

## Autonomous Steering Agents

Generated by Doxygen 1.8.17



<b>1 Intent</b>	<b>1</b>
1.1 Dependencies	1
1.2 Resources	1
<b>2 Todo List</b>	<b>3</b>
<b>3 Hierarchical Index</b>	<b>5</b>
3.1 Class Hierarchy	5
<b>4 Class Index</b>	<b>7</b>
4.1 Class List	7
<b>5 File Index</b>	<b>9</b>
5.1 File List	9
<b>6 Class Documentation</b>	<b>11</b>
6.1 agent Class Reference	11
6.1.1 Detailed Description	12
6.1.2 Constructor & Destructor Documentation	13
6.1.2.1 agent() [1/2]	13
6.1.2.2 agent() [2/2]	13
6.1.2.3 ~agent()	14
6.1.3 Member Function Documentation	14
6.1.3.1 setFeatures()	14
6.1.3.2 updatePosition()	14
6.1.4 Member Data Documentation	15
6.1.4.1 acceleration	15
6.1.4.2 arrive	15
6.1.4.3 desiredVelocity	16
6.1.4.4 fillColor	16
6.1.4.5 force	16
6.1.4.6 id	16
6.1.4.7 mass	16
6.1.4.8 maxForce	17
6.1.4.9 maxSpeed	17
6.1.4.10 name	17
6.1.4.11 position	17
6.1.4.12 r	17
6.1.4.13 steering	18
6.1.4.14 targetPoint	18
6.1.4.15 velocity	18
6.2 color Class Reference	18
6.2.1 Detailed Description	20
6.2.2 Constructor & Destructor Documentation	20

6.2.2.1 color() [1/2]	20
6.2.2.2 color() [2/2]	20
6.2.3 Member Function Documentation	21
6.2.3.1 createColors()	21
6.2.3.2 getColor()	21
6.2.4 Member Data Documentation	22
6.2.4.1 B	22
6.2.4.2 colors	22
6.2.4.3 G	22
6.2.4.4 R	22
6.3 evade Class Reference	23
6.3.1 Detailed Description	25
6.3.2 Constructor & Destructor Documentation	25
6.3.2.1 evade()	25
6.3.3 Member Function Documentation	25
6.3.3.1 loop()	25
6.4 flee Class Reference	26
6.4.1 Detailed Description	28
6.4.2 Constructor & Destructor Documentation	28
6.4.2.1 flee()	28
6.4.3 Member Function Documentation	28
6.4.3.1 loop()	28
6.5 flock Class Reference	29
6.5.1 Detailed Description	31
6.5.2 Constructor & Destructor Documentation	31
6.5.2.1 flock()	31
6.5.3 Member Function Documentation	31
6.5.3.1 loop()	31
6.6 flowField Class Reference	32
6.6.1 Detailed Description	33
6.6.2 Constructor & Destructor Documentation	33
6.6.2.1 flowField() [1/2]	33
6.6.2.2 flowField() [2/2]	33
6.6.3 Member Function Documentation	34
6.6.3.1 getField()	34
6.7 graphics Class Reference	35
6.7.1 Detailed Description	36
6.7.2 Member Function Documentation	36
6.7.2.1 drawAgent()	36
6.7.2.2 drawCircle()	37
6.7.2.3 drawLine()	37
6.7.2.4 drawPath()	38

6.7.2.5 drawPoint()	39
6.7.2.6 drawText()	39
6.7.2.7 forceInScreen()	40
6.7.2.8 getMousePosition()	40
6.7.2.9 handleKeypress()	41
6.7.2.10 handleResize()	42
6.7.2.11 initGraphics()	42
6.7.2.12 mouseButton()	43
6.7.2.13 mouseMove()	44
6.7.2.14 refreshScene()	45
6.7.2.15 timerEvent()	45
6.7.3 Member Data Documentation	45
6.7.3.1 target_x	45
6.7.3.2 target_y	46
6.8 mouseFollower Class Reference	46
6.8.1 Detailed Description	48
6.8.2 Constructor & Destructor Documentation	48
6.8.2.1 mouseFollower()	48
6.8.3 Member Function Documentation	48
6.8.3.1 loop()	48
6.9 obstacle Class Reference	49
6.9.1 Detailed Description	50
6.9.2 Constructor & Destructor Documentation	50
6.9.2.1 obstacle() [1/2]	50
6.9.2.2 obstacle() [2/2]	50
6.9.3 Member Data Documentation	50
6.9.3.1 p	51
6.9.3.2 r	51
6.10 obstacleAvoidance Class Reference	51
6.10.1 Detailed Description	53
6.10.2 Constructor & Destructor Documentation	53
6.10.2.1 obstacleAvoidance()	53
6.10.3 Member Function Documentation	53
6.10.3.1 createObstacle()	53
6.10.3.2 loop()	54
6.10.4 Member Data Documentation	55
6.10.4.1 obstacles	55
6.11 path Class Reference	55
6.11.1 Detailed Description	57
6.11.2 Constructor & Destructor Documentation	57
6.11.2.1 path() [1/2]	57
6.11.2.2 path() [2/2]	57

6.11.3 Member Function Documentation	58
6.11.3.1 addPoint()	58
6.11.4 Member Data Documentation	58
6.11.4.1 points	58
6.11.4.2 width	59
6.12 pathFollower Class Reference	59
6.12.1 Detailed Description	61
6.12.2 Constructor & Destructor Documentation	61
6.12.2.1 pathFollower()	61
6.12.3 Member Function Documentation	61
6.12.3.1 createPath()	61
6.12.3.2 loop()	62
6.12.4 Member Data Documentation	63
6.12.4.1 myPath	63
6.13 point Class Reference	63
6.13.1 Detailed Description	64
6.13.2 Constructor & Destructor Documentation	64
6.13.2.1 point() [1/2]	65
6.13.2.2 point() [2/2]	66
6.13.3 Member Function Documentation	66
6.13.3.1 div()	66
6.13.3.2 getNormalPoint()	67
6.13.3.3 mul()	68
6.13.3.4 operator+() [1/2]	69
6.13.3.5 operator+() [2/2]	69
6.13.3.6 operator-()	70
6.13.3.7 operator==()	70
6.13.3.8 print()	71
6.13.4 Member Data Documentation	71
6.13.4.1 x	71
6.13.4.2 y	71
6.14 prison Class Reference	72
6.14.1 Detailed Description	74
6.14.2 Constructor & Destructor Documentation	74
6.14.2.1 prison()	74
6.14.3 Member Function Documentation	74
6.14.3.1 loop()	74
6.15 pursuit Class Reference	75
6.15.1 Detailed Description	78
6.15.2 Constructor & Destructor Documentation	78
6.15.2.1 pursuit()	78
6.15.3 Member Function Documentation	78

6.15.3.1 loop()	78
6.16 pvector Class Reference	79
6.16.1 Detailed Description	80
6.16.2 Constructor & Destructor Documentation	80
6.16.2.1 pvector() [1/2]	80
6.16.2.2 pvector() [2/2]	80
6.16.3 Member Function Documentation	81
6.16.3.1 add()	81
6.16.3.2 angleBetween()	81
6.16.3.3 div()	82
6.16.3.4 dotProduct()	83
6.16.3.5 getAngle()	84
6.16.3.6 limit()	84
6.16.3.7 magnitude()	85
6.16.3.8 mul()	86
6.16.3.9 normalize()	87
6.16.3.10 operator+() [1/2]	88
6.16.3.11 operator+() [2/2]	88
6.16.3.12 operator+=()	89
6.16.3.13 operator-() [1/2]	89
6.16.3.14 operator-() [2/2]	90
6.16.3.15 operator==()	90
6.16.3.16 print()	91
6.16.4 Member Data Documentation	91
6.16.4.1 x	91
6.16.4.2 y	91
6.17 random Class Reference	92
6.17.1 Detailed Description	92
6.17.2 Member Function Documentation	92
6.17.2.1 createRandomArray()	92
6.18 scenario Class Reference	93
6.18.1 Detailed Description	95
6.18.2 Constructor & Destructor Documentation	95
6.18.2.1 scenario()	95
6.18.3 Member Function Documentation	95
6.18.3.1 createAgent()	95
6.18.3.2 initGL()	96
6.18.3.3 refresh()	97
6.18.4 Member Data Documentation	97
6.18.4.1 agents	97
6.18.4.2 behavior	98
6.18.4.3 callback	98

6.18.4.4 myColor . . . . .	98
6.18.4.5 name . . . . .	98
6.18.4.6 view . . . . .	99
6.19 steeringBehavior Class Reference . . . . .	99
6.19.1 Detailed Description . . . . .	100
6.19.2 Member Function Documentation . . . . .	100
6.19.2.1 align() . . . . .	100
6.19.2.2 avoid() . . . . .	101
6.19.2.3 cohesion() . . . . .	102
6.19.2.4 evade() . . . . .	103
6.19.2.5 flee() . . . . .	104
6.19.2.6 inFlowField() . . . . .	105
6.19.2.7 pursuit() . . . . .	106
6.19.2.8 seek() . . . . .	107
6.19.2.9 separation() . . . . .	107
6.19.2.10 setAngle() . . . . .	109
6.19.2.11 stayInArea() . . . . .	110
6.19.2.12 stayInPath() . . . . .	110
6.19.2.13 wander() . . . . .	111
6.20 wander Class Reference . . . . .	112
6.20.1 Detailed Description . . . . .	115
6.20.2 Constructor & Destructor Documentation . . . . .	115
6.20.2.1 wander() . . . . .	115
6.20.3 Member Function Documentation . . . . .	115
6.20.3.1 loop() . . . . .	115
6.21 windy Class Reference . . . . .	116
6.21.1 Detailed Description . . . . .	118
6.21.2 Constructor & Destructor Documentation . . . . .	118
6.21.2.1 windy() . . . . .	118
6.21.3 Member Function Documentation . . . . .	118
6.21.3.1 loop() . . . . .	118
6.21.4 Member Data Documentation . . . . .	119
6.21.4.1 flow . . . . .	119
<b>7 File Documentation . . . . .</b>	<b>121</b>
7.1 include/agent.h File Reference . . . . .	121
7.1.1 Detailed Description . . . . .	122
7.2 include/color.h File Reference . . . . .	122
7.2.1 Detailed Description . . . . .	123
7.2.2 Enumeration Type Documentation . . . . .	123
7.2.2.1 num . . . . .	123
7.3 include/evade.h File Reference . . . . .	124



7.3.1 Detailed Description . . . . .	125
7.4 include/flee.h File Reference . . . . .	125
7.4.1 Detailed Description . . . . .	126
7.5 include/flock.h File Reference . . . . .	126
7.5.1 Detailed Description . . . . .	128
7.6 include/flowField.h File Reference . . . . .	128
7.6.1 Detailed Description . . . . .	129
7.6.2 Macro Definition Documentation . . . . .	129
7.6.2.1 FIELD_HEIGHT . . . . .	129
7.6.2.2 FIELD_WIDTH . . . . .	129
7.6.2.3 GRAVITY . . . . .	129
7.6.2.4 WIND_WEST . . . . .	130
7.7 include/graphics.h File Reference . . . . .	130
7.7.1 Detailed Description . . . . .	131
7.7.2 Macro Definition Documentation . . . . .	131
7.7.2.1 ESC . . . . .	131
7.7.2.2 HEIGHT . . . . .	131
7.7.2.3 PI . . . . .	131
7.7.2.4 WIDTH . . . . .	132
7.8 include/mouseFollower.h File Reference . . . . .	132
7.8.1 Detailed Description . . . . .	133
7.9 include/obstacle.h File Reference . . . . .	133
7.9.1 Detailed Description . . . . .	134
7.10 include/obstacleAvoidance.h File Reference . . . . .	135
7.10.1 Detailed Description . . . . .	136
7.11 include/path.h File Reference . . . . .	136
7.11.1 Detailed Description . . . . .	137
7.12 include/pathFollower.h File Reference . . . . .	137
7.12.1 Detailed Description . . . . .	138
7.13 include/point.h File Reference . . . . .	138
7.13.1 Detailed Description . . . . .	139
7.14 include/prison.h File Reference . . . . .	140
7.14.1 Detailed Description . . . . .	141
7.15 include/pursuit.h File Reference . . . . .	141
7.15.1 Detailed Description . . . . .	142
7.16 include/pvector.h File Reference . . . . .	142
7.16.1 Detailed Description . . . . .	143
7.16.2 Macro Definition Documentation . . . . .	143
7.16.2.1 PI . . . . .	143
7.17 include/random.h File Reference . . . . .	144
7.17.1 Detailed Description . . . . .	144
7.18 include/scenario.h File Reference . . . . .	144

7.18.1 Detailed Description	145
7.18.2 Enumeration Type Documentation	146
7.18.2.1 types	146
7.19 include/steeringBehavior.h File Reference	146
7.19.1 Detailed Description	148
7.19.2 Macro Definition Documentation	148
7.19.2.1 AVOID_OBSTACLE	148
7.19.2.2 CIRCLE_DISTANCE	148
7.19.2.3 CIRCLE_RADIUS	148
7.19.2.4 EVADE	148
7.19.2.5 FLEE	149
7.19.2.6 FLOCK	149
7.19.2.7 FOLLOW_MOUSE	149
7.19.2.8 IN_FLOW_FIELD	149
7.19.2.9 PURSUIT	149
7.19.2.10 STAY_IN_FIELD	149
7.19.2.11 STAY_IN_PATH	150
7.19.2.12 WANDER	150
7.20 include/wander.h File Reference	150
7.20.1 Detailed Description	151
7.21 include/windy.h File Reference	151
7.21.1 Detailed Description	153
7.22 main.cpp File Reference	153
7.22.1 Detailed Description	154
7.22.2 Function Documentation	154
7.22.2.1 main()	154
7.22.2.2 menu()	155
7.22.3 Variable Documentation	155
7.22.3.1 mode	156
7.23 README.md File Reference	156
7.24 src/agent.cpp File Reference	156
7.24.1 Detailed Description	157
7.25 src/color.cpp File Reference	157
7.25.1 Detailed Description	157
7.26 src/evade.cpp File Reference	158
7.26.1 Detailed Description	158
7.27 src/flee.cpp File Reference	159
7.27.1 Detailed Description	159
7.28 src/flock.cpp File Reference	160
7.28.1 Detailed Description	160
7.29 src/flowField.cpp File Reference	161
7.29.1 Detailed Description	161

7.30 src/graphics.cpp File Reference . . . . .	161
7.30.1 Detailed Description . . . . .	162
7.31 src/mouseFollower.cpp File Reference . . . . .	162
7.31.1 Detailed Description . . . . .	163
7.32 src/obstacle.cpp File Reference . . . . .	163
7.32.1 Detailed Description . . . . .	164
7.33 src/obstacleAvoidance.cpp File Reference . . . . .	164
7.33.1 Detailed Description . . . . .	165
7.34 src/path.cpp File Reference . . . . .	165
7.34.1 Detailed Description . . . . .	166
7.35 src/pathFollower.cpp File Reference . . . . .	166
7.35.1 Detailed Description . . . . .	167
7.36 src/point.cpp File Reference . . . . .	168
7.36.1 Detailed Description . . . . .	168
7.37 src/prison.cpp File Reference . . . . .	168
7.37.1 Detailed Description . . . . .	169
7.37.2 Macro Definition Documentation . . . . .	170
7.37.2.1 DISTANCE . . . . .	170
7.37.2.2 WALL . . . . .	170
7.38 src/pursuit.cpp File Reference . . . . .	170
7.38.1 Detailed Description . . . . .	171
7.39 src/pvector.cpp File Reference . . . . .	172
7.39.1 Detailed Description . . . . .	172
7.40 src/random.cpp File Reference . . . . .	173
7.40.1 Detailed Description . . . . .	173
7.41 src/scenario.cpp File Reference . . . . .	173
7.41.1 Detailed Description . . . . .	174
7.41.2 Macro Definition Documentation . . . . .	174
7.41.2.1 MAX_NUMBER_OF_AGENTS . . . . .	175
7.42 src/steeringBehavior.cpp File Reference . . . . .	175
7.42.1 Detailed Description . . . . .	175
7.43 src/wander.cpp File Reference . . . . .	176
7.43.1 Detailed Description . . . . .	176
7.44 src/windy.cpp File Reference . . . . .	177
7.44.1 Detailed Description . . . . .	177
7.45 test/test_suites.cpp File Reference . . . . .	178
7.45.1 Detailed Description . . . . .	179
7.45.2 Macro Definition Documentation . . . . .	179
7.45.2.1 BOOST_TEST_MODULE . . . . .	179
7.45.3 Function Documentation . . . . .	179
7.45.3.1 BOOST_AUTO_TEST_CASE() [1/12] . . . . .	180
7.45.3.2 BOOST_AUTO_TEST_CASE() [2/12] . . . . .	180

7.45.3.3 BOOST_AUTO_TEST_CASE() [3/12]	181
7.45.3.4 BOOST_AUTO_TEST_CASE() [4/12]	181
7.45.3.5 BOOST_AUTO_TEST_CASE() [5/12]	182
7.45.3.6 BOOST_AUTO_TEST_CASE() [6/12]	182
7.45.3.7 BOOST_AUTO_TEST_CASE() [7/12]	183
7.45.3.8 BOOST_AUTO_TEST_CASE() [8/12]	183
7.45.3.9 BOOST_AUTO_TEST_CASE() [9/12]	184
7.45.3.10 BOOST_AUTO_TEST_CASE() [10/12]	184
7.45.3.11 BOOST_AUTO_TEST_CASE() [11/12]	185
7.45.3.12 BOOST_AUTO_TEST_CASE() [12/12]	185

<b>Index</b>	<b>187</b>
--------------	------------

# Chapter 1

## Intent

- 1- implementing Craig Reynolds autonomous steering agents
- 2- implementing genetics algorithms
- 3- implementing neural network

### 1.1 Dependencies

```
$sudo apt-get install libglu1-mesa-dev freeglut3-dev mesa-common-dev
```

```
$sudo apt-get install libboost-all-dev
```

### 1.2 Resources

<https://natureofcode.com/book/chapter-6-autonomous-agents>

<https://gamedevelopment.tutsplus.com/series/understanding-steering-behaviors-gamedev-12>

<https://videotutorialsrock.com/index.php>

<https://www.opengl.org/resources/libraries/glut/spec3/node1.html>

<https://learnopengl.com/Getting-started/Coordinate-Systems>



## Chapter 2

# Todo List

Member `wander::wander ()`

business logic will be changed





## Chapter 3

# Hierarchical Index

### 3.1 Class Hierarchy

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

agent . . . . .	11
color . . . . .	18
flowField . . . . .	32
graphics . . . . .	35
obstacle . . . . .	49
path . . . . .	55
point . . . . .	63
pvector . . . . .	79
random . . . . .	92
scenario . . . . .	93
evade . . . . .	23
flee . . . . .	26
flock . . . . .	29
mouseFollower . . . . .	46
obstacleAvoidance . . . . .	51
pathFollower . . . . .	59
prison . . . . .	72
pursuit . . . . .	75
wander . . . . .	112
windy . . . . .	116
steeringBehavior . . . . .	99



## Chapter 4

# Class Index

### 4.1 Class List

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

agent	11
color	18
evade	23
flee	26
flock	29
flowField	32
graphics	35
mouseFollower	46
obstacle	49
obstacleAvoidance	51
path	55
pathFollower	59
point	63
prison	72
pursuit	75
pvector	79
random	92
scenario	93
steeringBehavior	99
wander	112
windy	116



## Chapter 5

# File Index

### 5.1 File List

Here is a list of all files with brief descriptions:

