

This is how you make or install my software

1.2.1 Step 1: Opening the box

Here is some more details.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

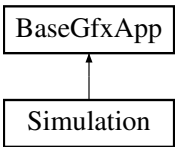
Here is a list of all documented files with brief descriptions:

Chapter 5

Class Documentation

5.1 BaseGfxApp Class Reference

Inheritance diagram for BaseGfxApp:



Public Member Functions

- **BaseGfxApp** (int argc, char ∗argv[], int width, int height, int x, int y, int glutFlags, bool createGLUIWin, int gluiWinX, int gluiWinY)
- void **setCaption** (const std::string &caption)
- void **runMainLoop** ()
- virtual void **display** ()
- virtual void **mouseMoved** (int x, int y)
- virtual void **mouseDragged** (int x, int y)
- virtual void **leftMouseDown** (int x, int y)
- virtual void **leftMouseUp** (int x, int y)
- virtual void **rightMouseDown** (int x, int y)
- virtual void **rightMouseUp** (int x, int y)
- virtual void **middleMouseDown** (int x, int y)
- virtual void **middleMouseUp** (int x, int y)
- virtual void **keyboard** (unsigned char c, int x, int y)
- virtual void **keyboardSpecial** (int key, int x, int y)
- virtual void **keyboardUp** (unsigned char c, int x, int y)
- virtual void **keyboardSpecialUp** (int key, int x, int y)
- virtual void **reshape** (int width, int height)
- virtual void **gluiControl** (int controlID)
- int **width** () const
- int **height** () const
- int **handle** ()
- GLUI ∗ **glui** ()

Static Protected Member Functions

- static void **s_reshape** (int width, int height)
- static void **s_keyboard** (unsigned char c, int x, int y)
- static void **s_keyboardspecial** (int key, int x, int y)
- static void **s_keyboardup** (unsigned char c, int x, int y)
- static void **s_keyboardspecialup** (int key, int x, int y)
- static void **s_mousemotion** (int x, int y)
- static void **s_mousebtn** (int b, int s, int x, int y)
- static void **s_draw** ()
- static void **s_gluicallback** (int controlID)

Protected Attributes

- int m_glutWindowHandle

  Underlying glut window handle.
- GLUI * **m_glui**
- bool **m_drag**
- int **m_width**
- int **m_height**

Static Protected Attributes

- static BaseGfxApp * s_currentApp = NULL

  GLUT and GLUI event callbacks are sent to the current window/app.
- static bool s_glutInitialized = false

  Has glutInit been called? (only allowed once per program)

5.1.1 Member Data Documentation

5.1.1.1 BaseGfxApp * BaseGfxApp::s_currentApp = NULL [static], [protected]

GLUT and GLUI event callbacks are sent to the current window/app.

Right now, there is only one window anyway (not counting the GLUI UI window.. in the future could be extended to support more windows. In any case, some structure like this is always needed when using glut with C++, since the glut callbacks must be either global or static functions.

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

- BaseGfxApp.h
- BaseGfxApp.cpp

5.2 BoundaryStruct Struct Reference

Contains information about a retangular boundary.

`#include <BoundaryStruct.h>`

Public Member Functions

- BoundaryStruct ()

  A default constructor.
- BoundaryStruct (SizeStruct &size)

  A normal constructor for setting up the boundary structure.
- BoundaryStruct (SizeStruct &size, int margin)

  A constructor setting the inner borders thought a margin.
- BoundaryStruct (SizeStruct &size, int tp, int bttm, int lft, int rght)

  A constructor setting the inner borders with preset values.
- void setMargin (int margin)

  Set border values according to a preset margin.

Public Attributes

- SizeStruct outerBoundarySize

  The outer boundary's size.
- int top

  Top border.
- int bottom

  Bottom border.
- int left

  Left border.
- int right

  Right border.

5.2.1 Detailed Description

Contains information about a retangular boundary.

The boundary is in fact an innner boundary within an outer one. Because of that, user has to provide information about the outer boundary, specifically its size. User also has to make sure the outer boundary size's width and height are non-negative values.

The inner boundary is represented as four border values top, bottom, left, and right. Assuming the outer boundary is a rectangle on a normal XY-coordinate system with its bottom left corner at origin and its top right corner at location (outerBoundarySize.width, outerBoundarySize.height), the inner boundary's top border is the y-intercept of its top line, bottom is the y-intercept of its bottom line, left is the x-intercept of its left line, and right is the x-intercept of its right line.

The border values can have negative values; however, then the inner boundary may not be "inner" boundary anymore.

See Also

SizeStruct

5.2.2 Constructor & Destructor Documentation

5.2.2.1 BoundaryStruct::BoundaryStruct ( ) [inline]

A default constructor.

WARNING: This is an empty constructor for the sake of place-holding.

Use this if absolutely needed! It is recommended to use others instead.

5.2.2.2 BoundaryStruct::BoundaryStruct (SizeStruct & *size*)

A normal constructor for setting up the boundary structure.

The outer boundary will be overlapped with the inner one. User has to make

sure the size parameter has non-negative values for both width and height.

Parameters

size	the size of the outer boundary

5.2.2.3 BoundaryStruct::BoundaryStruct (SizeStruct & *size,* int *margin*)

A constructor setting the inner borders thought a margin.

The inner boundary will be offset by the same amount of margin. Margin's

size should not exceed half the outer boundary's size, either width or height.

Negative values of margin will have unexpected behaviours. Use non-negative.

Parameters

size	the size of the outer boundary
margin	the preset margin

5.2.2.4 BoundaryStruct::BoundaryStruct (SizeStruct & *size,* int *tp,* int *bttm,* int *lft,* int *rght*)

A constructor setting the inner borders with preset values.

All preset values have to be appropriate, e.g. top has to be greater than

bottom. Allow negative values for the preset values of border.

Parameters

size	the size of the outer boundary
top	preset value for top border
bottom	preset value for bottom border
left	preset value for left border
right	preset value for right border

5.2.3 Member Function Documentation

5.2.3.1 void BoundaryStruct::setMargin (int *margin*)

Set border values according to a preset margin.

Similar to when construct a boundary struct through a margin.

Parameters

margin	the preset margin

See Also

BoundaryStruct::BoundaryStruct(SizeStruct&, int)

5.2.4 Member Data Documentation

5.2.4.1 int BoundaryStruct::bottom

Bottom border.

5.2.4.2 int BoundaryStruct::left

Left border.

5.2.4.3 SizeStruct BoundaryStruct::outerBoundarySize

The outer boundary's size.

5.2.4.4 int BoundaryStruct::right

Right border.

5.2.4.5 int BoundaryStruct::top

Top border.

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

- BoundaryStruct.h
- BoundaryStruct.cpp

5.3 ColorStruct Struct Reference

Contains information about color.

`#include <ColorStruct.h>`

Public Member Functions

- ColorStruct ()
  A default constructor.
- ColorStruct (int r, int g, int b)
  A constructor using RGB values.
- ColorStruct (std::string hex)
  A constructor using the hexadecimal representation.
- void setRGB ()
  Using hexadecimal string to set RGB values accordingly.
- void setHex ()
  Using RGB values to set hexadecimal string accordingly.

Public Attributes

- int rValue

  Red component of the color.
- int gValue

  Green component of the color.
- int bValue

  Blue component of the color.
- std::string hexStr

  Hexadecimal representation.

5.3.1 Detailed Description

Contains information about color.

Colors can be represented both in RGB values or hexadecimal string. The string has 7 characters beginning with a pound sign, e.g. #FFFFFF for color white.

The string can be in upper or lower case. The RGB values must be in range of [0, 255].

5.3.2 Constructor & Destructor Documentation

5.3.2.1 ColorStruct::ColorStruct ( ) [inline]

A default constructor.

Default color is black.

5.3.2.2 ColorStruct::ColorStruct (int *r*, int *g*, int *b*)

A constructor using RGB values.

The hexadecimal string will NOT be set automatically using this constructor.

User will have to set it manually if needed by calling setHex(). If one of the RBG values is out of range, set to default color black (0, 0, 0).

Parameters

r	red component
g	green component
b	blue component

See Also

setHex()

5.3.2.3 ColorStruct::ColorStruct (std::string *hex*)

A constructor using the hexadecimal representation.

The color's RGB values will be set automatically using this constructor.