<a href="#">main.cpp</a>	Client code . . . . .	153
<a href="#">include/agent.h</a>	Agent class defines all agent specifications . . . . .	121
<a href="#">include/color.h</a>	Color class used for agent, path, wall etc. color . . . . .	122
<a href="#">include/evade.h</a>	Evade class inherited from scenario class . . . . .	124
<a href="#">include/flee.h</a>	Agents flee from mouse scenario . . . . .	125
<a href="#">include/flock.h</a>	Flocking agents scenario . . . . .	126
<a href="#">include/flowField.h</a>	FlowField class, screen can be filled with a force for each pixel . . . . .	128
<a href="#">include/graphics.h</a>	Graphics class, drives openGL . . . . .	130
<a href="#">include/mouseFollower.h</a>	Agents follow mouse scenario . . . . .	132
<a href="#">include/obstacle.h</a>	Circular obstacles for agent avoidance behaviors . . . . .	133
<a href="#">include/obstacleAvoidance.h</a>	Agents avoid from obstacles scenario . . . . .	135
<a href="#">include/path.h</a>	Path class used for path following steering behaviors . . . . .	136
<a href="#">include/pathFollower.h</a>	Path following scenario . . . . .	137
<a href="#">include/point.h</a>	Point class used for point operations . . . . .	138
<a href="#">include/prison.h</a>	Agents cant escape from field scenario . . . . .	140
<a href="#">include/pursuit.h</a>	One agent pursue other one scenario . . . . .	141
<a href="#">include/pvector.h</a>	Pvector class used for 2D vector operations . . . . .	142
<a href="#">include/random.h</a>	Utility class for random operations . . . . .	144

include/ <a href="#">scenario.h</a>	Base class for all scenarios . . . . .	144
include/ <a href="#">steeringBehavior.h</a>	Functions for autonomous steering behaviors . . . . .	146
include/ <a href="#">wander.h</a>	Random wandering agents scenario . . . . .	150
include/ <a href="#">windy.h</a>	Windy air scenario . . . . .	151
src/ <a href="#">agent.cpp</a>	Implementation of the agent class . . . . .	156
src/ <a href="#">color.cpp</a>	Color class implementation . . . . .	157
src/ <a href="#">evade.cpp</a>	Evade class implementation . . . . .	158
src/ <a href="#">flee.cpp</a>	Flee class implementation . . . . .	159
src/ <a href="#">flock.cpp</a>	Flock class implementation . . . . .	160
src/ <a href="#">flowField.cpp</a>	FlowField class implementation . . . . .	161
src/ <a href="#">graphics.cpp</a>	Graphics class implementation . . . . .	161
src/ <a href="#">mouseFollower.cpp</a>	MouseFollower class implementation . . . . .	162
src/ <a href="#">obstacle.cpp</a>	Obstacle class implementation . . . . .	163
src/ <a href="#">obstacleAvoidance.cpp</a>	ObstacleAvoidance class implementation . . . . .	164
src/ <a href="#">path.cpp</a>	Path class implementation . . . . .	165
src/ <a href="#">pathFollower.cpp</a>	PathFollower class implementation . . . . .	166
src/ <a href="#">point.cpp</a>	Point class implementation file . . . . .	168
src/ <a href="#">prison.cpp</a>	Prison class implementation . . . . .	168
src/ <a href="#">pursuit.cpp</a>	Prison class implementation . . . . .	170
src/ <a href="#">pvector.cpp</a>	Pvector class implementation . . . . .	172
src/ <a href="#">random.cpp</a>	Utility class for random operations . . . . .	173
src/ <a href="#">scenario.cpp</a>	Scenario base class implementation . . . . .	173
src/ <a href="#">steeringBehavior.cpp</a>	Implementation of autonomous steering behaviors . . . . .	175
src/ <a href="#">wander.cpp</a>	Wander class implementation . . . . .	176
src/ <a href="#">windy.cpp</a>	Windy class implementation . . . . .	177
test/ <a href="#">test_suites.cpp</a>	Unit test suites . . . . .	178

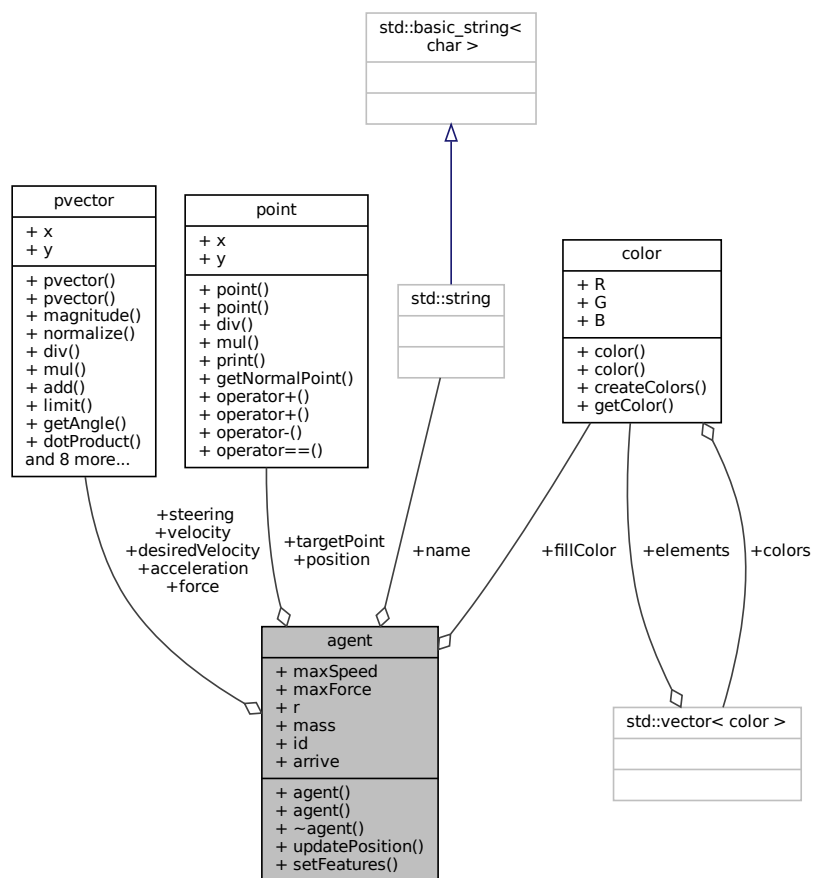
## Chapter 6

# Class Documentation

### 6.1 agent Class Reference

```
#include <agent.h>
```

Collaboration diagram for agent:



## Public Member Functions

- [agent](#) ()  
*default constructor.*
- [agent](#) (float x, float y)  
*constructor.*
- [~agent](#) ()  
*destructor*
- void [updatePosition](#) (bool [arrive](#))  
*position update calculations*
- void [setFeatures](#) (float s, float f, float [r](#), float m)  
*initialize the agent attributes*

## Public Attributes

- string [name](#)  
*name of the agent*
- color [fillColor](#)  
*color of the agent*
- point [position](#)  
*position of the agent*
- pvector [velocity](#)  
*velocity of the agent*
- point [targetPoint](#)  
*target of the agent*
- float [maxSpeed](#)  
*maximum speed of the agent*
- float [maxForce](#)  
*maximum force of the agent*
- pvector [steering](#)  
*steering force of the apply*
- pvector [force](#)  
*force of the agent*
- pvector [acceleration](#)  
*acceleration of the agent*
- pvector [desiredVelocity](#)  
*desired velocity of the agent*
- float [r](#)  
*radius of the agent*
- float [mass](#)  
*mass of the agent*
- int [id](#)  
*id of the agent*
- bool [arrive](#) = false  
*has arriving behavior or not*

### 6.1.1 Detailed Description

Definition at line 20 of file agent.h.



## 6.1.2 Constructor & Destructor Documentation

### 6.1.2.1 agent() [1/2]

```
agent::agent ( )
```

default constructor.

See also

[agent\(float x, float y\)](#)

Definition at line 16 of file agent.cpp.

```
17 {  
18  
19 }
```

### 6.1.2.2 agent() [2/2]

```
agent::agent (  
    float x,  
    float y )
```

constructor.

Parameters

<i>x</i>	position x of the agent
<i>y</i>	position y of the agent

See also

[agent\(\)](#)

Definition at line 21 of file agent.cpp.

```
22 {  
23     position      = point(x, y);  
24     velocity      = pvector(0.6, 0.0);  
25     acceleration  = pvector(0.0, 0.0);  
26     steering      = pvector(0.0, 0.0);  
27     desiredVelocity = pvector(0.0, 0.0);  
28     force         = pvector(0.0, 0.0);  
29     targetPoint   = point(0.0, 0.0);  
30     fillColor     = color(1.0, 0.0, 0.0);  
31 }
```

### 6.1.2.3 ~agent()

```
agent::~~agent ( )
```

destructor

Definition at line 62 of file agent.cpp.

```
63 {
64
65 }
```

## 6.1.3 Member Function Documentation

### 6.1.3.1 setFeatures()

```
void agent::setFeatures (
    float s,
    float f,
    float r,
    float m )
```

initialize the agent attributes

Parameters

<i>s</i>	maximum velocity
<i>f</i>	maximum force
<i>r</i>	radius for arriving behavior
<i>m</i>	mass

Definition at line 54 of file agent.cpp.

```
55 {
56     this->maxSpeed = s;
57     this->maxForce = f;
58     this->r = r;
59     this->mass = m;
60 }
```

### 6.1.3.2 updatePosition()

```
void agent::updatePosition (
    bool arrive )
```

position update calculations

Parameters

<i>arrive</i>	has arriving behavior or not
---------------	------------------------------

See also

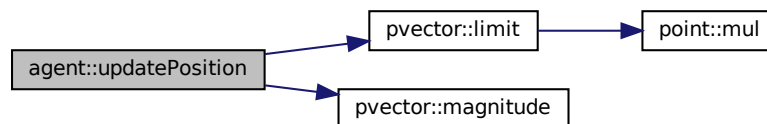
[agent\(\)](#)

Definition at line 33 of file agent.cpp.

```

34 {
35     force.limit(maxForce);
36     acceleration = force;
37     velocity += acceleration;
38
39     //arriving behavior implementation
40     if(arrive == true){
41         pvector diff = targetPoint - position;
42         if(diff.magnitude() > r)
43             velocity.limit(maxSpeed);
44         else
45             velocity.limit(maxSpeed * diff.magnitude() / r);
46     }
47     else
48         velocity.limit(maxSpeed);
49
50     position = position + velocity;
51     force = pvector(0,0);
52 }
```

Here is the call graph for this function:



## 6.1.4 Member Data Documentation

### 6.1.4.1 acceleration

`pvector` `agent::acceleration`

acceleration of the agent

Definition at line 105 of file agent.h.

### 6.1.4.2 arrive

`bool` `agent::arrive = false`

has arriving behavior or not

Definition at line 130 of file agent.h.

#### 6.1.4.3 desiredVelocity

`pvector agent::desiredVelocity`

desired velocity of the agent

Definition at line 110 of file agent.h.

#### 6.1.4.4 fillColor

`color agent::fillColor`

color of the agent

Definition at line 65 of file agent.h.

#### 6.1.4.5 force

`pvector agent::force`

force of the agent

Definition at line 100 of file agent.h.

#### 6.1.4.6 id

`int agent::id`

id of the agent

Definition at line 125 of file agent.h.

#### 6.1.4.7 mass

`float agent::mass`

mass of the agent

Definition at line 120 of file agent.h.

#### 6.1.4.8 maxForce

```
float agent::maxForce
```

maximum force of the agent

Definition at line 90 of file agent.h.

#### 6.1.4.9 maxSpeed

```
float agent::maxSpeed
```

maximum speed of the agent

Definition at line 85 of file agent.h.

#### 6.1.4.10 name

```
string agent::name
```

name of the agent

Definition at line 60 of file agent.h.

#### 6.1.4.11 position

```
point agent::position
```

position of the agent

Definition at line 70 of file agent.h.

#### 6.1.4.12 r

```
float agent::r
```

radius of the agent

Definition at line 115 of file agent.h.

#### 6.1.4.13 steering

`pvector` `agent::steering`

steering force of the apply

Definition at line 95 of file agent.h.

#### 6.1.4.14 targetPoint

`point` `agent::targetPoint`

target of the agent

Definition at line 80 of file agent.h.

#### 6.1.4.15 velocity

`pvector` `agent::velocity`

velocity of the agent

Definition at line 75 of file agent.h.

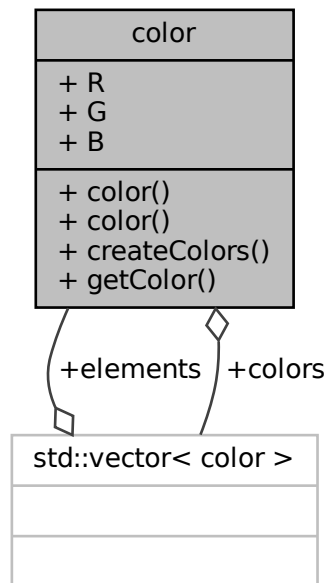
The documentation for this class was generated from the following files:

- `include/agent.h`
- `src/agent.cpp`

## 6.2 color Class Reference

```
#include <color.h>
```

Collaboration diagram for color:



## Public Member Functions

- `color()`  
*default constructor.*
- `color(float r, float g, float b)`  
*constructor.*
- `void createColors()`  
*creation of fundamental 8 colors*
- `color getColor(int index)`  
*gets requested color*

## Public Attributes

- `float R`  
*portion of red color*
- `float G`  
*portion of green color*
- `float B`  
*portion of blue color*
- `vector<color> colors`  
*storage structure of created fundamental colors*

## 6.2.1 Detailed Description

Definition at line 19 of file color.h.

## 6.2.2 Constructor & Destructor Documentation

### 6.2.2.1 color() [1/2]

```
color::color ( )
```

default constructor.

See also

[color\(float r, float g, float b\)](#)

Definition at line 25 of file color.cpp.

```
26 {  
27  
28 }
```

### 6.2.2.2 color() [2/2]

```
color::color (  
    float r,  
    float g,  
    float b )
```

constructor.

Parameters

<i>r</i>	red (0-255)
<i>g</i>	green (0-255)
<i>b</i>	blue (0-255)

See also

[path\(\)](#)

Definition at line 13 of file color.cpp.

```
14 {  
15     R = r;  
16     G = g;  
17     B = b;  
18 }
```



## 6.2.3 Member Function Documentation

### 6.2.3.1 createColors()

```
void color::createColors ( )
```

creation of fundamental 8 colors

Definition at line 30 of file color.cpp.

```
31 {  
32     colors.push_back(color(0.0, 0.0, 0.0));  
33     colors.push_back(color(0.0, 0.0, 1.0));  
34     colors.push_back(color(0.0, 1.0, 0.0));  
35     colors.push_back(color(0.0, 1.0, 1.0));  
36     colors.push_back(color(1.0, 0.0, 0.0));  
37     colors.push_back(color(1.0, 0.0, 1.0));  
38     colors.push_back(color(1.0, 1.0, 0.0));  
39     colors.push_back(color(1.0, 1.0, 1.0));  
40 }
```

### 6.2.3.2 getColor()

```
color color::getColor (  
    int index )
```

gets requested color

#### Parameters

<i>i</i>	color index
----------	-------------

#### Returns

requested color

Definition at line 20 of file color.cpp.

```
21 {  
22     return colors.at(index);  
23 }
```

Here is the caller graph for this function:



## 6.2.4 Member Data Documentation

### 6.2.4.1 B

```
float color::B
```

portion of blue color

Definition at line 61 of file color.h.

### 6.2.4.2 colors

```
vector<color> color::colors
```

storage structure of created fundamental colors

Definition at line 66 of file color.h.

### 6.2.4.3 G

```
float color::G
```

portion of green color

Definition at line 56 of file color.h.

### 6.2.4.4 R

```
float color::R
```

portion of red color

Definition at line 51 of file color.h.

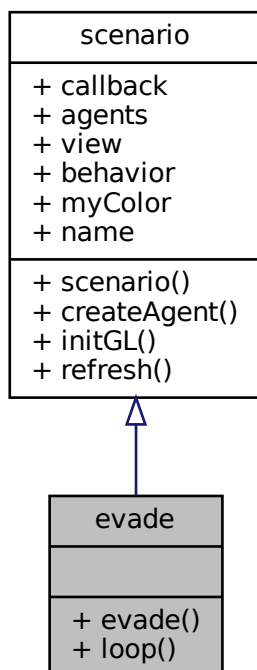
The documentation for this class was generated from the following files:

- include/[color.h](#)
- src/[color.cpp](#)

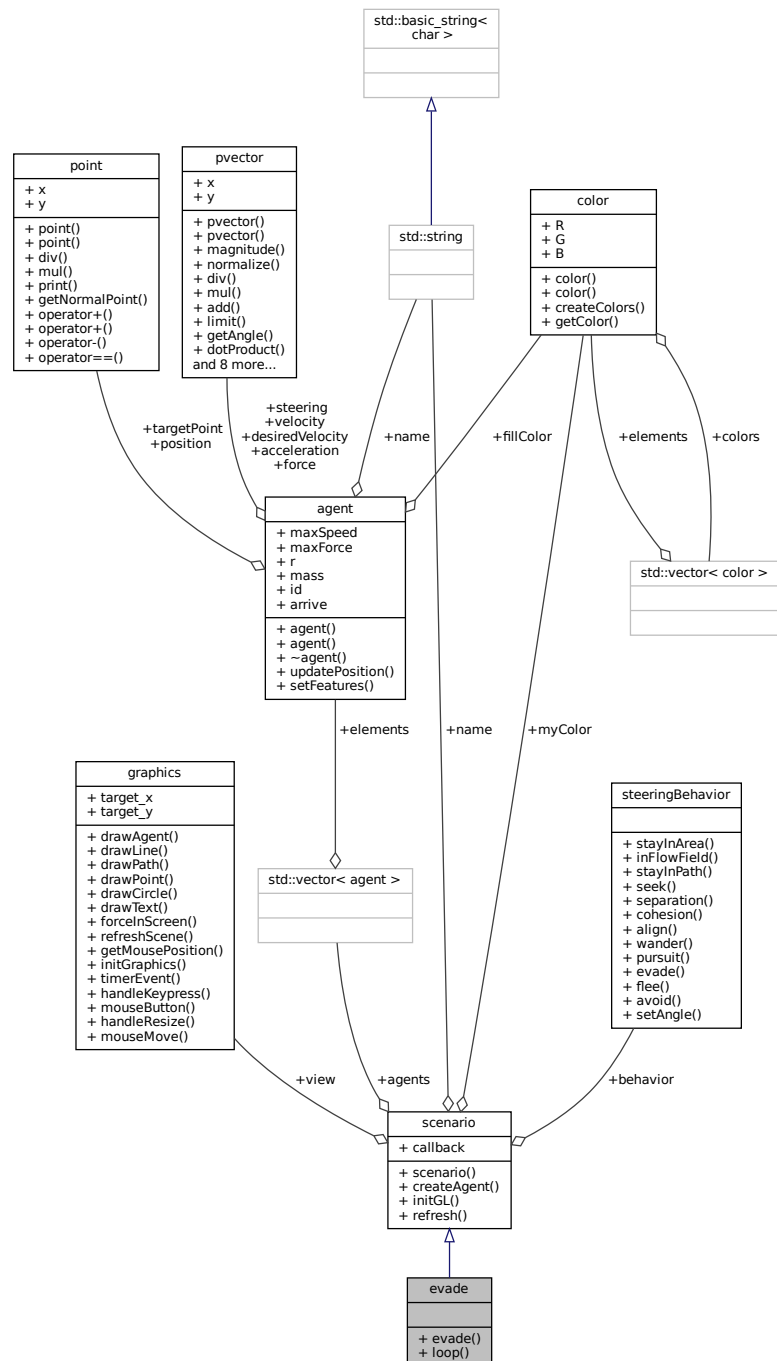
## 6.3 evade Class Reference

```
#include <evade.h>
```

Inheritance diagram for evade:



Collaboration diagram for evade:



## Public Member Functions

- [evade](#) ()

*default constructor.*

## Static Public Member Functions

- static void `loop` ()  
*loop function of evading scenario*

## Additional Inherited Members

### 6.3.1 Detailed Description

Definition at line 15 of file `evade.h`.

### 6.3.2 Constructor & Destructor Documentation

#### 6.3.2.1 `evade()`

`evade::evade ( )`

default constructor.