Parameters

hex	hexadecimal string (e.g. #FFFFFF)

See Also

setRGB()

5.3.3 Member Function Documentation

5.3.3.1 void ColorStruct::setHex ( )

Using RGB values to set hexadecimal string accordingly.
The result string will have 7 characters (0-9 or A-F) starting with a
pound sign. For instance, (255, 0, 0) -> #FF0000.

5.3.3.2 void ColorStruct::setRGB ( )

Using hexadecimal string to set RGB values accordingly.
The hexadecimal string has to be in the right format, having 6 characters
(0-9 or A-F) following a pound sign. For instance, #FF0000 -> (255, 0, 0).
Unexpected results if the string is not in the right format.

5.3.4 Member Data Documentation

5.3.4.1 int ColorStruct::bValue

Blue component of the color.

5.3.4.2 int ColorStruct::gValue

Green component of the color.

5.3.4.3 std::string ColorStruct::hexStr

Hexadecimal representation.

5.3.4.4 int ColorStruct::rValue

Red component of the color.
The documentation for this struct was generated from the following files:

- ColorStruct.h
- ColorStruct.cpp

5.4 EnvironmentClass Class Reference

A virtual physical environment in which objects interact.

`#include <EnvironmentClass.h>`

Public Member Functions

- EnvironmentClass (int width, int height, int margin=0)

  A normal constructor.
- EnvironmentClass (int width, int height, int top, int bottom, int left, int right)

  A constructor for setting the boundary manually.
- EnvironmentClass (int width, int height, PhysObjList &objLs, int margin=0)

  A constructor with a pre-defined list of operational objects.
- ~EnvironmentClass ()

  Destructor of the EnvironmentClass.
- int getWidth ()

  Environment's width size getter.
- int getHeight ()

  Environment's height size getter.
- int getTopBorder ()

  Environment's top border getter.
- int getBottomBorder ()

  Environment's bottom border getter.
- int getLeftBorder ()

  Environment's left border getter.
- int getRightBorder ()

  Environment's right border getter.
- int getObjectCount ()

  Environment's object list's length getter.
- PhysObjList getObjectList ()

  Environment's object list getter.
- void setBoundary (int top, int bottom, int left, int right)

  Environment's boundary setter.
- ID registerObject (PhysObj obj)

  Function that adds objects to the environment.
- SensorValue touchSensorReading (PhysObj obj)

  Function that provides "Touch" sensor feedback to an object.
- float vectorHoming (PhysObj obj)

  DOCUMENTATION GOES HERE!
- void update (double elapsedTime)

  DOCUMENTATION GOES HERE!

5.4.1 Detailed Description

A virtual physical environment in which objects interact.

The environment knows the configuration of the physical world including the size and boundary of the operational environment, and it provides sensor feedback to the various objects that operate in its environment. [1]

The size will be of a rectangle representing a graphical window, while the boundary will limit the ground where objects (i.e., a robot) can operate.

See Also

SizeStruct
BoundaryStruct
PhysicalObjectClass

5.4.2 Constructor & Destructor Documentation

5.4.2.1 EnvironmentClass::EnvironmentClass (int *width*, int *height*, int *margin* = 0)

A normal constructor.

Set up an environment of rectangular shape. User can determine the margin between the environment's sides and the objects' operational boundary. The default setting is boundary has the same size as environment's. Upon created, environment has no objects.

Parameters

width	width size of the environment
height	height size of the environment
margin	margin between size and boundary (default value is 0)

5.4.2.2 EnvironmentClass::EnvironmentClass (int *width*, int *height*, int *top*, int *bottom*, int *left*, int *right*)

A constructor for setting the boundary manually.

Parameters

width	width size of the environment
height	height size of the environment
top	top border of the environment
bottom	bottom border of the environment
left	left border of the environment
right	right border of the environment

5.4.2.3 EnvironmentClass::EnvironmentClass (int *width*, int *height*, PhysObjList & *objLs*, int *margin* = 0)

A constructor with a pre-defined list of operational objects.

This is useful to create an environment already having objects in it without manually adding one by one. User can still add more objects if needed later.

Parameters

width	width size of the environment
height	height size of the environment
objLs	a list of physical objects
margin	margin between size and boundary (default value is 0)

5.4.2.4 EnvironmentClass::~EnvironmentClass ( )

Destructor of the EnvironmentClass.

Cleaning up the physical object list.

5.4.3 Member Function Documentation

5.4.3.1 int EnvironmentClass::getBottomBorder ( ) [inline]

Environment's bottom border getter.

Returns

bottom border

5.4.3.2 int EnvironmentClass::getHeight ( ) [inline]

Environment's height size getter.

Returns

height size

5.4.3.3 int EnvironmentClass::getLeftBorder ( ) [inline]

Environment's left border getter.

Returns

left border

5.4.3.4 int EnvironmentClass::getObjectCount ( ) [inline]

Environment's object list's length getter.

Returns

how many objects currently in the list

5.4.3.5 PhysObjList EnvironmentClass::getObjectList ( ) [inline]

Environment's object list getter.

Returns

a copy of the object list

5.4.3.6 int EnvironmentClass::getRightBorder ( ) [inline]

Environment's right border getter.

Returns

right border

5.4.3.7 int EnvironmentClass::getTopBorder ( ) [inline]

Environment's top border getter.

Returns

top border

5.4.3.8 int EnvironmentClass::getWidth ( ) [inline]

Environment's width size getter.

Returns

width size

5.4.3.9 ID EnvironmentClass::registerObject (PhysObj *obj*)

Function that adds objects to the environment.
The parameter is actually a pointer to PhysicalObjectClass. Object will not be added if null pointer. Uninitialized objects may result in a core dump.
After objects register with the environment, they will be given an ID number for later use in retrieving information from the list. The object's ID starts from 1; e.g. 1 for first added object, 2 for second added and so on.
Example usage:

```
EnvironmentClass env( 800, 600 );
ID robotID = env.registerObject( new RobotClass() ); // ID is 1
RobotClass *robot = env.getObjectList().at( IDtoIndex( robotID ) ); // index is 0
```

Parameters

obj	a physical object to be added

Returns

the added object's ID number, or zero on failure

See Also

#IDtoIndex(n)

5.4.3.10 void EnvironmentClass::setBoundary (int *top,* int *bottom,* int *left,* int *right*)

Environment's boundary setter.
All parameters have to be non-negative values, and the boundary has to be either overlapping the environment's sides or within the environment. Refer to BoundaryStruct for how the borders are represented.

Parameters

top	top border of the environment
bottom	bottom border of the environment
left	left border of the environment
right	right border of the environment

See Also

BoundaryStruct

5.4.3.11 SensorValue EnvironmentClass::touchSensorReading (PhysObj *obj*)

Function that provides "Touch" sensor feedback to an object.

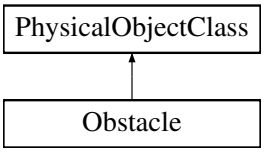
TO BE IMPLEMENTED LATER!

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

- EnvironmentClass.h
- EnvironmentClass.cpp

5.5 Obstacle Class Reference

Inheritance diagram for Obstacle:



Additional Inherited Members

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

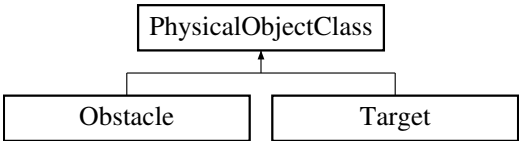
- Obstacle.h
- Obstacle.cpp

5.6 PhysicalObjectClass Class Reference

The representation of a physical object within the simulation.

`#include <PhysicalObjectClass.h>`

Inheritance diagram for PhysicalObjectClass:



Public Member Functions

- PhysicalObjectClass ()
  Default Constructor.
- ~PhysicalObjectClass ()
  Destructor.
- int getPosition ()
  Position getter.
- void **foo** ()
- int getXPosition ()

X-coordinate getter.

- int getYPosition ()

  Y-coordinate getter.

- int getRadius ()

  Radius getter.

- void setPosition (int x, int y)

  XY-coordinate setter.

- void setRadius (int r)

  Radius setter.

Protected Attributes

- type **Type**
- int **xPosition**
- int **yPosition**
- int **radius**
- color **Color**
- int **position**

5.6.1 Detailed Description

The representation of a physical object within the simulation.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 PhysicalObjectClass::~PhysicalObjectClass ( )

Destructor.

(I wonder if we need to have documentation for destructors)

5.6.3 Member Function Documentation

5.6.3.1 int PhysicalObjectClass::getPosition ( ) [inline]

Position getter.