Definition at line 31 of file `evade.cpp`.

```
32 {
33     name = "evading";
34     createAgent(STATIC, nullptr, nullptr, nullptr);
35     callback = reinterpret_cast <void(*)()> ( (void *)(&loop) );
36 }
```

### 6.3.3 Member Function Documentation

#### 6.3.3.1 `loop()`

`void evade::loop ( )` [static]

loop function of evading scenario

#### Note

opengl callback forces that function to be static

Definition at line 15 of file `evade.cpp`.

```
16 {
17     for(auto it = agents.begin(); it < agents.end(); it++){
18         if((*it).name == "lion"){
19             (*it).targetPoint = view.getMousePosition();
20             (*it).force = behavior.seek(*it);
21             (*it).arrive = true;
22         }
23         else{//gazelle
24             (*it).force = behavior.evade(agents, *it, view, "lion");
25         }
26     }
27     refresh();
28 }
29 }
```

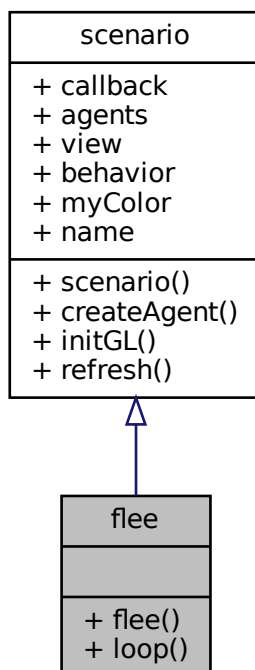
The documentation for this class was generated from the following files:

- `include/evade.h`
- `src/evade.cpp`

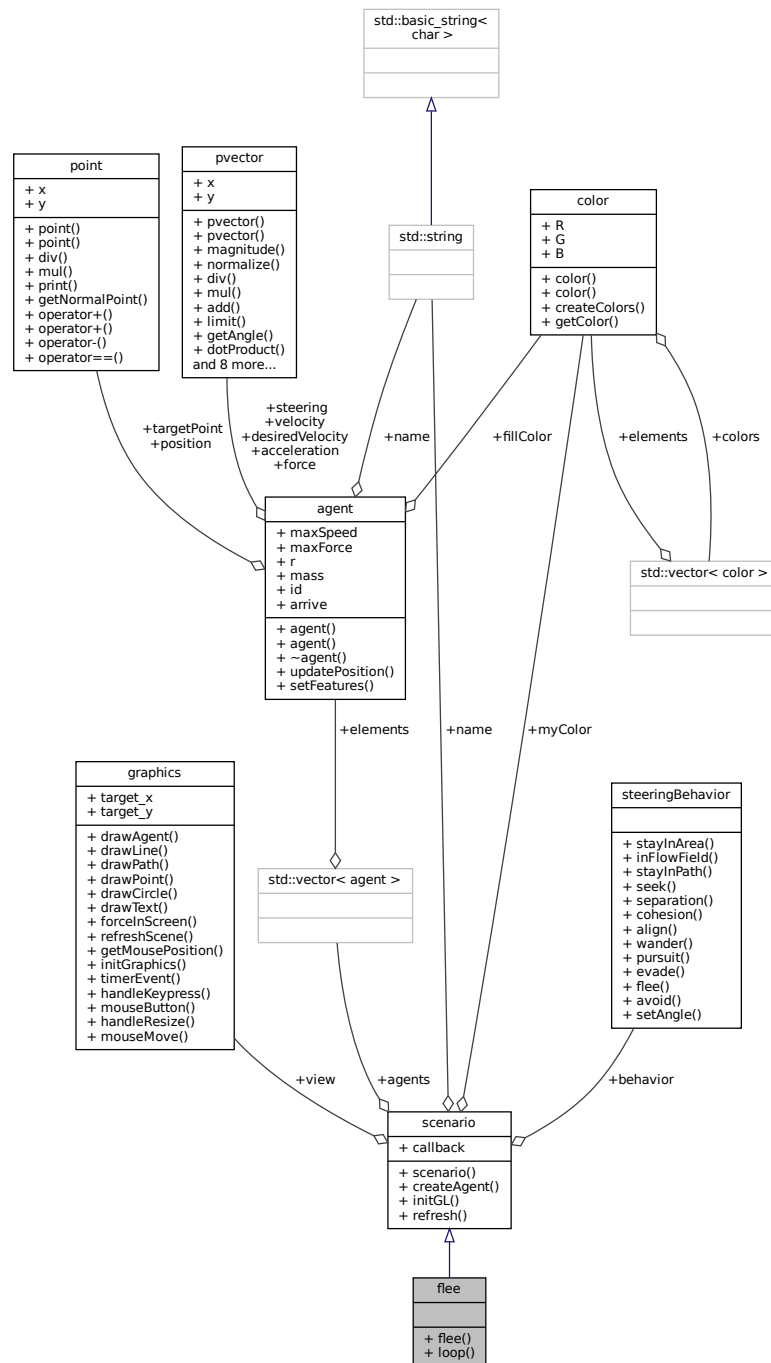
## 6.4 flee Class Reference

```
#include <flee.h>
```

Inheritance diagram for flee:



Collaboration diagram for flee:



## Public Member Functions

- [flee\(\)](#)

*default constructor.*

## Static Public Member Functions

- static void [loop](#) ()  
*evading scenario loop function*

## Additional Inherited Members

### 6.4.1 Detailed Description

Definition at line 14 of file flee.h.

### 6.4.2 Constructor & Destructor Documentation

#### 6.4.2.1 flee()

```
flee::flee ( )
```

default constructor.

Definition at line 24 of file flee.cpp.

```
25 {  
26     int agentCount = 196;  
27     name = "fleeing troop";  
28     createAgent(TROOP, &agentCount, nullptr, nullptr);  
29     callback = reinterpret_cast <void(*)()> ( (void *)(&loop) );  
30 }
```

### 6.4.3 Member Function Documentation

#### 6.4.3.1 loop()

```
void flee::loop ( ) [static]
```

evading scenario loop function

#### Note

opengl callback forces that function to be static

Definition at line 15 of file flee.cpp.

```
16 {  
17     for(auto it = agents.begin(); it < agents.end(); it++){  
18         (*it).force = behavior.flee((*it), view, view.getMousePosition());  
19     }  
20  
21     refresh();  
22 }
```

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

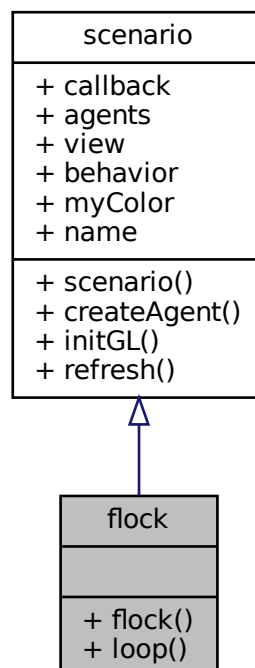
- include/[flee.h](#)
- src/[flee.cpp](#)



## 6.5 flock Class Reference

```
#include <flock.h>
```

Inheritance diagram for flock:





## Static Public Member Functions

- static void `loop` ()  
*flocking scenario loop function*

## Additional Inherited Members

### 6.5.1 Detailed Description

Definition at line 15 of file flock.h.

### 6.5.2 Constructor & Destructor Documentation

#### 6.5.2.1 flock()

```
flock::flock ( )
```

default constructor.

Definition at line 36 of file flock.cpp.

```
37 {  
38     int agentCount = 50;  
39     float maxForce = 0.3;  
40     float maxSpeed = 0.8;  
41     name = "flocking agents";  
42     createAgent(RANDOM, &agentCount, &maxForce, &maxSpeed);  
43     callback = reinterpret_cast <void(*)()> ( (void *)(&loop) );  
44 }
```

### 6.5.3 Member Function Documentation

#### 6.5.3.1 loop()

```
void flock::loop ( ) [static]
```

flocking scenario loop function

**Note**

opengl callback forces that function to be static

Definition at line 15 of file flock.cpp.

```

16 {
17     for(auto it = agents.begin(); it < agents.end(); it++){
18         view.forceInScreen((*it));
19
20         pvector sep = behavior.separation(agents, *it);
21         sep.mul(1.5);
22         pvector ali = behavior.align(agents, *it);
23         ali.mul(4);
24         pvector coh = behavior.cohesion(agents, *it);
25         coh.mul(0.1);
26
27         (*it).force = sep + ali + coh;
28         (*it).desiredVelocity = (*it).force + (*it).velocity;
29         (*it).targetPoint = (*it).position + (*it).desiredVelocity;
30         (*it).arrive = true;
31     }
32
33     refresh();
34 }
```

Here is the call graph for this function:



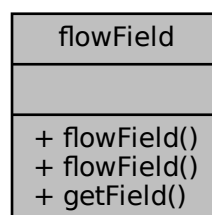
The documentation for this class was generated from the following files:

- [include/flock.h](#)
- [src/flock.cpp](#)

## 6.6 flowField Class Reference

```
#include <flowField.h>
```

Collaboration diagram for flowField:



## Public Member Functions

- [flowField](#) ()  
*default constructor.*
- [flowField](#) (pvector p)  
*constructor.*
- [pvector getField](#) (int x, int y)  
*get force at individual pixel*

### 6.6.1 Detailed Description

Definition at line 18 of file flowField.h.

### 6.6.2 Constructor & Destructor Documentation

#### 6.6.2.1 flowField() [1/2]

```
flowField::flowField ( )
```

default constructor.

See also

[flowField\(pvector p\)](#)

Definition at line 15 of file flowField.cpp.

```
16 {
17
18 }
```

#### 6.6.2.2 flowField() [2/2]

```
flowField::flowField (
    pvector p )
```

constructor.

Parameters

<i>p</i>	force vector
----------	--------------

See also

[flowField\(\)](#)

Definition at line 10 of file flowField.cpp.

```
11 {  
12     createFlowField(p);  
13 }
```

## 6.6.3 Member Function Documentation

### 6.6.3.1 getField()

```
pvector flowField::getField (  
    int x,  
    int y )
```

get force at individual pixel

#### Parameters

<i>x</i>	coordinate
<i>y</i>	coordinate

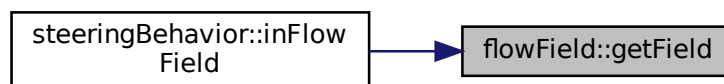
#### Returns

force at specified position

Definition at line 39 of file flowField.cpp.

```
40 {  
41     return uniformField[x][y];  
42 }
```

Here is the caller graph for this function:



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

- [include/flowField.h](#)
- [src/flowField.cpp](#)

## 6.7 graphics Class Reference

```
#include <graphics.h>
```

Collaboration diagram for graphics:

graphics
+ target_x + target_y
+ drawAgent() + drawLine() + drawPath() + drawPoint() + drawCircle() + drawText() + forceInScreen() + refreshScene() + getMousePosition() + initGraphics() + timerEvent() + handleKeypress() + mouseButton() + handleResize() + mouseMove()

### Public Member Functions

- void [drawAgent](#) ([agent](#) &[agent](#), [color](#) &[color](#))  
*drawing with corresponding angle*
- void [drawLine](#) ([point](#) p1, [point](#) p2, [color](#) cl)  
*drawing line*
- void [drawPath](#) ([path](#) &[path](#), [color](#) color)  
*draws path*
- void [drawPoint](#) ([point](#) p)  
*draws point*
- void [drawCircle](#) ([point](#) p, float radius)  
*draws circle*
- void [drawText](#) (string text, [point](#) p)  
*draws text on screen*
- void [forceInScreen](#) ([agent](#) &[agent](#))  
*changes agent position so that it stays in screen*
- void [refreshScene](#) ()  
*update agent position*
- [point](#) [getMousePosition](#) ()  
*gets mouse position*
- void [initGraphics](#) (int \*argv, char \*\*argc, void(\*callback)())  
*initialization of graphics*

## Static Public Member Functions

- static void `timerEvent` (int value)  
*periodic timer event*
- static void `handleKeyPress` (unsigned char key, int x, int y)  
*key press event*
- static void `mouseButton` (int button, int state, int x, int y)  
*mouse press event*
- static void `handleResize` (int w, int h)  
*event triggered with screen resizing*
- static void `mouseMove` (int x, int y)  
*event triggered with mouse movements*

## Static Public Attributes

- static int `target_x` = `-WIDTH`  
*mouse position x*
- static int `target_y` = `HEIGHT`  
*mouse position y*

### 6.7.1 Detailed Description

Definition at line 22 of file `graphics.h`.

### 6.7.2 Member Function Documentation

#### 6.7.2.1 `drawAgent()`

```
void graphics::drawAgent (
    agent & agent,
    color & color )
```

drawing with corresponding angle

#### Parameters

<i>agent</i>	instance to change
<i>color</i>	of the agent

Definition at line 160 of file `graphics.cpp`.

```
161 {
162     glPushMatrix();
163     glTranslatef(agent.position.x, agent.position.y, 0.0f);
164     glRotatef(agent.velocity.getAngle(), 0.0f, 0.0f, 1.0f);
165     glBegin(GL_TRIANGLES);
166     glColor3f( color.R, color.G, color.B);
167     glVertex3f( 1.0f, 0.0f, 0.0f);
```



```

168     glVertex3f(-1.0f,  0.5f, 0.0f);
169     glVertex3f(-1.0f, -0.5f, 0.0f);
170     glEnd();
171     glPopMatrix();
172 }

```

Here is the call graph for this function:



### 6.7.2.2 drawCircle()

```

void graphics::drawCircle (
    point p,
    float radius )

```

draws circle

#### Parameters

<i>p</i>	center of the circle
<i>radius</i>	radius of the circle

Definition at line 138 of file graphics.cpp.

```

139 {
140     glBegin(GL_LINE_STRIP);
141     glLineWidth(2);
142     for (int i = 0; i <= 300; i++) {
143         float angle = 2 * PI * i / 300;
144         float x = cos(angle) * radius;
145         float y = sin(angle) * radius;
146         glVertex2d(p.x + x, p.y + y);
147     }
148     glEnd();
149 }

```

### 6.7.2.3 drawLine()

```

void graphics::drawLine (
    point p1,
    point p2,
    color c1 )

```

drawing line

## Parameters

<i>p1</i>	start point of the line
<i>p2</i>	end point of the line
<i>color</i>	of the line

Definition at line 128 of file graphics.cpp.

```

129 {
130     glColor3f( c1.R, c1.G, c1.B);
131     glLineWidth(2);
132     glBegin(GL_LINES);
133     glVertex2f(p1.x, p1.y);
134     glVertex2f(p2.x, p2.y);
135     glEnd();
136 }
```

### 6.7.2.4 drawPath()

```

void graphics::drawPath (
    path & path,
    color color )
```

draws path

## Parameters

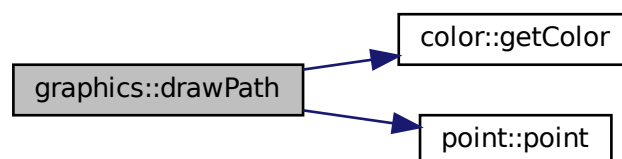
<i>path</i>	to draw
<i>color</i>	of the path

Definition at line 114 of file graphics.cpp.

```

115 {
116     point p1, p2;
117     for(auto it = path.points.begin(); it < path.points.end()-1; it++){
118         p1 = point((*it).x, (*it).y - path.width/2);
119         p2 = point((*it+1).x, (*it+1).y - path.width/2);
120         drawLine(p1, p2, color.getColor(BLUE));
121
122         p1 = point((*it).x, (*it).y + path.width/2);
123         p2 = point((*it+1).x, (*it+1).y + path.width/2);
124         drawLine(p1, p2, color.getColor(BLUE));
125     }
126 }
```

Here is the call graph for this function:



### 6.7.2.5 drawPoint()

```
void graphics::drawPoint (
    point p )
```

draws point

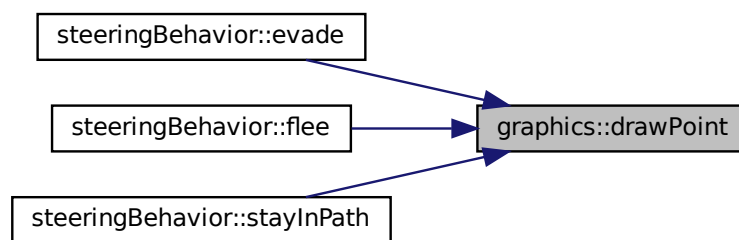
#### Parameters

<i>p</i>	point to draw
----------	---------------

Definition at line 151 of file graphics.cpp.

```
152 {
153     glColor3f(1,1,1);
154     glPointSize(4.0);
155     glBegin(GL_POINTS);
156     glVertex2f(p.x, p.y);
157     glEnd();
158 }
```

Here is the caller graph for this function:



### 6.7.2.6 drawText()

```
void graphics::drawText (
    string text,
    point p )
```

draws text on screen

#### Parameters

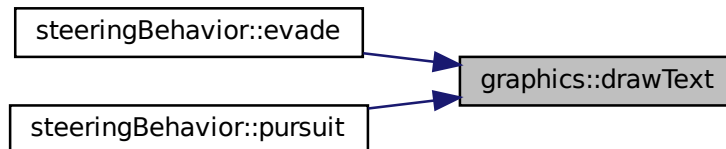
<i>p</i>	position of the text
<i>text</i>	to display

Definition at line 21 of file graphics.cpp.

```

22 {
23     glColor3f (0.0, 0.0, 1.0);
24     //glRasterPos2f(-34, 32.5);
25     glRasterPos2f(p.x, p.y);
26     for ( string::iterator it=text.begin(); it!=text.end(); ++it){
27         glutBitmapCharacter(GLUT_BITMAP_9_BY_15, *it);
28     }
29 }
```

Here is the caller graph for this function:



### 6.7.2.7 forceInScreen()

```

void graphics::forceInScreen (
    agent & agent )
```

changes agent position so that it stays in screen

#### Parameters

<i>agent</i>	instance
--------------	----------

Definition at line 63 of file graphics.cpp.

```

64 {
65     if(agent.position.x > WIDTH)
66         agent.position.x -= 2 * WIDTH;
67     if(agent.position.x < -WIDTH)
68         agent.position.x += 2 * WIDTH;
69     if(agent.position.y > HEIGHT)
70         agent.position.y -= 2 * HEIGHT;
71     if(agent.position.y < -HEIGHT)
72         agent.position.y += 2 * HEIGHT;
73 }
```

### 6.7.2.8 getMousePosition()

```

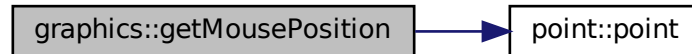
point graphics::getMousePosition ( )
```

gets mouse position

Definition at line 58 of file graphics.cpp.

```
59 {
60     return point (graphics::target_x, graphics::target_y);
61 }
```

Here is the call graph for this function:



### 6.7.2.9 handleKeypress()

```
void graphics::handleKeypress (
    unsigned char key,
    int x,
    int y ) [static]
```

key press event

#### Parameters

<i>key</i>	pressed
<i>x</i>	unused but required for openGL
<i>y</i>	unused but required for openGL

Definition at line 107 of file graphics.cpp.

```
108 {
109     if (key == ESC) {
110         exit (0);
111     }
112 }
```

Here is the caller graph for this function:



### 6.7.2.10 handleResize()

```
void graphics::handleResize (
    int w,
    int h ) [static]
```

event triggered with screen resizing

#### Parameters

<i>w</i>	width of the screen
<i>h</i>	height of the screen

Definition at line 83 of file graphics.cpp.

```
84 {
85     glViewport(0, 0, w, h); //Tell OpenGL how to convert from coordinates to pixel values
86     glMatrixMode(GL_PROJECTION); //Switch to setting the camera perspective
87     glLoadIdentity(); //Reset the camera
88     //Set the camera perspective
89     gluPerspective(45.0,          //The camera angle
90                   (double)w / (double)h, //The width-to-height ratio
91                   1.0,             //The near z clipping coordinate
92                   200.0);          //The far z clipping coordinate
93 }
```

Here is the caller graph for this function:



### 6.7.2.11 initGraphics()

```
void graphics::initGraphics (
    int * argv,
    char ** argc,
    void(*)() callback )
```

initialization of graphics

#### Parameters

<i>argv</i>	user parameters
<i>argc</i>	count of user parameters
<i>callback</i>	loop function for openGL periodic callback

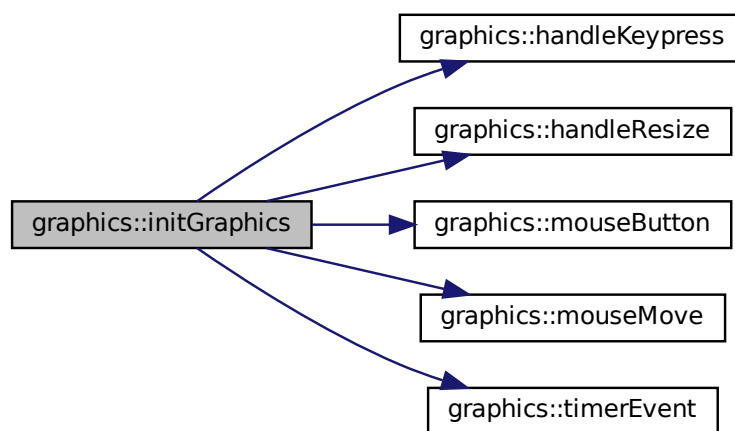
Definition at line 41 of file graphics.cpp.

```

42 {
43     glutInit(argv, argc);
44     glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
45     glutInitWindowSize(400, 400);
46     glutCreateWindow("Autonomous Steering Agents");
47     glClearColor(0.7f, 0.7f, 0.7f, 1.0f); //set background color
48     glEnable(GL_DEPTH_TEST);
49     glutDisplayFunc(*callback);
50     glutMouseFunc(graphics::mouseButton);
51     glutPassiveMotionFunc(graphics::mouseMove);
52     glutKeyboardFunc(graphics::handleKeypress);
53     glutReshapeFunc(graphics::handleResize);
54     glutTimerFunc(20, graphics::timerEvent, 0);
55     glutMainLoop();
56 }

```

Here is the call graph for this function:



### 6.7.2.12 mouseButton()

```

void graphics::mouseButton (
    int button,
    int state,
    int x,
    int y ) [static]

```

mouse press event

#### Parameters

<i>button</i>	mouse key pressed
<i>x</i>	unused but required for openGL
<i>y</i>	unused but required for openGL

Definition at line 101 of file `graphics.cpp`.

```

102 {
103     if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
104     }
105 }

```

Here is the caller graph for this function:



#### 6.7.2.13 mouseMove()

```

void graphics::mouseMove (
    int x,
    int y ) [static]

```

event triggered with mouse movements

##### Parameters

<i>x</i>	osition of the mouse
<i>y</i>	position of the mouse

Definition at line 75 of file graphics.cpp.

```

76 {
77     //TODO: mouse position to glut
78     //TODO: magic numbers
79     graphics::target_x = x / 5.88 - 34;
80     graphics::target_y = 34 - y / 5.88;
81 }

```

Here is the caller graph for this function:





### 6.7.2.14 refreshScene()

```
void graphics::refreshScene ( )
```

update agent position

Definition at line 32 of file graphics.cpp.

```
33 {
34     glutSwapBuffers();
35     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
36     glMatrixMode(GL_MODELVIEW); //Switch to the drawing perspective
37     glLoadIdentity(); //Reset the drawing perspective
38     glTranslatef(0.0f, 0.0f, -85.0f); //Move to the center of the triangle
39 }
```

### 6.7.2.15 timerEvent()

```
void graphics::timerEvent (
    int value ) [static]
```

periodic timer event

#### Parameters

<i>value</i>	period as ms
--------------	--------------

Definition at line 95 of file graphics.cpp.

```
96 {
97     glutPostRedisplay(); //Tell GLUT that the display has changed
98     glutTimerFunc(value, timerEvent, 20);
99 }
```

Here is the caller graph for this function:



## 6.7.3 Member Data Documentation

### 6.7.3.1 target\_x

```
int graphics::target_x = -WIDTH [static]
```

mouse position x

Definition at line 129 of file graphics.h.