Returns

position

5.6.3.2 int PhysicalObjectClass::getRadius ( ) [inline]

Radius getter.

Returns

radius

5.6.3.3 int PhysicalObjectClass::getXPosition ( ) [inline]

X-coordinate getter.

Returns

x-coordinate

5.6.3.4 int PhysicalObjectClass::getYPosition ( ) [inline]

Y-coordinate getter.

Returns

y-coordinate

5.6.3.5 void PhysicalObjectClass::setPosition (int *x*, int *y*) [inline]

XY-coordinate setter.

Parameters

x	x-coordinate
y	y-coordinate

5.6.3.6 void PhysicalObjectClass::setRadius (int *r*) [inline]

Radius setter.

(assuming object is a circle)

Parameters

r	radius

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

- PhysicalObjectClass.h
- PhysicalObjectClass.cpp

5.7 RobotClass Class Reference

RobotClass.

```
#include <RobotClass.h>
```

Public Member Functions

- RobotClass ()

  RobotClass constructor.
- ~RobotClass ()

  ~RobotClass destructor.
- int getXPosition ()

  getXPosition get x-coordinate.

- int getYPosition ()

  getYPosition get y-coordinate.
- int getRadius ()

  getRadius get radius of robot.
- int getOrientation ()

  getOrientation get the orientation, in degrees, from the horizontal counterclockwise.
- int getSpeed ()

  getSpeed get speed in pps.
- void setPosition (int x, int y)

  setPosition set position of robot.
- void setRadius (int r)

  setRadius set the radius of the robot, which is a circle.
- void setOrientation (int degrees)

  setOrientation set the orientation.
- void setSpeed (int pps)

  setSpeed set the speed.
- bool detectWall ()

  detectWall detect if object has hit a wall.
- bool detectObstacle (PhysicalObjectClass *obstacle)

  detectObstacle detect an obstalce.
- void **rotate** (int degrees)
- int translateX (int distance)

  translateX translate robot a certain distance along X axis.
- int translateY (int distance)

  translateY translate robot a certain distance along Y axis.
- void **updatePosition** (double tbf)
- void pointTo (PhysicalObjectClass *target)

  pointTo point robot to a target.

Protected Attributes

- int **xPosition**
- int **yPosition**
- int **radius**
- int **orientation**
- int **speed**

5.7.1 Detailed Description

RobotClass.

This provides means to store and alter robot state.

5.7.2 Member Function Documentation

5.7.2.1 bool RobotClass::detectObstacle (PhysicalObjectClass * *obstacle*)

detectObstacle detect an obstalce.

Parameters

robot	a robot obstacle.

Returns

bool whether or not a wall is detected

5.7.2.2 int RobotClass::getOrientation ( )

getOrientation get the orientation, in degrees, from the horizontal counterclockwise.

Returns

degrees

5.7.2.3 int RobotClass::getRadius ( )

getRadius get radius of robot.

Returns

radius

5.7.2.4 int RobotClass::getSpeed ( )

getSpeed get speed in pps.

Returns

speed

5.7.2.5 int RobotClass::getXPosition ( )

getXPosition get x-coordinate.

Returns

x-coordinate

5.7.2.6 int RobotClass::getYPosition ( )

getYPosition get y-coordinate.

Returns

y-coordinate

5.7.2.7 void RobotClass::pointTo (PhysicalObjectClass * *target*)

pointTo point robot to a target.

Parameters

target	the target to orient toward.

Returns

new orientation

5.7.2.8 void RobotClass::setOrientation (int *degrees*)

setOrientation set the orientation.

Parameters

degrees	degrees from horizontal, counterclockwise.

5.7.2.9 void RobotClass::setPosition (int *x,* int *y*)

setPosition set position of robot.

Parameters

x	x-coordinate.
y	y-coordinate.

5.7.2.10 void RobotClass::setRadius (int *r*)

setRadius set the radius of the robot, which is a circle.

Parameters

r	radius of robot.

5.7.2.11 void RobotClass::setSpeed (int *pps*)

setSpeed set the speed.

Parameters

pps	pixels per second.

5.7.2.12 int RobotClass::translateX (int *distance*)

translateX translate robot a certain distance along X axis.

Parameters

distance	distance the robot is to be translated on X axis.

Returns

bool whether or not a collision has occurred

5.7.2.13 int RobotClass::translateY (int *distance*)

translateY translate robot a certain distance along Y axis.

Parameters

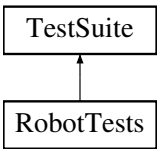
distance	distance the robot is to be translated on Y axis.

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

- RobotClass.h
- RobotClass.cpp

5.8 RobotTests Class Reference

Inheritance diagram for RobotTests:



Public Member Functions

- void **setUp** ()
- void **tearDown** ()
- void testPosition ()

  testPosition test position of robot.
- void testRadius ()

  testRadius test radius of robot
- void testRadiusForNegativeValues ()

  testRadiusForNegativeValue test if radius is proper value
- void testOrientation ()

  testRadiusForNegativeValues see of radius is set to 0 properly
- void **testSpeed** ()
- void **testSpeedForNegativeValues** ()
- void **testDetectWallCollision** ()
- void **testDetectWallNoCollision** ()
- void **testDetectWallEdge** ()
- void **testDetectObstacleCollision** ()
- void **testDetectObstacleNoCollision** ()
- void **testRotateTurnLeft** ()
- void **testRotateTurnRight** ()
- void **testTranslateXHorizontally** ()
- void **testTranslateXVertically** ()
- void **testTranslateXDiagonally** ()
- void **testTranslateYHorizontally** ()
- void **testTranslateYVertically** ()
- void **testTranslateYDiagonally** ()
- void **testUpdatePosition** ()
- void **testPointTo** ()

Public Attributes

- RobotClass * **robot**
- RobotClass * **obstacle**
- RobotClass * **target**

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

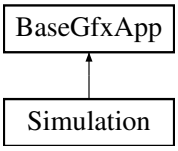
- RobotTests.h

5.9 Simulation Class Reference

The Simulation class.

`#include <Simulation.h>`

Inheritance diagram for Simulation:



Public Types

- enum **UIControlType** { **UI_QUIT** = 0 }

Public Member Functions

- **Simulation** (int argc, char *argv[], int width, int height)
- void **display** ()
- void **addObstacle** (Obstacle *ob)
- void **addTarget** (Target *tg)
- void **gluiControl** (int controlID)
- void **leftMouseDown** (int x, int y)
- void **leftMouseUp** (int x, int y)

Additional Inherited Members

5.9.1 Detailed Description

The Simulation class.

This sets up the GUI and the drawing environment.

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

- Simulation.h
- Simulation.cpp

5.10 SizeStruct Struct Reference

Contains information about different types of size.

`#include <SizeStruct.h>`

Public Member Functions

- SizeStruct ()
  A default constructor.
- SizeStruct (int w, int h)
  A constructor for a rectangle's size.
- SizeStruct (double r)
  A constructor for a circle's size.
- void setRadius ()
  Using width and height to set radius accordingly.

Public Attributes

- int width
  Width of a rectangle.
- int height
  Height of a rectangle.
- double radius
  Radius of a circle.

5.10.1 Detailed Description

Contains information about different types of size.

Can be either width and height of a rectangular object or radius of a circular one. Note that all values are assumed to be non-negative. Value's negativity checking should be handled by the using class or struct.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 SizeStruct::SizeStruct ( ) [inline]

A default constructor.

Set everything to zeros.

5.10.2.2 SizeStruct::SizeStruct (int *w*, int *h*) [inline]

A constructor for a rectangle's size.

Radius is set to zero.

Parameters

w	rectangle's width
h	rectangle's height

5.10.2.3 SizeStruct::SizeStruct (double *r*) [inline]

A constructor for a circle's size.

Width and height are set to circle's diameter. In this case, the circle

is enclosed inside an invisible square with the size of its diameter.

Parameters

r	circle's radius

5.10.3 Member Function Documentation

5.10.3.1 void SizeStruct::setRadius ( )

Using width and height to set radius accordingly.

Width and height must be equal and non-zero. Radius will be half as much.

5.10.4 Member Data Documentation

5.10.4.1 int SizeStruct::height

Height of a rectangle.

5.10.4.2 double SizeStruct::radius

Radius of a circle.