### 6.7.3.2 target\_y

```
int graphics::target_y = HEIGHT [static]
```

mouse position y

Definition at line 134 of file graphics.h.

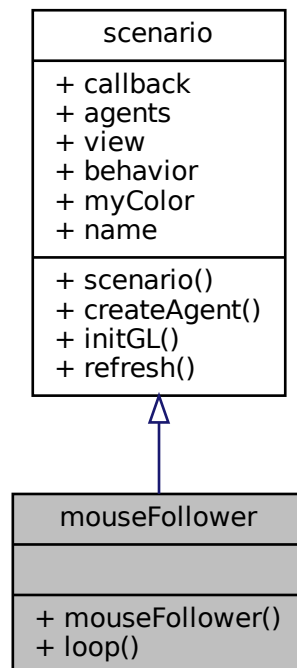
The documentation for this class was generated from the following files:

- [include/graphics.h](#)
- [src/graphics.cpp](#)

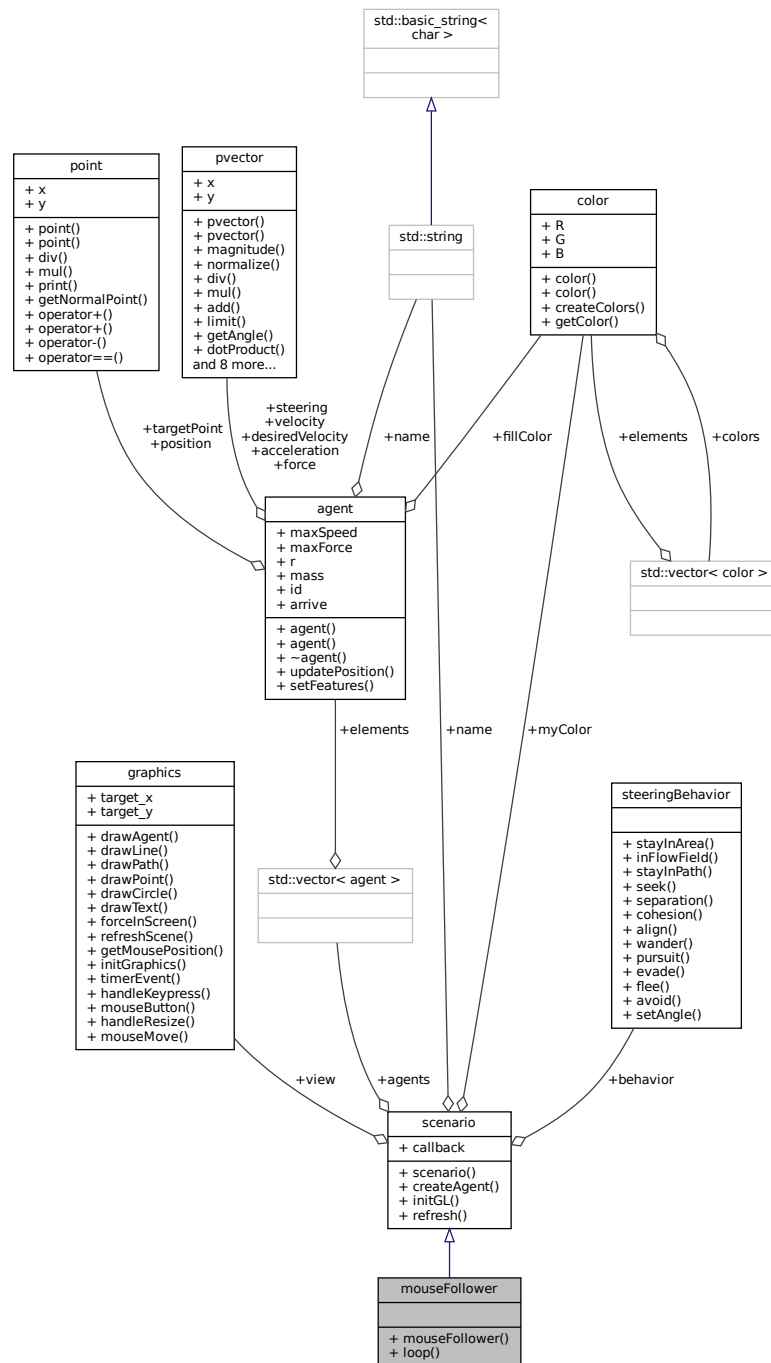
## 6.8 mouseFollower Class Reference

```
#include <mouseFollower.h>
```

Inheritance diagram for mouseFollower:



Collaboration diagram for mouseFollower:



## Public Member Functions

- [mouseFollower](#) ()

*default constructor.*

## Static Public Member Functions

- static void [loop](#) ()  
*mouse following scenario loop function*

## Additional Inherited Members

### 6.8.1 Detailed Description

Definition at line 14 of file [mouseFollower.h](#).

### 6.8.2 Constructor & Destructor Documentation

#### 6.8.2.1 [mouseFollower\(\)](#)

`mouseFollower::mouseFollower ( )`

default constructor.

Definition at line 25 of file [mouseFollower.cpp](#).

```
26 {  
27     int agentCount = 30;  
28     float maxForce = 0.3;  
29     float maxSpeed = 0.6;  
30     name = "mouse following";  
31     createAgent(RANDOM, &agentCount, &maxForce, &maxSpeed);  
32     callback = reinterpret_cast <void(*) ()> ( (void *)(&loop) );  
33 }
```

### 6.8.3 Member Function Documentation

#### 6.8.3.1 [loop\(\)](#)

`void mouseFollower::loop ( ) [static]`

mouse following scenario loop function

#### Note

opengl callback forces that function to be static

Definition at line 15 of file [mouseFollower.cpp](#).

```
16 {  
17     for(auto it = agents.begin(); it < agents.end(); it++){  
18         (*it).targetPoint = view.getMousePosition();  
19         (*it).force = behavior.seek(*it);  
20         (*it).arrive = true;  
21     }  
22     refresh();  
23 }
```

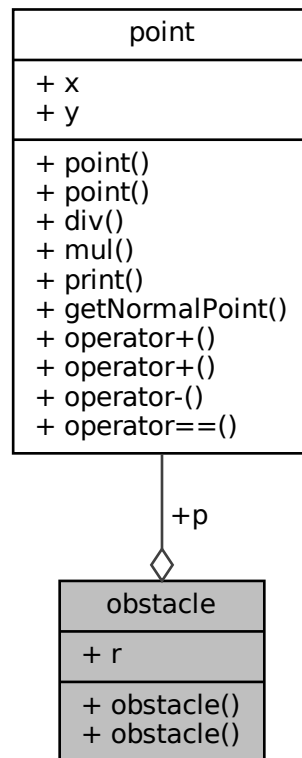
The documentation for this class was generated from the following files:

- include/[mouseFollower.h](#)
- src/[mouseFollower.cpp](#)

## 6.9 obstacle Class Reference

```
#include <obstacle.h>
```

Collaboration diagram for obstacle:



### Public Member Functions

- `obstacle ()`  
*default constructor.*
- `obstacle (point p, float r)`  
*constructor*

### Public Attributes

- `point p`  
*center point of the obstacle*
- `float r`  
*radius of the obstacle*

### 6.9.1 Detailed Description

Definition at line 12 of file obstacle.h.

### 6.9.2 Constructor & Destructor Documentation

#### 6.9.2.1 obstacle() [1/2]

```
obstacle::obstacle ( )
```

default constructor.

See also

[obstacle\(point p, float r](#)

Definition at line 15 of file obstacle.cpp.

```
16 {
17
18 }
```

#### 6.9.2.2 obstacle() [2/2]

```
obstacle::obstacle (
    point p,
    float r )
```

constructor

Parameters

<i>p</i>	center of the circular obstacle
<i>r</i>	radius of the obstacle

See also

[obstacle\(point p, float r\);](#)

Definition at line 20 of file obstacle.cpp.

```
21 {
22     this->p = p;
23     this->r = r;
24 }
```

### 6.9.3 Member Data Documentation

### 6.9.3.1 p

```
point obstacle::p
```

center point of the obstacle

Definition at line 31 of file obstacle.h.

### 6.9.3.2 r

```
float obstacle::r
```

radius of the obstacle

Definition at line 36 of file obstacle.h.

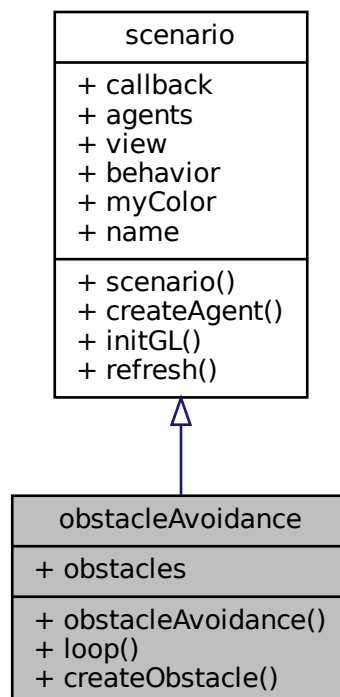
The documentation for this class was generated from the following files:

- include/[obstacle.h](#)
- src/[obstacle.cpp](#)

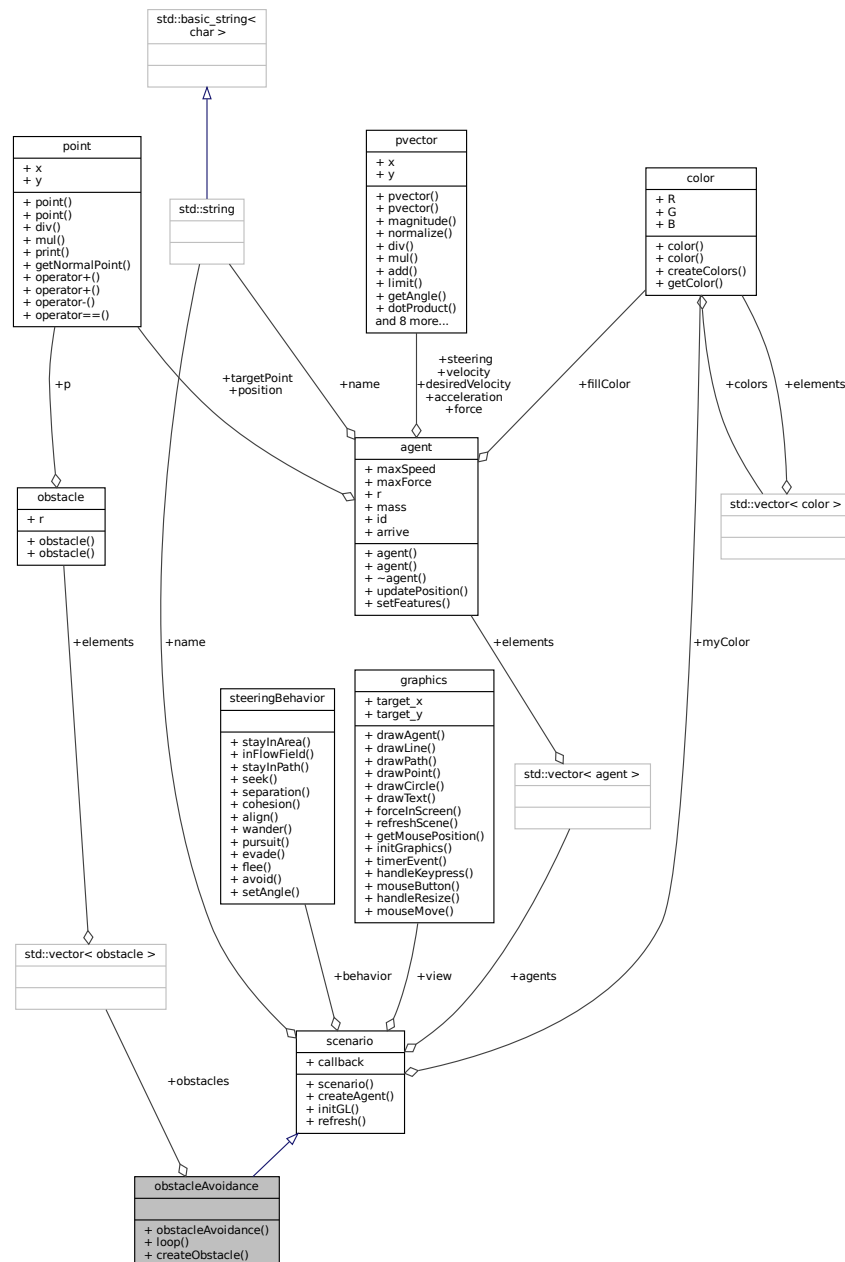
## 6.10 obstacleAvoidance Class Reference

```
#include <obstacleAvoidance.h>
```

Inheritance diagram for obstacleAvoidance:



Collaboration diagram for obstacleAvoidance:



## Public Member Functions

- [obstacleAvoidance](#) ()  
*default constructor.*

## Static Public Member Functions

- static void [loop](#) ()  
*obstacle avoidance scenario loop function*
- static void [createObstacle](#) (vector< [obstacle](#) > &obstacles)  
*creation of list of obstacles*



## Static Public Attributes

- static vector< [obstacle](#) > [obstacles](#)  
*list of obstacles*

## Additional Inherited Members

### 6.10.1 Detailed Description

Definition at line 15 of file obstacleAvoidance.h.

### 6.10.2 Constructor & Destructor Documentation

#### 6.10.2.1 obstacleAvoidance()

```
obstacleAvoidance::obstacleAvoidance ( )
```

default constructor.

Definition at line 43 of file obstacleAvoidance.cpp.

```
44 {
45     name = "avoid obstacles";
46     createAgent(STATIC, nullptr, nullptr, nullptr);
47     createObstacle(obstacles);
48     callback = reinterpret_cast <void(*)()> ( (void *)(&loop) );
49 }
```

### 6.10.3 Member Function Documentation

#### 6.10.3.1 createObstacle()

```
void obstacleAvoidance::createObstacle (
    vector< obstacle > & obstacles ) [static]
```

creation of list of obstacles

#### Parameters

<i>obstacles</i>	list to be created
------------------	--------------------

#### Note

opengl callback forces that function to be static

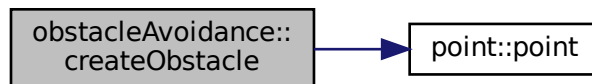
Definition at line 36 of file obstacleAvoidance.cpp.

```

37 {
38     obstacles.push_back(obstacle(point(0,0), 8));
39     obstacles.push_back(obstacle(point(-20,0), 3));
40     obstacles.push_back(obstacle(point(20,-10), 4));
41 }

```

Here is the call graph for this function:



### 6.10.3.2 loop()

```
void obstacleAvoidance::loop ( ) [static]
```

obstacle avoidance scenario loop function

#### Note

opengl callback forces that function to be static

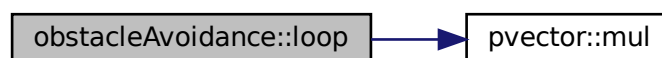
Definition at line 17 of file obstacleAvoidance.cpp.

```

18 {
19     for(auto it = agents.begin(); it < agents.end(); it++){
20         for(auto it = obstacles.begin(); it < obstacles.end(); it++){
21             point p = (*it).p;
22             view.drawCircle(p, (*it).r);
23         }
24     }
25     (*it).targetPoint = view.getMousePosition();
26     pvector seek = behavior.seek(*it);
27     seek.mul(0.5);
28
29     pvector avoid = behavior.avoid(obstacles, *it);
30     (*it).force = avoid + seek;
31     (*it).arrive = true;
32 }
33 refresh();
34 }

```

Here is the call graph for this function:



## 6.10.4 Member Data Documentation

### 6.10.4.1 obstacles

```
vector< obstacle > obstacleAvoidance::obstacles [static]
```

list of obstacles

#### Note

opengl callback forces that function to be static

Definition at line 32 of file obstacleAvoidance.h.

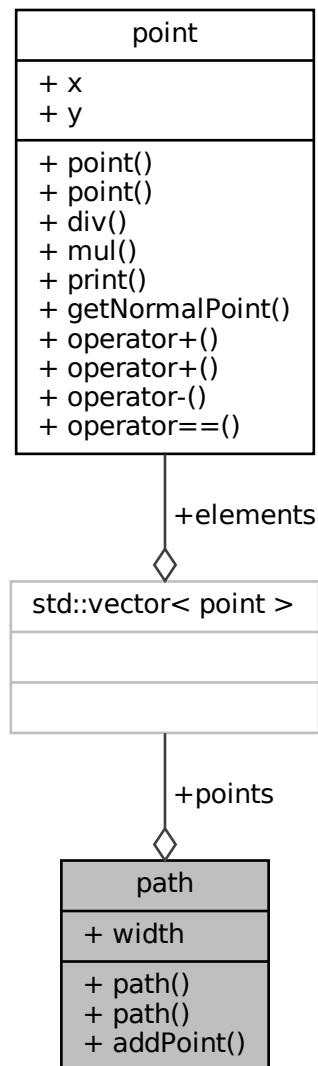
The documentation for this class was generated from the following files:

- include/[obstacleAvoidance.h](#)
- src/[obstacleAvoidance.cpp](#)

## 6.11 path Class Reference

```
#include <path.h>
```

Collaboration diagram for path:



## Public Member Functions

- `path ()`  
*default constructor.*
- `path (float width)`  
*donstructor.*
- `void addPoint (point p)`  
*adds a new point to the path*

## Public Attributes

- `vector< point > points`

- list of points added to the path*
  - int [width](#)  
*width of the path*

### 6.11.1 Detailed Description

Definition at line 15 of file path.h.

### 6.11.2 Constructor & Destructor Documentation

#### 6.11.2.1 `path()` [1/2]

```
path::path ( )
```

default constructor.

See also

[path\(float width\)](#)

Definition at line 16 of file path.cpp.

```
17 {
18
19 }
```

#### 6.11.2.2 `path()` [2/2]

```
path::path (
    float width )
```

donstructor.

Parameters

<i>width</i>	The width of the path.
--------------	------------------------

See also

[path\(\)](#)

Definition at line 21 of file path.cpp.

```
22 {
23     this->width = width;
24 }
```

### 6.11.3 Member Function Documentation

#### 6.11.3.1 addPoint()

```
void path::addPoint (
    point p )
```

adds a new point to the path

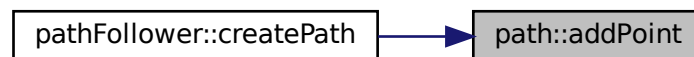
##### Parameters

<i>point</i>	to add to the path
--------------	--------------------

Definition at line 11 of file path.cpp.

```
12 {
13     points.push_back(p);
14 }
```

Here is the caller graph for this function:



### 6.11.4 Member Data Documentation

#### 6.11.4.1 points

```
vector<point> path::points
```

list of points added to the path

Definition at line 39 of file path.h.

#### 6.11.4.2 width

```
int path::width
```

width of the path

Definition at line 44 of file path.h.

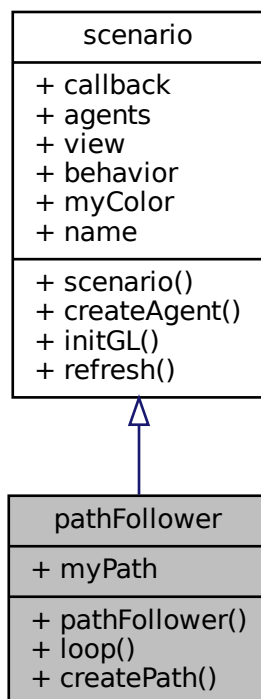
The documentation for this class was generated from the following files:

- [include/path.h](#)
- [src/path.cpp](#)

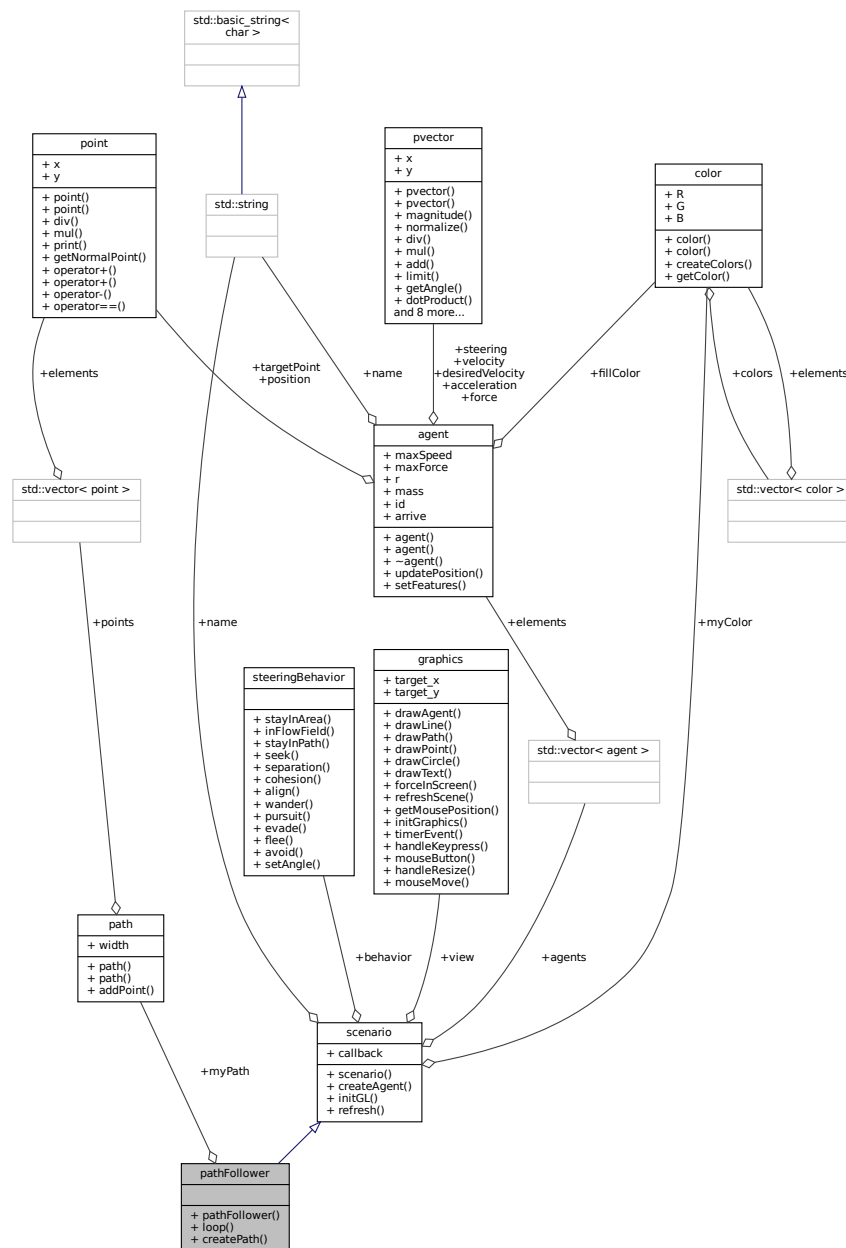
## 6.12 pathFollower Class Reference

```
#include <pathFollower.h>
```

Inheritance diagram for pathFollower:



Collaboration diagram for pathFollower:



## Public Member Functions

- [pathFollower](#) ()  
default constructor.

## Static Public Member Functions

- static void [loop](#) ()  
path follower scenario loop function
- static void [createPath](#) (path &p)  
creates path



## Static Public Attributes

- static [path myPath](#)  
*path that will be followed*

## Additional Inherited Members

### 6.12.1 Detailed Description

Definition at line 14 of file pathFollower.h.

### 6.12.2 Constructor & Destructor Documentation

#### 6.12.2.1 pathFollower()

```
pathFollower::pathFollower ( )
```

default constructor.

Definition at line 37 of file pathFollower.cpp.

```
38 {
39     int agentCount = 40;
40     float maxForce = 0.2;
41     float maxSpeed = 0.4;
42     myPath = path(8);
43     createPath(myPath);
44     name = "path following";
45     createAgent(RANDOM, &agentCount, &maxForce, &maxSpeed);
46     callback = reinterpret_cast<void(*)()> ( (void *)(&loop) );
47 }
```

### 6.12.3 Member Function Documentation

#### 6.12.3.1 createPath()

```
void pathFollower::createPath (
    path & p ) [static]
```

creates path

Parameters

<i>path</i>	to create
-------------	-----------

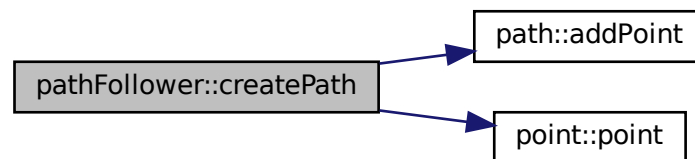
**Note**

opengl callback forces that function to be static

Definition at line 29 of file pathFollower.cpp.

```
30 {
31     p.addPoint(point(-40, 5));
32     p.addPoint(point(-14, 15));
33     p.addPoint(point( 10, 7));
34     p.addPoint(point( 40, 12));
35 }
```

Here is the call graph for this function:

**6.12.3.2 loop()**

```
void pathFollower::loop ( ) [static]
```

path follower scenario loop function

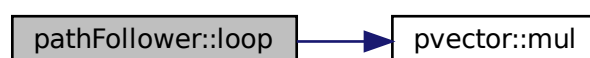
**Note**

opengl callback forces that function to be static

Definition at line 17 of file pathFollower.cpp.

```
18 {
19     for(auto it = agents.begin(); it < agents.end(); it++){
20         view.drawPath(myPath, myColor);
21         pvector flwpth = behavior.stayInPath(*it, myPath, view);
22         pvector sep = behavior.separation(agents, *it);
23         sep.mul(5);
24         (*it).force = sep + flwpth;
25     }
26     refresh();
27 }
```

Here is the call graph for this function:



## 6.12.4 Member Data Documentation

### 6.12.4.1 myPath

```
path pathFollower::myPath [static]
```

path that will be followed

#### Note

opengl callback forces that function to be static

Definition at line 38 of file pathFollower.h.

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

- include/pathFollower.h
- src/pathFollower.cpp

## 6.13 point Class Reference

```
#include <point.h>
```

Collaboration diagram for point:

point
+ x + y
+ point() + point() + div() + mul() + print() + getNormalPoint() + operator+() + operator+() + operator-() + operator==( )

## Public Member Functions

- `point ()`  
*default constructor*
- `point (float x, float y)`  
*constructor*
- `void div (float d)`  
*divide point*
- `void mul (float d)`  
*multiply point*
- `void print (const string &s)`  
*debug function*
- `void getNormalPoint (point predicted, point start, point end)`  
*provides normal point on a vector of a point*
- `point operator+ (pvector const &obj)`  
*overloaded + operator*
- `point operator+ (point const &obj)`  
*overloaded + operator*
- `pvector operator- (point const &obj)`  
*overloaded - operator*
- `bool operator== (point const &obj)`  
*overloaded == operator*

## Public Attributes

- `float x`  
*x position*
- `float y`  
*y position*

### 6.13.1 Detailed Description

Definition at line 15 of file point.h.

### 6.13.2 Constructor & Destructor Documentation

### 6.13.2.1 point() [1/2]

```
point::point ( )
```

default constructor

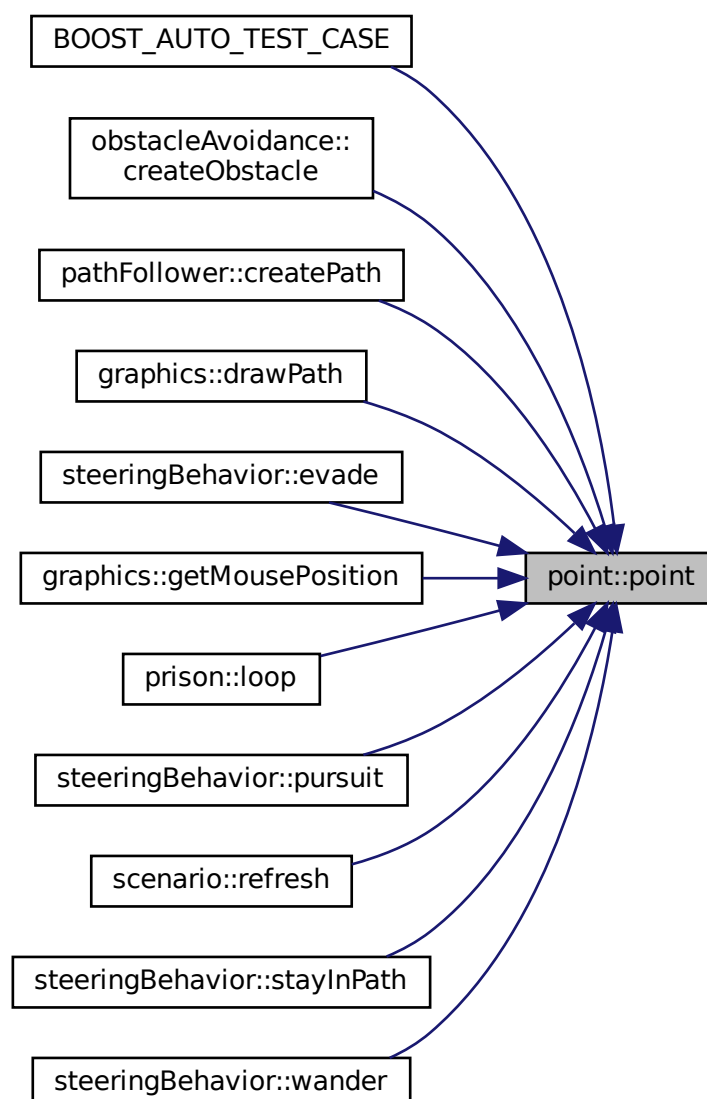
See also

[point\(float x, float y\)](#)

Definition at line 21 of file point.cpp.

```
21 {}
```

Here is the caller graph for this function:



### 6.13.2.2 point() [2/2]

```
point::point (
    float x,
    float y )
```

constructor

#### Parameters

<i>x</i>	position x of the point
<i>y</i>	position y of the point

See also

[point\(\)](#)

Definition at line 15 of file point.cpp.

```
16 {
17     this->x = x;
18     this->y = y;
19 }
```

## 6.13.3 Member Function Documentation

### 6.13.3.1 div()

```
void point::div (
    float d )
```

divide point

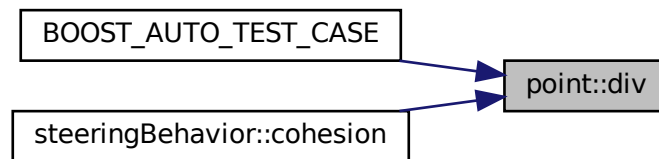
#### Parameters

<i>d</i>	scalar to divide position of the point
----------	--

Definition at line 38 of file point.cpp.

```
39 {
40     x = x / d;
41     y = y / d;
42 }
```

Here is the caller graph for this function:



### 6.13.3.2 getNormalPoint()

```

void point::getNormalPoint (
    point predicted,
    point start,
    point end )
  
```

provides normal point on a vector of a point

#### Parameters

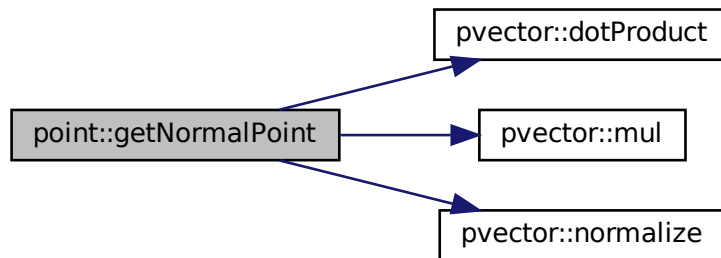
<i>predicted</i>	point that caller require normal on the vector
<i>start</i>	point of the vector
<i>end</i>	point of the vector

Definition at line 67 of file point.cpp.

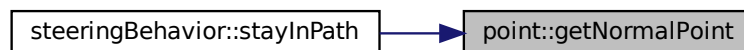
```

68 {
69     pvector a = predicted - start;
70     pvector b = end - start;
71     b.normalize();
72     float a_dot_b = a.dotProduct(b);
73     b.mul(a_dot_b);
74     point normalPoint = start + b;
75     this->x = normalPoint.x;
76     this->y = normalPoint.y;
77 }
  
```

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.13.3.3 mul()

```
void point::mul (
    float d )
```

multiply point

#### Parameters

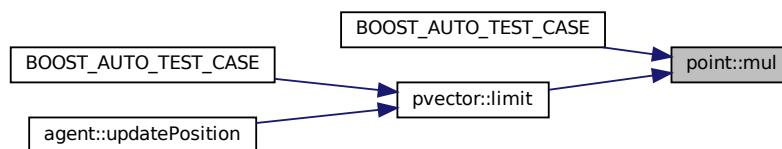
<i>d</i>	scalar to multiply position of the point
----------	--

Definition at line 44 of file point.cpp.

```
45 {
46     x = x * d;
47     y = y * d;
48 }
```



Here is the caller graph for this function:



### 6.13.3.4 operator+() [1/2]

```
point point::operator+ (
    point const & obj )
```

overloaded + operator

#### Parameters

<i>obj</i>	point to add
------------	--------------

#### Returns

sum

Definition at line 51 of file point.cpp.

```
52 {
53     point res;
54     res.x = x + obj.x;
55     res.y = y + obj.y;
56     return res;
57 }
```

### 6.13.3.5 operator+() [2/2]

```
point point::operator+ (
    pvector const & obj )
```

overloaded + operator

#### Parameters

<i>obj</i>	vector to add
------------	---------------

**Returns**

sum

Definition at line 23 of file point.cpp.

```
24 {  
25     point res;  
26     res.x = x + obj.x;  
27     res.y = y + obj.y;  
28     return res;  
29 }
```

**6.13.3.6 operator-()**

```
pvector point::operator- (  
    point const & obj )
```

overloaded - operator

**Parameters**

<i>obj</i>	point to subtract
------------	-------------------

**Returns**

difference

Definition at line 59 of file point.cpp.

```
60 {  
61     pvector res;  
62     res.x = x - obj.x;  
63     res.y = y - obj.y;  
64     return res;  
65 }
```

**6.13.3.7 operator==()**

```
bool point::operator== (  
    point const & obj )
```

overloaded == operator

**Parameters**

<i>obj</i>	point to compare
------------	------------------

**Returns**

comparison result

Definition at line 31 of file point.cpp.

```
32 {  
33     if(x == obj.x && y == obj.y)  
34         return true;  
35     return false;  
36 }
```

### 6.13.3.8 print()

```
void point::print (  
    const string & s )
```

debug function

Parameters

s	explanation string of the log
---	-------------------------------

Definition at line 79 of file point.cpp.

```
80 {  
81     cout << " " << s << " " << x << " " << y << endl;  
82 }
```

## 6.13.4 Member Data Documentation

### 6.13.4.1 x

```
float point::x
```

x position

Definition at line 88 of file point.h.

### 6.13.4.2 y

```
float point::y
```

y position

Definition at line 93 of file point.h.

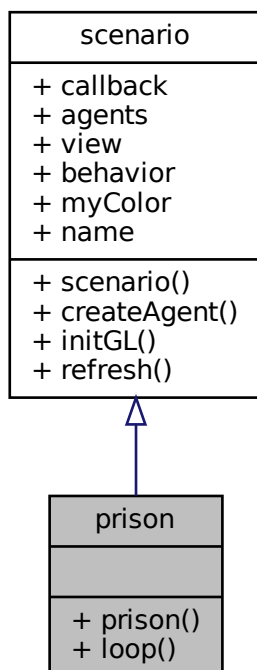
The documentation for this class was generated from the following files:

- include/[point.h](#)
- src/[point.cpp](#)

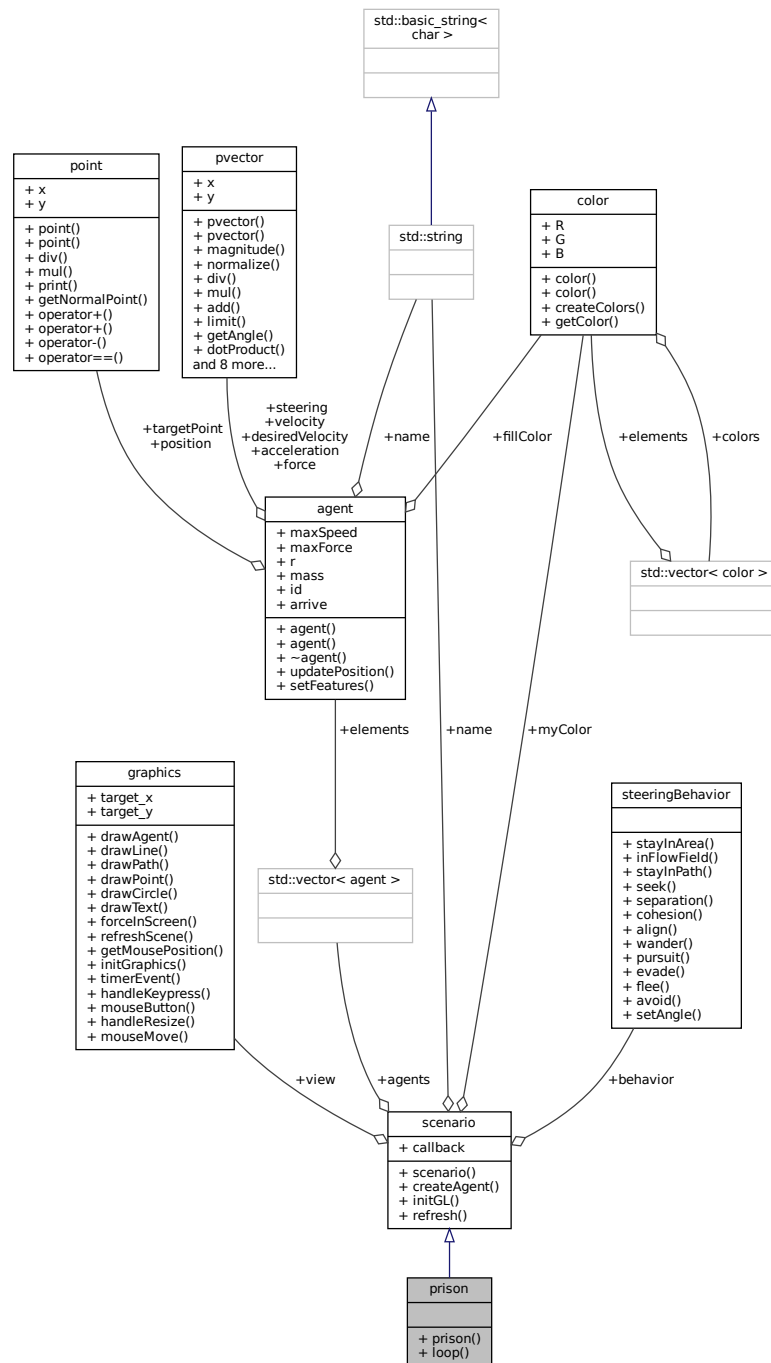
## 6.14 prison Class Reference

```
#include <prison.h>
```

Inheritance diagram for prison:



Collaboration diagram for prison:



## Public Member Functions

- [prison \(\)](#)

*default constructor.*

## Static Public Member Functions

- static void `loop` ()  
*prisoning scenario loop function*

## Additional Inherited Members

### 6.14.1 Detailed Description

Definition at line 15 of file prison.h.