5.10.4.3 int SizeStruct::width

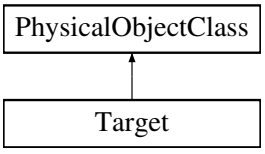
Width of a rectangle.

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

- SizeStruct.h
- SizeStruct.cpp

5.11 Target Class Reference

Inheritance diagram for Target:



Additional Inherited Members

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

- Target.h
- Target.cpp

Chapter 6

File Documentation

6.1 BaseGfxApp.h File Reference

The basic application class for CSci-3081 project. Uses GLUT and GLUI and wraps them in a nice C++ interface.

```
#include <string>
#include <iostream>
#include <assert.h>
#include <GL/glui.h>
```

Classes

- class BaseGfxApp

6.1.1 Detailed Description

The basic application class for CSci-3081 project. Uses GLUT and GLUI and wraps them in a nice C++ interface.

Author

CSci3081 Guru

6.2 BoundaryStruct.cpp File Reference

Implemetation of BoundaryStruct.

```
#include "BoundaryStruct.h"
```

6.2.1 Detailed Description

Implemetation of BoundaryStruct.

Author

Khoi Pham

See Also

BoundaryStruct.h

6.3 BoundaryStruct.h File Reference

A header file contains various a boundary struct to be used in the robot simulation.

```
#include "SizeStruct.h"
```

Classes

- struct BoundaryStruct

  Contains information about a retangular boundary.

6.3.1 Detailed Description

A header file contains various a boundary struct to be used in the robot simulation.

Author

Khoi Pham

See Also

BoundaryStruct.cpp

6.4 ColorStruct.cpp File Reference

Implemetation of ColorStruct.

```
#include "ColorStruct.h"
```

6.4.1 Detailed Description

Implemetation of ColorStruct.

Author

Khoi Pham

See Also

ColorStruct.h

6.5 ColorStruct.h File Reference

A header file contains various a color struct to be used in the robot simulation.

```
#include <string>
```

Classes

- struct ColorStruct

  Contains information about color.

6.5.1 Detailed Description

A header file contains various a color struct to be used in the robot simulation.

Author

Khoi Pham

See Also

ColorStruct.cpp

6.6 EnvironmentClass.cpp File Reference

Implemetation of EnvironmentClass.

```
#include "EnvironmentClass.h"
#include <cmath>
```

Functions

- SensorValue **touchSensorReading** (PhysObj obj)
- void **update** (double elapsedTime)

6.6.1 Detailed Description

Implemetation of EnvironmentClass.

Author

Group 6

See Also

EnvironmentClass.h

6.7 EnvironmentClass.h File Reference

A header file contains class declaration of the class EnvironmentClass.

```
#include "SizeStruct.h"
#include "BoundaryStruct.h"
#include "PhysicalObjectClass.h"
#include <vector>
```

Classes

- class EnvironmentClass

  A virtual physical environment in which objects interact.

Typedefs

- typedef std::vector
  < PhysicalObjectClass * > PhysObjList

  Short-hand for a vector of PhysicalObjectClass pointer.
- typedef PhysicalObjectClass * PhysObj

  Short-hand for a PhysicalObjectClass pointer.
- typedef int ID

  An identification number type.

Enumerations

- enum SensorValue { NOT_ACTIVATED =0, ACTIVATED =1 }

  DOCUMENTATION GOES HERE!

Functions

- int IDtoIndex (ID n)

  Convert an ID type into an integer for use as an array index.
- ID IndextoID (int n)

  Convert an integer for use as an array index into an ID.

6.7.1 Detailed Description

A header file contains class declaration of the class EnvironmentClass.

Author

Group 6

See Also

EnvironmentClass.cpp

6.7.2 Typedef Documentation

6.7.2.1 typedef int ID

An identification number type.

Used as a return type for the registerObject() function. ID starts at 1, while

array's index starts at 0.

See Also

EnvironmentClass::registerObject(PhysObj)
IDtoIndex(ID)
IndextoID(int)

6.7.2.2 typedef std::vector<PhysicalObjectClass*> PhysObjList

Short-hand for a vector of PhysicalObjectClass pointer.

A list of pointers intended use for dynamic binded objects in polymorphism.

6.7.3 Enumeration Type Documentation

6.7.3.1 enum SensorValue

DOCUMENTATION GOES HERE!