### 6.14.2 Constructor & Destructor Documentation

#### 6.14.2.1 `prison()`

```
prison::prison ( )
```

default constructor.

Definition at line 31 of file prison.cpp.

```
32 {  
33     int agentCount = 30;  
34     float maxForce = 0.6;  
35     float maxSpeed = 0.6;  
36  
37     name = "stay in prison";  
38     createAgent(RANDOM, &agentCount, &maxForce, &maxSpeed);  
39     callback = reinterpret_cast <void(*)()> ( (void *)(&loop) );  
40 }
```

### 6.14.3 Member Function Documentation

#### 6.14.3.1 `loop()`

```
void prison::loop ( ) [static]
```

prisoning scenario loop function

prison loop function

**Note**

opengl callback forces that function to be static

Definition at line 18 of file prison.cpp.

```
19 {  
20     for(auto it = agents.begin(); it < agents.end(); it++){  
21         view.drawLine(point(-WALL, WALL), point(WALL, WALL), myColor.getColor(BLUE));  
22         view.drawLine(point(WALL, WALL), point(WALL, -WALL), myColor.getColor(BLUE));  
23         view.drawLine(point(WALL, -WALL), point(-WALL, -WALL), myColor.getColor(BLUE));  
24         view.drawLine(point(-WALL, WALL), point(-WALL, -WALL), myColor.getColor(BLUE));  
25         (*it).force = behavior.stayInArea(*it, WALL - DISTANCE);  
26         (*it).force += behavior.separation(agents, *it);  
27     }  
28     refresh();  
29 }
```

Here is the call graph for this function:



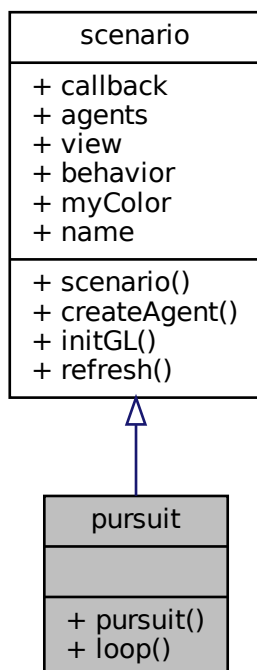
The documentation for this class was generated from the following files:

- include/prison.h
- src/prison.cpp

## 6.15 pursuit Class Reference

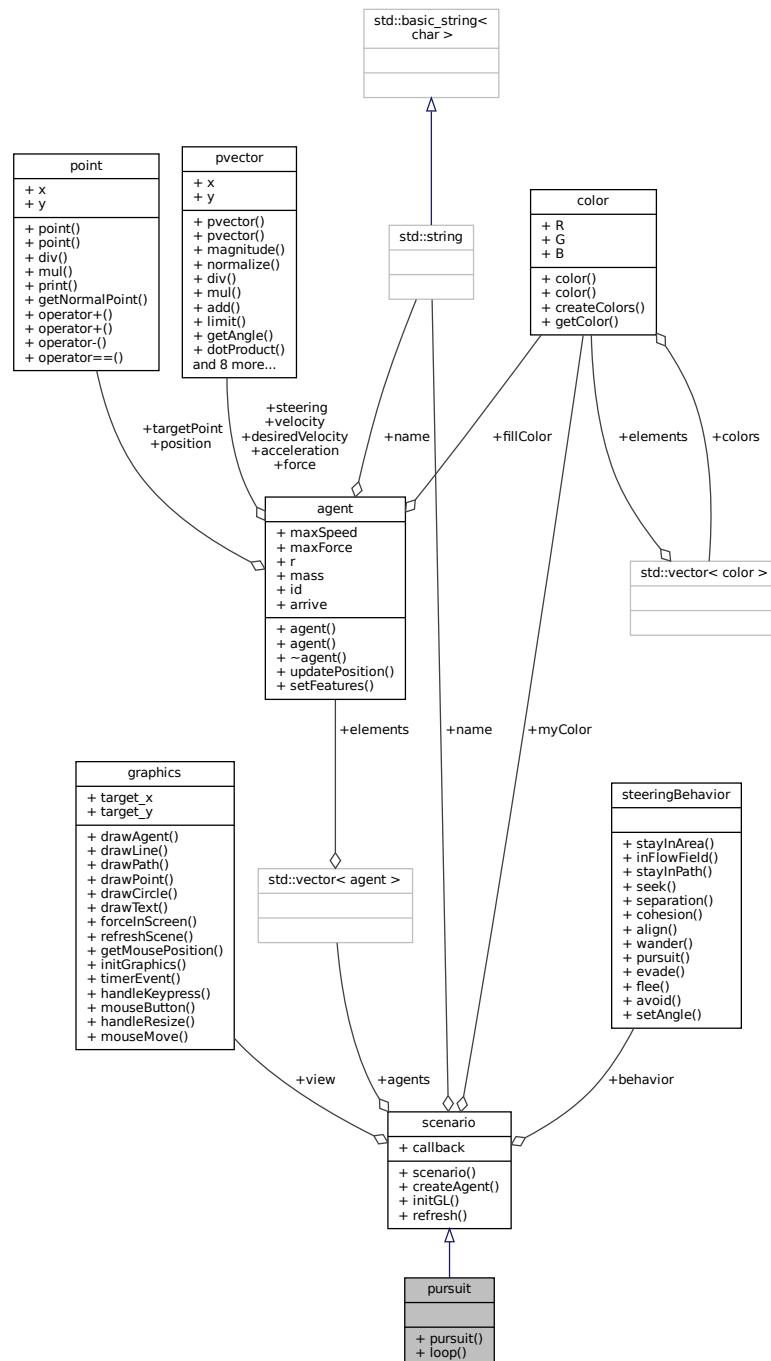
```
#include <pursuit.h>
```

Inheritance diagram for pursuit:





Collaboration diagram for pursuit:



## Public Member Functions

- [pursuit\(\)](#)

*default constructor.*

## Static Public Member Functions

- static void `loop` ()  
*pursuing scenario loop function*

## Additional Inherited Members

### 6.15.1 Detailed Description

Definition at line 14 of file `pursuit.h`.

### 6.15.2 Constructor & Destructor Documentation

#### 6.15.2.1 `pursuit()`

```
pursuit::pursuit ( )
```

default constructor.

Definition at line 31 of file `pursuit.cpp`.

```
32 {
33     name = "pursuit";
34     createAgent(STATIC, nullptr, nullptr, nullptr);
35     callback = reinterpret_cast <void(*) ()> ( (void *)(&loop) );
36 }
```

### 6.15.3 Member Function Documentation

#### 6.15.3.1 `loop()`

```
void pursuit::loop ( ) [static]
```

pursuing scenario loop function

#### Note

opengl callback forces that function to be static

Definition at line 15 of file `pursuit.cpp`.

```
16 {
17     for(auto it = agents.begin(); it < agents.end(); it++){
18         if((*it).name == "gazelle"){
19             (*it).targetPoint = view.getMousePosition();
20             (*it).force = behavior.seek(*it);
21         }
22         else{//lion
23             (*it).force = behavior.pursuit(agents, *it, view, "gazelle");
24         }
25         (*it).arrive = true;
26     }
27     refresh();
28 }
29 }
```

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

- `include/pursuit.h`
- `src/pursuit.cpp`

## 6.16 pvector Class Reference

```
#include <pvector.h>
```

Collaboration diagram for pvector:

pvector
+ x + y
+ pvector() + pvector() + magnitude() + normalize() + div() + mul() + add() + limit() + getAngle() + dotProduct() and 8 more...

### Public Member Functions

- [pvector](#) ()  
*default constructor*
- [pvector](#) (float [x](#), float [y](#))  
*constructor*
- float [magnitude](#) ()  
*calculates magnitude of the vector*
- [pvector](#) & [normalize](#) ()  
*normalize*
- void [div](#) (float [i](#))  
*vector division*
- void [mul](#) (float [i](#))  
*vector multiplication*
- void [add](#) ([pvector](#) [p](#))  
*addition of vectors*
- void [limit](#) (float [limit](#))  
*vector limitation*
- float [getAngle](#) ()  
*calculates vector angle*
- float [dotProduct](#) ([pvector](#) [v](#))  
*dot product of two vectors*
- float [angleBetween](#) ([pvector](#) [v](#))

- angle calculation between two vectors*
- void `print` (const string &s)
- debug function*
- `pvector operator+=` (`pvector` const &obj)
- overloaded += operator*
- `pvector operator+` (`pvector` const &obj)
- overloaded + operator*
- `pvector operator-` (`pvector` const &obj)
- overloaded - operator*
- `pvector operator-` (`point` const &obj)
- overloaded - operator*
- `pvector operator+` (`point` const &obj)
- overloaded + operator*
- bool `operator==` (`pvector` const &obj)
- overloaded == operator*

## Public Attributes

- float `x`
- x magnitude of the vector*
- float `y`
- y magnitude of the vector*

### 6.16.1 Detailed Description

Definition at line 17 of file `pvector.h`.

### 6.16.2 Constructor & Destructor Documentation

#### 6.16.2.1 `pvector()` [1/2]

```
pvector::pvector ( )
```

default constructor

See also

`pvector(float x, float y)`

Definition at line 35 of file `pvector.cpp`.

```
36 {
37
38 }
```

#### 6.16.2.2 `pvector()` [2/2]

```
pvector::pvector (
    float x,
    float y )
```

constructor

## Parameters

$x$	magnitude of the vector
$y$	magnitude of the vector

## See also

[pvector\(\)](#)

Definition at line 40 of file pvector.cpp.

```
41 {  
42     this->x = x;  
43     this->y = y;  
44 }
```

## 6.16.3 Member Function Documentation

### 6.16.3.1 add()

```
void pvector::add (  
    pvector p )
```

addition of vectors

## Parameters

$p$	vector to add
-----	---------------

Definition at line 58 of file pvector.cpp.

```
59 {  
60     x = x + p.x;  
61     y = y + p.y;  
62 }
```

### 6.16.3.2 angleBetween()

```
float pvector::angleBetween (  
    pvector v )
```

angle calculation between two vectors

## Parameters

$v$	vector to calculate angle
-----	---------------------------

**Returns**

angle

Definition at line 23 of file pvector.cpp.

```

24 {
25     float angle = this->dotProduct(v) / (this->magnitude() * v.magnitude());
26     angle = acos(angle) * 180 / PI;
27     return angle;
28 }

```

Here is the call graph for this function:



Here is the caller graph for this function:

**6.16.3.3 div()**

```

void pvector::div (
    float i )

```

vector division

**Parameters**

<i>i</i>	scalar value to divide
----------	------------------------

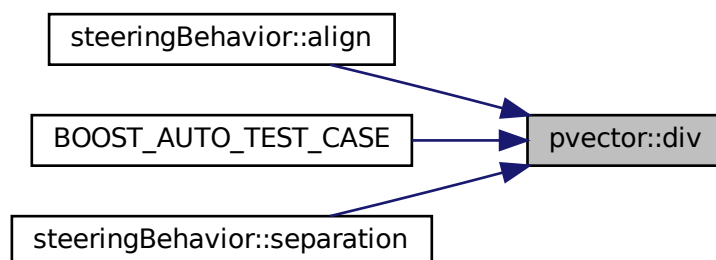
Definition at line 46 of file pvector.cpp.

```

47 {
48     x = x / i;
49     y = y / i;
50 }

```

Here is the caller graph for this function:



#### 6.16.3.4 dotProduct()

```
float pvector::dotProduct (
    pvector v )
```

dot product of two vectors

##### Parameters

<code>v</code>	vector to calculate dot product
----------------	---------------------------------

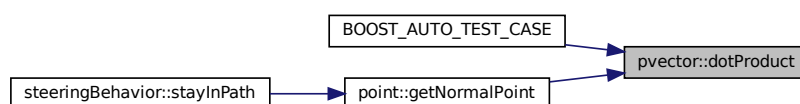
##### Returns

returns scalar dot product

Definition at line 30 of file pvector.cpp.

```
31 {
32     return ((x * v.x) + (y * v.y));
33 }
```

Here is the caller graph for this function:



### 6.16.3.5 getAngle()

```
float pvector::getAngle ( )
```

calculates vector angle

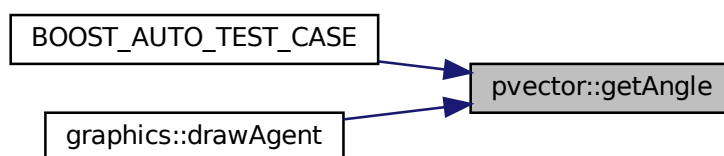
#### Returns

angle

Definition at line 16 of file pvector.cpp.

```
17 {
18     float angle;
19     angle = atan2 (this->y, this->x) * 180 / PI;
20     return angle;
21 }
```

Here is the caller graph for this function:



### 6.16.3.6 limit()

```
void pvector::limit (
    float limit )
```

vector limitation

#### Parameters

<i>limit</i>	value to restrict vector magnitude
--------------	------------------------------------

Definition at line 83 of file pvector.cpp.

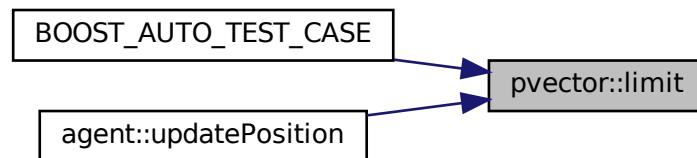
```
84 {
85     this->normalize();
86     this->mul(limit);
87 }
```



Here is the call graph for this function:



Here is the caller graph for this function:



### 6.16.3.7 magnitude()

```
float pvector::magnitude ( )
```

calculates magnitude of the vector

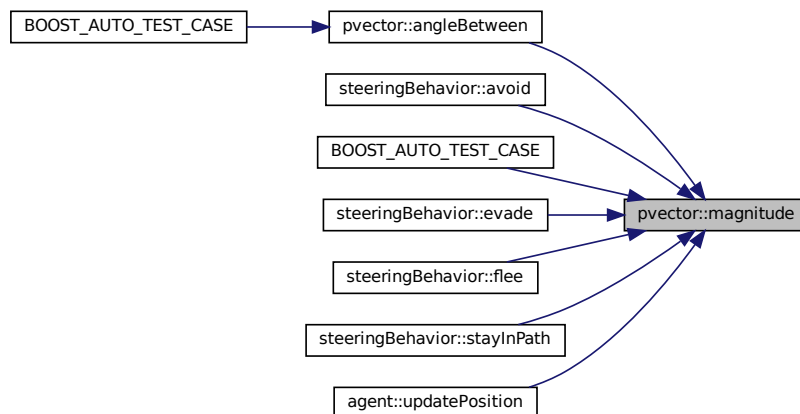
#### Returns

magnitude of the vector

Definition at line 64 of file `pvector.cpp`.

```
65 {  
66     return sqrt((this->x * this->x) + (this->y * this->y));  
67 }
```

Here is the caller graph for this function:



### 6.16.3.8 mul()

```
void pvector::mul (
    float i )
```

vector multiplication

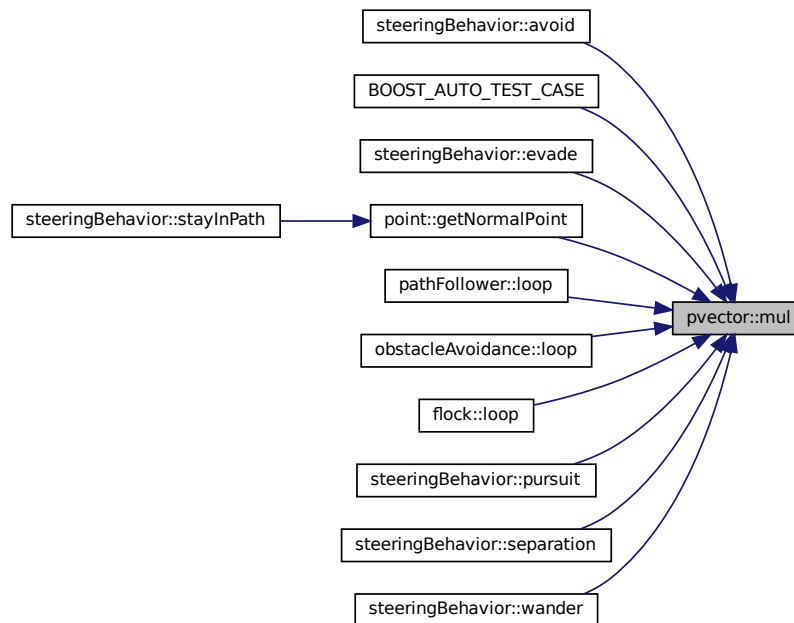
#### Parameters

<i>i</i>	scalar value to multiply
----------	--------------------------

Definition at line 52 of file `pvector.cpp`.

```
53 {
54     x = x * i;
55     y = y * i;
56 }
```

Here is the caller graph for this function:



### 6.16.3.9 normalize()

```
pvector & pvector::normalize ( )
```

normalize

#### Returns

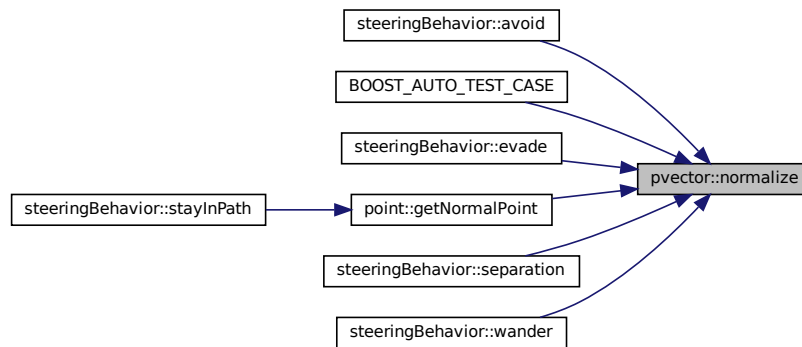
normalized vector

Definition at line 69 of file `pvector.cpp`.

```

70 {
71     float magnitude = this->magnitude();
72     if(magnitude != 0){
73         this->x = this->x / magnitude;
74         this->y = this->y / magnitude;
75     }
76     else{
77         this->x = 0;
78         this->y = 0;
79     }
80     return *this;
81 }
```

Here is the caller graph for this function:



#### 6.16.3.10 operator+() [1/2]

```
pvector pvector::operator+ (
    point const & obj )
```

overloaded + operator

##### Parameters

<i>obj</i>	point to add
------------	--------------

##### Returns

sum

Definition at line 111 of file `pvector.cpp`.

```
112 {
113     pvector res;
114     res.x = x + obj.x;
115     res.y = y + obj.y;
116     return res;
117 }
```

#### 6.16.3.11 operator+() [2/2]

```
pvector pvector::operator+ (
    pvector const & obj )
```

overloaded + operator

## Parameters

<i>obj</i>	vector to add
------------	---------------

## Returns

sum

Definition at line 89 of file pvector.cpp.

```
90 {  
91     pvector res;  
92     res.x = x + obj.x;  
93     res.y = y + obj.y;  
94     return res;  
95 }
```

### 6.16.3.12 operator+=()

```
pvector pvector::operator+= (  
    pvector const & obj )
```

overloaded += operator

## Parameters

<i>obj</i>	vector to add
------------	---------------

## Returns

sum

Definition at line 97 of file pvector.cpp.

```
98 {  
99     x = x + obj.x;  
100     y = y + obj.y;  
101     return *this;  
102 }
```

### 6.16.3.13 operator-() [1/2]

```
pvector pvector::operator- (  
    point const & obj )
```

overloaded - operator

## Parameters

<i>obj</i>	point to subtract
------------	-------------------

**Returns**

difference

Definition at line 119 of file pvector.cpp.

```
120 {  
121     pvector res;  
122     res.x = x - obj.x;  
123     res.y = y - obj.y;  
124     return res;  
125 }
```

**6.16.3.14 operator-() [2/2]**

```
pvector pvector::operator- (  
    pvector const & obj )
```

overloaded - operator

**Parameters**

<i>obj</i>	vector to subtract
------------	--------------------

**Returns**

difference

Definition at line 132 of file pvector.cpp.

```
133 {  
134     pvector res;  
135     res.x = x - obj.x;  
136     res.y = y - obj.y;  
137     return res;  
138 }
```

**6.16.3.15 operator==( )**

```
bool pvector::operator== (  
    pvector const & obj )
```

overloaded == operator

**Parameters**

<i>obj</i>	vector to check if equal
------------	--------------------------

**Returns**

comparison result

Definition at line 104 of file pvector.cpp.

```
105 {  
106     if (x == obj.x && y == obj.y)  
107         return true;  
108     return false;  
109 }
```

### 6.16.3.16 print()

```
void pvector::print (  
    const string & s )
```

debug function

#### Parameters

s	identification text
---	---------------------

Definition at line 127 of file pvector.cpp.

```
128 {  
129     cout << s << " " << x << " " << y << endl;  
130 }
```

## 6.16.4 Member Data Documentation

### 6.16.4.1 x

```
float pvector::x
```

x magnitude of the vector

Definition at line 140 of file pvector.h.

### 6.16.4.2 y

```
float pvector::y
```

y magnitude of the vector

Definition at line 145 of file pvector.h.

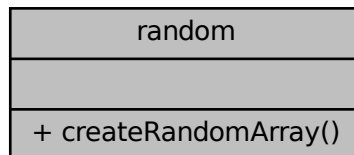
The documentation for this class was generated from the following files:

- [include/pvector.h](#)
- [src/pvector.cpp](#)

## 6.17 random Class Reference

```
#include <random.h>
```

Collaboration diagram for random:



### Static Public Member Functions

- static void [createRandomArray](#) (int \*arr, int size)  
*random array generation*

#### 6.17.1 Detailed Description

Definition at line 9 of file random.h.

#### 6.17.2 Member Function Documentation

##### 6.17.2.1 createRandomArray()

```
void random::createRandomArray (
    int * arr,
    int size ) [static]
```

random array generation

##### Parameters

<i>arr</i>	struct that includes random values
<i>size</i>	of the array

Definition at line 14 of file random.cpp.

```
14                                     {
15     srand(time(NULL));
```



```
16     for(int i=0; i<size; i++)
17         arr[i] = i+1;
18
19     for (int i=0; i < size; i++){
20         int r = rand() % size;
21         swap(arr[i], arr[r]);
22     }
23 }
```

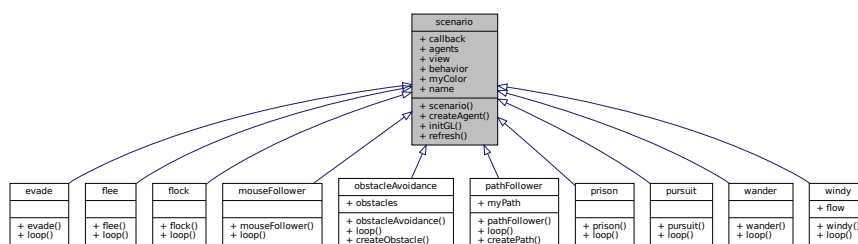
The documentation for this class was generated from the following files:

- include/[random.h](#)
- src/[random.cpp](#)

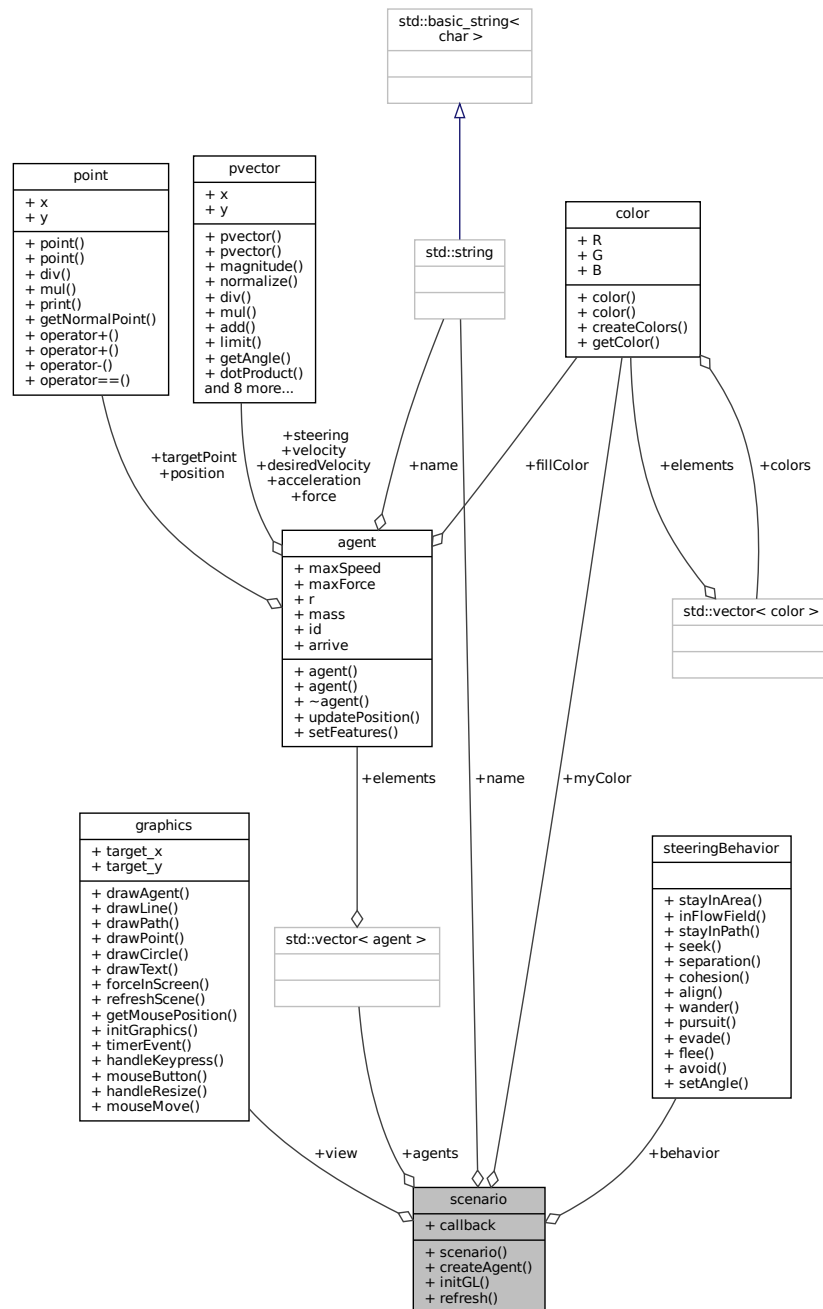
## 6.18 scenario Class Reference

```
#include <scenario.h>
```

Inheritance diagram for scenario:



Collaboration diagram for scenario:



## Public Member Functions

- `scenario ()`  
*default constructor.*
- `void createAgent (int type, int *count, float *force, float *speed)`  
*agent creation*
- `void initGL (int *argv, char **argc)`  
*graphics initialization*

## Static Public Member Functions

- static void [refresh](#) ()  
*refreshes all items*

## Public Attributes

- void(\* [callback](#) )()  
*openGL screen refresh callback function, used as main loop in derived classes*

## Static Public Attributes

- static vector< [agent](#) > [agents](#)  
*structure stores agents*
- static [graphics](#) [view](#)  
*graphics instance used*
- static [steeringBehavior](#) [behavior](#)  
*behavior instance used*
- static [color](#) [myColor](#)  
*color instance used*
- static string [name](#)  
*scenario name*

### 6.18.1 Detailed Description

Definition at line 19 of file scenario.h.

### 6.18.2 Constructor & Destructor Documentation

#### 6.18.2.1 scenario()

```
scenario::scenario ( )
```

default constructor.

Definition at line 28 of file scenario.cpp.

```
29 {
30     srand(time(NULL));
31     myColor.createColors();
32     view = graphics();
33 }
```

### 6.18.3 Member Function Documentation

#### 6.18.3.1 createAgent()

```
void scenario::createAgent (
    int type,
    int * count,
    float * force,
    float * speed )
```

agent creation

## Parameters

<i>type</i>	type of creation method
<i>count</i>	number of agents to be created
<i>force</i>	max force of agents to be created
<i>speed</i>	max speed of agents to be created

Definition at line 109 of file scenario.cpp.

```

110 {
111     if(type == TROOP){
112         createTroop(*count);
113     }
114     else if(type == RANDOM){
115         createRandomAgents(*count, *force, *speed);
116     }
117     else if(type == STATIC){
118         createStaticAgents();
119     }
120     else{
121         //error message
122     }
123 }
```

### 6.18.3.2 initGL()

```

void scenario::initGL (
    int * argv,
    char ** argc )
```

graphics initialization

## Parameters

<i>argv</i>	list of user arguments
<i>argc</i>	number of user arguments

Definition at line 22 of file scenario.cpp.

```

23 {
24     view.initGraphics(argc, argv, callback);
25 }
```

Here is the caller graph for this function:



### 6.18.3.3 refresh()

```
void scenario::refresh ( ) [static]
```

refreshes all items

#### Note

opengl callback forces that function to be static

Definition at line 35 of file scenario.cpp.

```
36 {  
37     point textPosition = point(-34, 32.25);  
38     for(auto it = agents.begin(); it < agents.end(); it++){  
39         (*it).updatePosition((*it).arrive);  
40         view.drawAgent(*it, (*it).fillColor);  
41     }  
42  
43     view.drawText(name, textPosition);  
44     view.refreshScene();  
45 }
```

Here is the call graph for this function:



## 6.18.4 Member Data Documentation

### 6.18.4.1 agents

```
vector< agent > scenario::agents [static]
```

structure stores agents

#### Note

opengl callback forces that function to be static

Definition at line 52 of file scenario.h.

#### 6.18.4.2 behavior

```
steeringBehavior scenario::behavior [static]
```

behavior instance used

##### Note

opengl callback forces that function to be static

Definition at line 64 of file scenario.h.

#### 6.18.4.3 callback

```
void(* scenario::callback) ()
```

OpenGL screen refresh callback function, used as main loop in derived classes

Definition at line 81 of file scenario.h.

#### 6.18.4.4 myColor

```
color scenario::myColor [static]
```

color instance used

##### Note

opengl callback forces that function to be static

Definition at line 70 of file scenario.h.

#### 6.18.4.5 name

```
string scenario::name [static]
```

scenario name

##### Note

opengl callback forces that function to be static

Definition at line 76 of file scenario.h.

#### 6.18.4.6 view

```
graphics scenario::view [static]
```

graphics instance used

#### Note

opengl callback forces that function to be static

Definition at line 58 of file scenario.h.

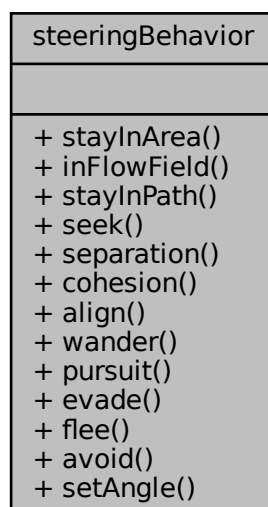
The documentation for this class was generated from the following files:

- include/scenario.h
- src/scenario.cpp

## 6.19 steeringBehavior Class Reference

```
#include <steeringBehavior.h>
```

Collaboration diagram for steeringBehavior:



## Public Member Functions

- `pvector stayInArea` (`agent &agent`, `int turnPoint`)  
*gets reflection force*
- `pvector inFlowField` (`agent &agent`, `flowField &flow`)  
*gets flow field force*
- `pvector stayInPath` (`agent &agent`, `path &path`, `graphics view`)  
*gets force to follow path*
- `pvector seek` (`agent &agent`)  
*force to seek*
- `pvector separation` (`vector< agent > agents`, `agent &agent`)  
*force to separate*
- `pvector cohesion` (`vector< agent > boids`, `agent &agent`)  
*force to cohesion*
- `pvector align` (`vector< agent > boids`, `agent &agent`)  
*force to align*
- `pvector wander` (`agent &agent`)  
*force to wander*
- `pvector pursuit` (`vector< agent > boids`, `agent &pursuer`, `graphics view`, `string name`)  
*force to pursue*
- `pvector evade` (`vector< agent > boids`, `agent &evader`, `graphics view`, `string name`)  
*force to evade*
- `pvector flee` (`agent &agent`, `graphics &view`, `point p`)  
*force to flee*
- `pvector avoid` (`vector< obstacle > obstacles`, `agent &agent`)  
*force to avoid*
- `void setAngle` (`pvector &p`, `float angle`)  
*applies angle on vector*

### 6.19.1 Detailed Description

Definition at line 35 of file `steeringBehavior.h`.

### 6.19.2 Member Function Documentation

#### 6.19.2.1 align()

```
pvector steeringBehavior::align (
    vector< agent > boids,
    agent & agent )
```

force to align

#### Parameters

<i>agent</i>	to be aligned
<i>boids</i>	list of all the agents



## Returns

force to be applied

Definition at line 119 of file steeringBehavior.cpp.

```

120 {
121     float neighborDist = 30;
122     pvector sum {0,0};
123     int count = 0;
124     for(auto it = boids.begin(); it < boids.end(); it++){
125         float d = (agent.position - (*it).position).magnitude();
126         if( (d > 0) && (d < neighborDist) ){
127             sum += (*it).velocity;
128             count++;
129         }
130     }
131     if(count > 0){
132         sum.div(count);
133         sum.normalize().mul(agent.maxSpeed);
134         agent.steering = sum - agent.velocity;
135         return agent.steering;
136     }
137     return pvector(0,0);
138 }
```

Here is the call graph for this function:



### 6.19.2.2 avoid()

```

pvector steeringBehavior::avoid (
    vector< obstacle > obstacles,
    agent & agent )
```

force to avoid

## Parameters

<i>agent</i>	agent that will avoid from obstacles
<i>obstacles</i>	list of all existing objects

## Returns

force to be applied

Definition at line 183 of file steeringBehavior.cpp.

```

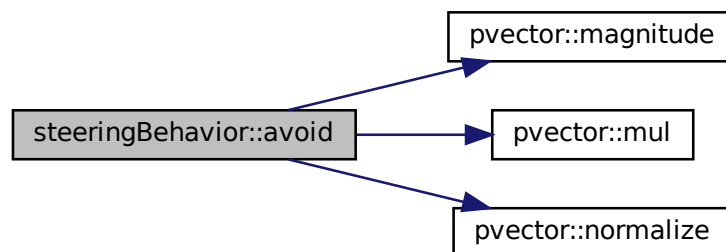
184 {
185     float dynamic_length = agent.velocity.magnitude() / agent.maxSpeed;
```

```

186   pvector vel = agent.velocity;
187   vel.normalize().mul(dynamic_length);
188   pvector ahead = vel + agent.position;
189   vel.mul(6);
190   pvector ahead2 = vel + agent.position;
191   //view.drawPoint(point(ahead.x, ahead.y));
192   //view.drawPoint(point(ahead2.x, ahead2.y));
193
194   for(auto it = obstacles.begin(); it < obstacles.end(); it++){
195       float dist = (ahead - (*it).p).magnitude();
196       float dist2 = (ahead2 - (*it).p).magnitude();
197       if(dist < (*it).r + 2 || dist2 < (*it).r + 2){
198           pvector avoidance = ahead - (*it).p;
199           avoidance.normalize().mul(20);
200           /*a = point(avoidance.x, avoidance.y);
201           view.drawLine(agent.position, agent.position + a, color(0,1,0));*/
202           return avoidance;
203       }
204   }
205   return pvector(0,0);
206 }

```

Here is the call graph for this function:



### 6.19.2.3 cohesion()

```

pvector steeringBehavior::cohesion (
    vector< agent > boids,
    agent & agent )

```

force to cohesion

#### Parameters

<i>agent</i>	to go to center of other agents, with specified distance
<i>boids</i>	list of all the agents

#### Returns

force to be applied

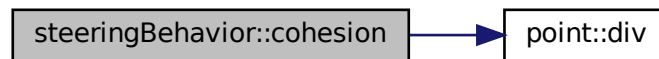
Definition at line 140 of file steeringBehavior.cpp.

```

141 {
142     float neighborDist = 20;
143     point sum {0,0};
144     int count = 0;
145     for(auto it = boids.begin(); it < boids.end(); it++){
146         float d = (agent.position - (*it).position).magnitude();
147         if( (d > 0) && (d < neighborDist) ){
148             sum = sum + (*it).position;
149             count++;
150         }
151     }
152     if(count > 0){
153         sum.div(count);
154         agent.targetPoint = sum;
155         return seek(agent);
156     }
157     return pvector(0,0);
158 }

```

Here is the call graph for this function:



#### 6.19.2.4 evade()

```

pvector steeringBehavior::evade (
    vector< agent > boids,
    agent & evader,
    graphics view,
    string name )

```

force to evade

##### Parameters

<i>evader</i>	agent that will escape
<i>view</i>	used for debugging
<i>boids</i>	list of all the agents
<i>name</i>	other agent to evade

##### Returns

force to be applied

Definition at line 47 of file steeringBehavior.cpp.

```

48 {
49     agent target;
50     for(auto it = boids.begin(); it < boids.end(); it++){
51         if((*it).name == name){
52             target = *it;

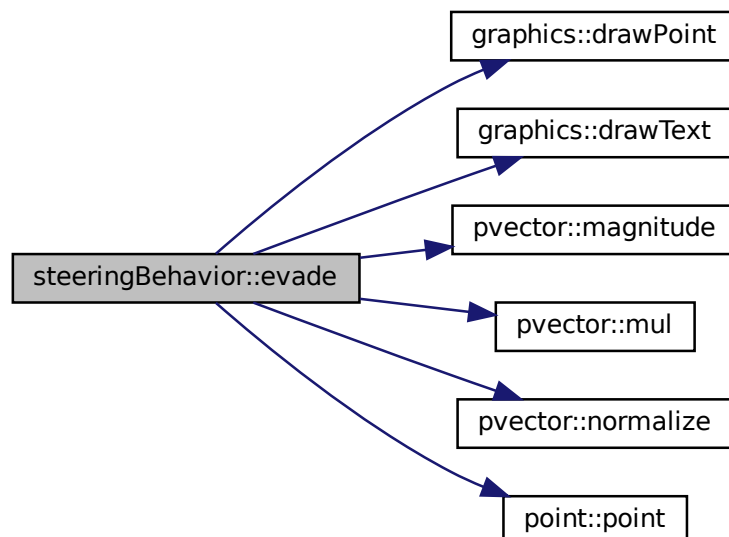
```

```

53     }
54 }
55
56 point p = point(evader.position.x + 2, evader.position.y - 2);
57 view.drawText(evader.name, p);
58 p = point(target.position.x + 2, target.position.y - 2);
59 view.drawText(target.name, p);
60
61 pvector targetVel = target.velocity;
62 targetVel.mul(5); //TODO: magic number
63
64 point futurePos = target.position + targetVel;
65 view.drawPoint(futurePos);
66
67 pvector dist = evader.position - futurePos;
68 dist.normalize().mul( 1 / dist.magnitude() );
69
70 evader.targetPoint = evader.position + dist;
71 return flee(evader, view, futurePos);
72 }

```

Here is the call graph for this function:



### 6.19.2.5 flee()

```

pvector steeringBehavior::flee (
    agent & agent,
    graphics & view,
    point p )

```

force to flee

#### Parameters

<i>agent</i>	agent that will flee
<i>view</i>	used for debugging
<i>p</i>	point that agent flees

**Returns**

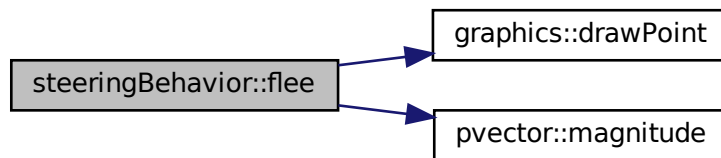
force to be applied

Definition at line 28 of file steeringBehavior.cpp.

```

29 {
30     int radius = 15;
31
32     pvector dist = agent.targetPoint - p;
33     view.drawPoint(agent.targetPoint);
34
35     if(dist.magnitude() < radius){
36         agent.arrive = false;
37         agent.desiredVelocity = agent.position - p;
38     }
39     else{
40         agent.arrive = true;
41         agent.desiredVelocity = agent.targetPoint - agent.position;
42     }
43     agent.steering = agent.desiredVelocity - agent.velocity;
44     return agent.steering;
45 }
```

Here is the call graph for this function:

**6.19.2.6 inFlowField()**

```

pvector steeringBehavior::inFlowField (
    agent & agent,
    flowField & flow )
```

gets flow field force

**Parameters**

<i>agent</i>	unit to apply flow field
<i>flow</i>	field

**Returns**

force to be applied

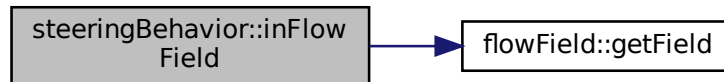
Definition at line 238 of file steeringBehavior.cpp.

```

239 {
240     //pos_x, pos_y must be non negative integer
241     int pos_x = abs((int)agent.position.x) % WIDTH;
242     int pos_y = abs((int)agent.position.y) % HEIGHT;
243     //TODO: modification required for non uniform fields
244     return flow.getField(pos_x, pos_y);
245 }