Enumerator

NOT_ACTIVATED DOCUMENTATION GOES HERE!

ACTIVATED DOCUMENTATION GOES HERE!

6.7.4 Function Documentation

6.7.4.1 int IDtoIndex (ID *n*) [inline]

Convert an ID type into an integer for use as an array index.

See Also

IndextoID(int)

6.7.4.2 ID IndextoID (int *n*) [inline]

Convert an integer for use as an array index into an ID.

See Also

(ID)

6.8 main.cpp File Reference

Main function.

```
#include "Simulation.h"
```

Functions

- int **main** (int argc, char *argv[])

6.8.1 Detailed Description

Main function.

Author

CSci5107 Guru

6.9 Obstacle.cpp File Reference

The implemetation of the obstacle class.

```
#include "Obstacle.h"
```

6.9.1 Detailed Description

The implemetation of the obstacle class.

Author

Group 6

6.10 Obstacle.h File Reference

The representation of obstacle within the simulation.

```
#include "PhysicalObjectClass.h"
```

Classes

- class Obstacle

6.10.1 Detailed Description

The representation of obstacle within the simulation.

Author

Group 6

6.11 PhysicalObjectClass.cpp File Reference

Implemetation of PhysicalObjectClass.

```
#include "PhysicalObjectClass.h"
```

6.11.1 Detailed Description

Implemetation of PhysicalObjectClass.

Author

Group 6

See Also

PhysicalObjectClass.h

6.12 RobotClass.cpp File Reference

The implemetation of the robot class.

```
#include "RobotClass.h"
#include <cmath>
```

6.12.1 Detailed Description

The implemetation of the robot class.

Author

Group 6

6.13 RobotClass.h File Reference

The representation of robot within the simulation.

```
#include "PhysicalObjectClass.h"
```

Classes

- class RobotClass

  RobotClass.

6.13.1 Detailed Description

The representation of robot within the simulation.

Author

Group 6

6.14 Simulation.cpp File Reference

The implemetation of the robot simulation's class.

```
#include "Simulation.h"
#include <cstdlib>
#include <ctime>
#include <cmath>
```

Macros

- #define **ROBOT_COLOR** 0.0f, 1.0f, 0.0f
- #define **ROBOT_DIRECTION_LINE_COLOR** 1.0f, 0.0f, 0.0f
- #define **OBSTACLE_COLOR** 0.0f, 0.0f, 1.0f
- #define **TARGET_COLOR** 1.0f, 0.0f, 0.0f

Variables

- const int **ROBOT_RADIUS** = 50
- const int **OBSTACLE_RADIUS** = 50
- const int **TARGET_RADIUS** = 50

6.14.1 Detailed Description

The implemetation of the robot simulation's class.

Author

Group 6

6.15 Simulation.h File Reference

Main application class for the robot simulation.

```
#include "BaseGfxApp.h"
#include "RobotClass.h"
#include "Obstacle.h"
#include "Target.h"
#include <vector>
```

Classes

- class Simulation

  The Simulation class.

6.15.1 Detailed Description

Main application class for the robot simulation.

Author

Group 6

6.16 SizeStruct.cpp File Reference

Implemetation of SizeStruct.

```
#include "SizeStruct.h"
```

6.16.1 Detailed Description

Implemetation of SizeStruct.

Author

Khoi Pham

See Also

SizeStruct.h

6.17 SizeStruct.h File Reference

A header file contains a size struct to be used in the robot simulation.

Classes

- struct SizeStruct

  Contains information about different types of size.

6.17.1 Detailed Description

A header file contains a size struct to be used in the robot simulation.

Author

Khoi Pham

See Also

SizeStruct.cpp

6.18 Target.cpp File Reference

The implemetation of the target class.

```
#include "Target.h"
```

6.18.1 Detailed Description

The implemetation of the target class.

Author

Group 6

6.19 Target.h File Reference

The representation of target within the simulation.

```
#include "PhysicalObjectClass.h"
```

Classes

- class Target

6.19.1 Detailed Description

The representation of target within the simulation.

Author

Group 6

Bibliography

[1] Amy Larson. Robot simulation - iteration 2 unit simulation environment, graphics display, physical objects. This is a requirement document for a project in CSci 3081W class., Feb 2015. 16

Index