```

Here is the call graph for this function:



### 6.19.2.7 pursuit()

```

pvector steeringBehavior::pursuit (
    vector< agent > boids,
    agent & pursuer,
    graphics view,
    string name )

```

force to pursue

#### Parameters

<i>pursuer</i>	agent that will follow specified agent
<i>view</i>	used for debugging
<i>boids</i>	list of all the agents
<i>name</i>	other agent to pursue

#### Returns

force to be applied

Definition at line 74 of file steeringBehavior.cpp.

```

75 {
76     agent target;
77     for(auto it = boids.begin(); it < boids.end(); it++){
78         if((*it).name == name){
79             target = *it;
80         }
81     }
82
83     point p = point(target.position.x + 2, target.position.y - 2);
84     view.drawText(target.name, p);
85     p = point(pursuer.position.x + 2, pursuer.position.y - 2);
86     view.drawText(pursuer.name, p);
87
88     float dist = (target.position - pursuer.position).magnitude();

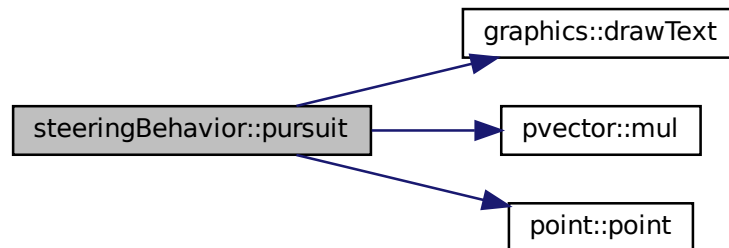
```

```

89     float t = dist / target.maxSpeed;
90
91     pvector targetVel = target.velocity;
92     targetVel.mul(t);
93     point futurePos = target.position + targetVel;
94     pursuer.targetPoint = futurePos;
95     return seek(pursuer);
96 }

```

Here is the call graph for this function:



### 6.19.2.8 seek()

```

pvector steeringBehavior::seek (
    agent & agent )

```

force to seek

#### Parameters

<i>agent</i>	that will go to specific target point
--------------	---------------------------------------

#### Returns

force to be applied

Definition at line 208 of file `steeringBehavior.cpp`.

```

209 {
210     agent.desiredVelocity = agent.targetPoint - agent.position;
211     agent.steering = agent.desiredVelocity - agent.velocity;
212     return agent.steering;
213 }

```

### 6.19.2.9 separation()

```

pvector steeringBehavior::separation (
    vector< agent > agents,
    agent & agent )

```

force to separate



## Parameters

<i>agent</i>	agent that will be stayed away
<i>agents</i>	list of all the agents

## Returns

force to be applied

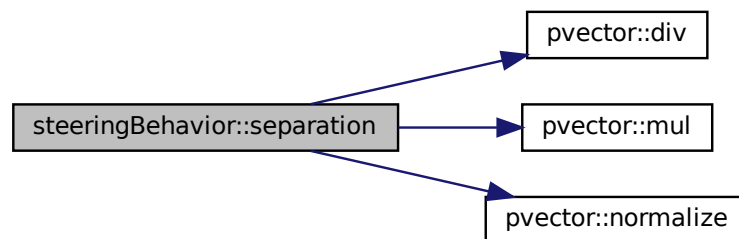
Definition at line 160 of file steeringBehavior.cpp.

```

161 {
162     float desiredSeparation = 5;
163     pvector sum = pvector(0,0);
164     int count = 0;
165     for(auto it = agents.begin(); it < agents.end(); it++){
166         float d = (agent.position - (*it).position).magnitude();
167         if( (d > 0) && (d < desiredSeparation) ){
168             pvector diff = agent.position - (*it).position;
169             diff.normalize().div(d);
170             sum = sum + diff;
171             count++;
172         }
173     }
174     if(count > 0){
175         sum.div(count);
176         sum.normalize().mul(agent.maxSpeed);
177         agent.steering = sum - agent.velocity;
178         return agent.steering;
179     }
180     return pvector(0,0);
181 }

```

Here is the call graph for this function:



## 6.19.2.10 setAngle()

```

void steeringBehavior::setAngle (
    pvector & p,
    float angle )

```

applies angle on vector

**Parameters**

<i>angle</i>	that will be set
<i>p</i>	vector that angle will be applied

Definition at line 22 of file steeringBehavior.cpp.

```

23 {
24     p.x = cos ( angle * PI / 180.0 );
25     p.y = sin ( angle * PI / 180.0 );
26 }
```

**6.19.2.11 stayInArea()**

```

pvector steeringBehavior::stayInArea (
    agent & agent,
    int turnPoint )
```

gets reflection force

**Parameters**

<i>agent</i>	unit to check
<i>turnpoint</i>	defines border to apply force

**Returns**

force to be applied

Definition at line 247 of file steeringBehavior.cpp.

```

248 {
249     if(agent.position.x >= turnPoint){
250         agent.desiredVelocity = pvector( -agent.maxSpeed, agent.velocity.y );
251         agent.steering = agent.desiredVelocity - agent.velocity;
252         return agent.steering;
253     }
254     else if(agent.position.x <= -turnPoint){
255         agent.desiredVelocity = pvector( agent.maxSpeed, agent.velocity.y );
256         agent.steering = agent.desiredVelocity - agent.velocity;
257         return agent.steering;
258     }
259     else if(agent.position.y >= turnPoint){
260         agent.desiredVelocity = pvector( agent.velocity.x, -agent.maxSpeed );
261         agent.steering = agent.desiredVelocity - agent.velocity;
262         return agent.steering;
263     }
264     else if(agent.position.y <= -turnPoint){
265         agent.desiredVelocity = pvector( agent.velocity.x, agent.maxSpeed );
266         agent.steering = agent.desiredVelocity - agent.velocity;
267         return agent.steering;
268     }
269     return pvector(0,0);
270 }
```

**6.19.2.12 stayInPath()**

```

pvector steeringBehavior::stayInPath (
    agent & agent,
```

```

    path & path,
    graphics view )

```

gets force to follow path

#### Parameters

<i>agent</i>	to follow the pathk
<i>path</i>	to follow
<i>view</i>	used for debugging

#### Returns

force to be applied

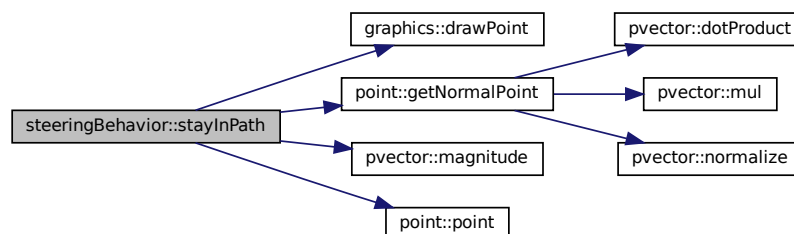
Definition at line 215 of file steeringBehavior.cpp.

```

216 {
217     float worldRecord = 1000000;
218     point normalPoint, predictedPos, start, end;
219     pvector distance;
220     for(auto it = path.points.begin(); it < path.points.end()-1; it++){
221         start = point ((*it).x, (*it).y);
222         end = point ((*it+1).x, (*it+1).y);
223         predictedPos = agent.position + agent.velocity;
224         normalPoint.getNormalPoint(predictedPos, start, end);
225         if (normalPoint.x < start.x || normalPoint.x > end.x){
226             normalPoint = end;
227         }
228         distance = predictedPos - normalPoint;
229         if (distance.magnitude() < worldRecord){
230             worldRecord = distance.magnitude();
231             agent.targetPoint = end;
232         }
233         view.drawPoint(agent.targetPoint);
234     }
235     return seek(agent);
236 }

```

Here is the call graph for this function:



#### 6.19.2.13 wander()

```

pvector steeringBehavior::wander (
    agent & agent )

```

force to wander

## Parameters

<i>agent</i>	agent that will wander
--------------	------------------------

## Returns

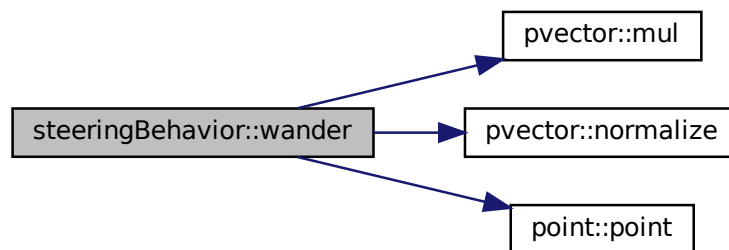
force to be applied

Definition at line 98 of file steeringBehavior.cpp.

```

99 {
100     pvector circleCenter = agent.velocity;
101     circleCenter.normalize().mul(CIRCLE_DISTANCE + CIRCLE_RADIUS);
102
103     int wanderAngle = (rand() % 360);
104     pvector displacement {0, 1};
105     setAngle(displacement, wanderAngle);
106     displacement.mul(CIRCLE_RADIUS);
107
108     agent.desiredVelocity = displacement + circleCenter;
109     agent.steering = agent.desiredVelocity - agent.velocity;
110
111     //move it to the center when it is out of screen
112     if(agent.position.x > WIDTH || agent.position.x < -WIDTH ||
113        agent.position.y > HEIGHT || agent.position.y < -HEIGHT)
114         agent.position = point(0,0);
115
116     return agent.steering;
117 }
```

Here is the call graph for this function:



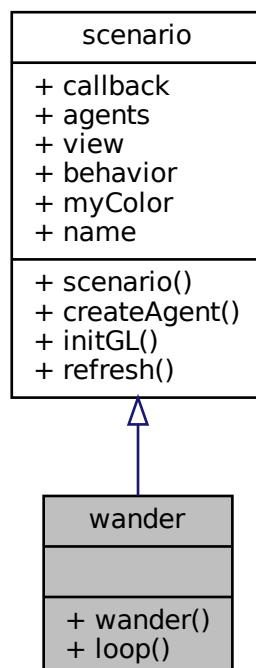
The documentation for this class was generated from the following files:

- include/steeringBehavior.h
- src/steeringBehavior.cpp

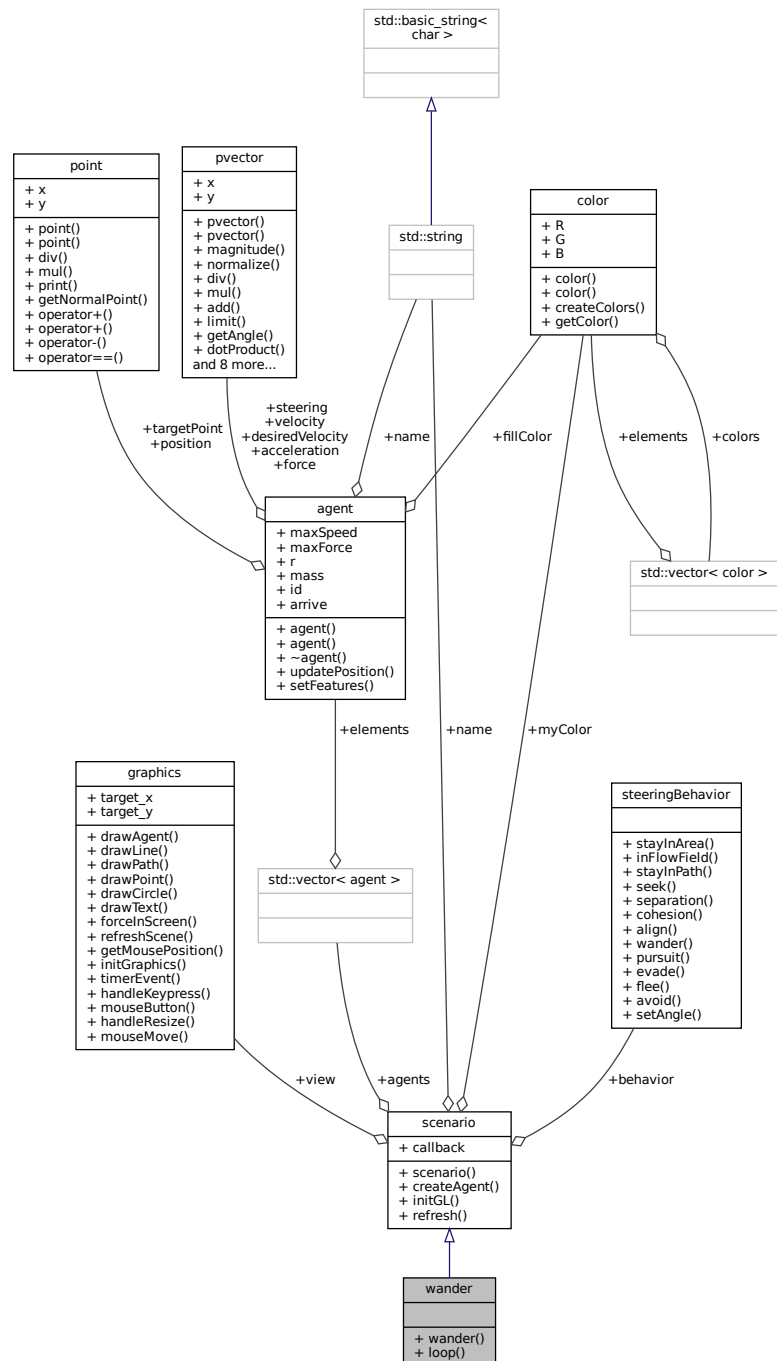
## 6.20 wander Class Reference

```
#include <wander.h>
```

Inheritance diagram for wander:



Collaboration diagram for wander:



## Public Member Functions

- `wander ()`

*default constructor*

## Static Public Member Functions

- static void [loop](#) ()  
*wander scenario loop function*

## Additional Inherited Members

### 6.20.1 Detailed Description

Definition at line 14 of file wander.h.

### 6.20.2 Constructor & Destructor Documentation

#### 6.20.2.1 wander()

```
wander::wander ( )
```

default constructor

**Todo** business logic will be changed

Definition at line 24 of file wander.cpp.

```
25 {
26     int agentCount = 30;
27     float maxForce = 0.3;
28     float maxSpeed = 0.6;
29
30     name = "wandering objects";
31     createAgent(RANDOM, &agentCount, &maxForce, &maxSpeed);
32     callback = reinterpret_cast <void(*)()> ( (void *)(&loop) );
33 }
```

### 6.20.3 Member Function Documentation

#### 6.20.3.1 loop()

```
void wander::loop ( ) [static]
```

wander scenario loop function

#### Note

opengl callback forces that function to be static

Definition at line 15 of file wander.cpp.

```
16 {
17     for(auto it = agents.begin(); it < agents.end(); it++){
18         (*it).force = behavior.wander(*it);
19     }
20
21     refresh();
22 }
```

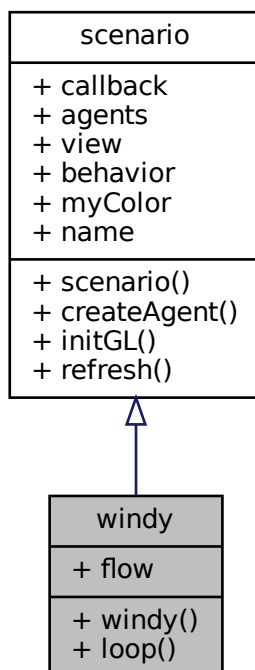
The documentation for this class was generated from the following files:

- include/[wander.h](#)
- src/[wander.cpp](#)

## 6.21 windy Class Reference

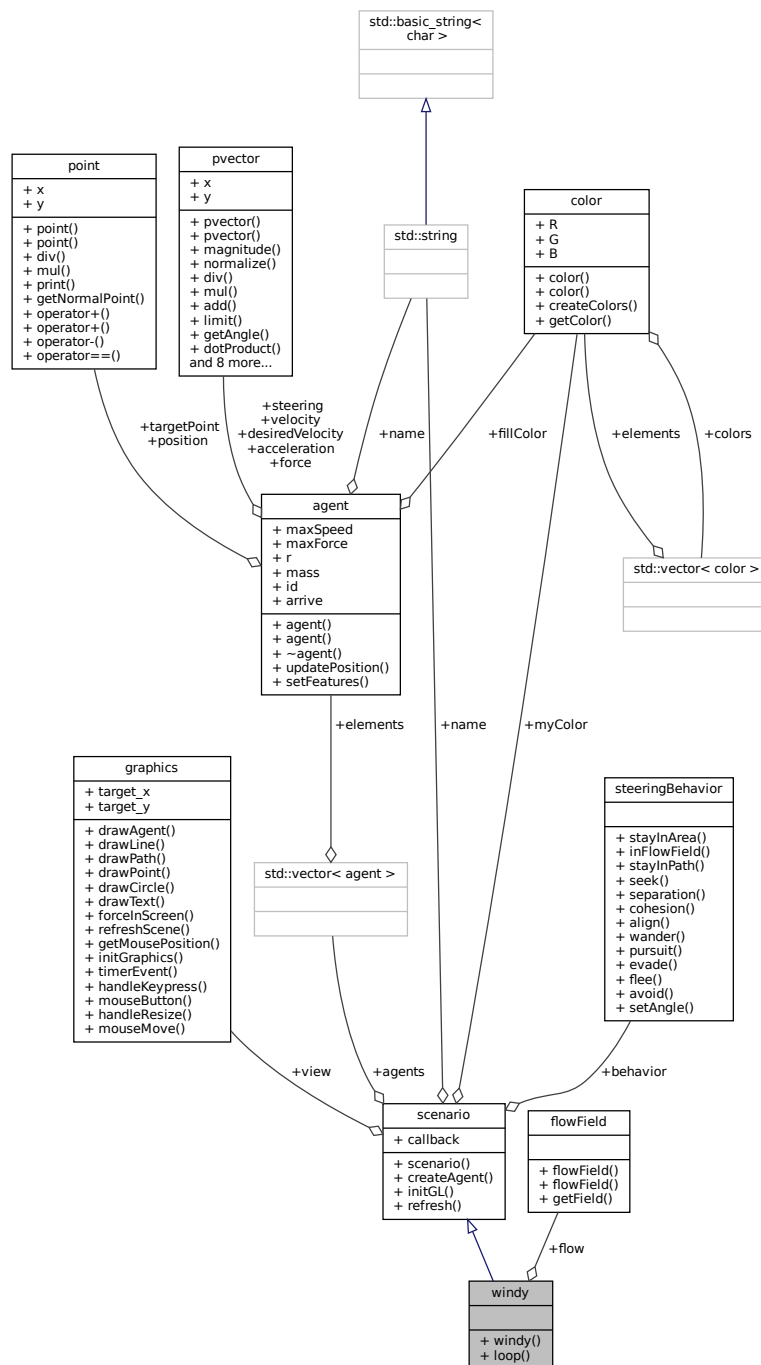
```
#include <windy.h>
```

Inheritance diagram for windy:





Collaboration diagram for windy:



## Public Member Functions

- [windy \(\)](#)

*default constructor.*

## Static Public Member Functions

- static void `loop` ()  
*windy scenario loop function*

## Static Public Attributes

- static `flowField` `flow`  
*flow field used*

## Additional Inherited Members

### 6.21.1 Detailed Description

Definition at line 15 of file windy.h.

### 6.21.2 Constructor & Destructor Documentation

#### 6.21.2.1 windy()

`windy::windy ( )`

default constructor.

Definition at line 29 of file windy.cpp.

```
30 {
31     int agentCount = 30;
32     float maxForce = 0.3;
33     float maxSpeed = 0.6;
34
35     name = "flow field";
36     createAgent(RANDOM, &agentCount, &maxForce, &maxSpeed);
37     callback = reinterpret_cast<void(*)()> ( (void *)(&loop) );
38 }
```

### 6.21.3 Member Function Documentation

#### 6.21.3.1 loop()

`void windy::loop ( ) [static]`

windy scenario loop function

#### Note

opengl callback forces that function to be static

Definition at line 17 of file windy.cpp.

```
18 {
19     for(auto it = agents.begin(); it < agents.end(); it++){
20         flow = flowField(pvector(GRAVITY));
21         (*it).force = behavior.inFlowField(*it, flow);
22
23         flow = flowField(pvector(WIND_WEST));
24         (*it).force += behavior.inFlowField(*it, flow);
25     }
26     refresh();
27 }
```

## 6.21.4 Member Data Documentation

### 6.21.4.1 flow

```
flowField windy::flow [static]
```

flow field used

#### Note

opengl callback forces that function to be static

Definition at line 32 of file windy.h.

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

- include/[windy.h](#)
- src/[windy.cpp](#)



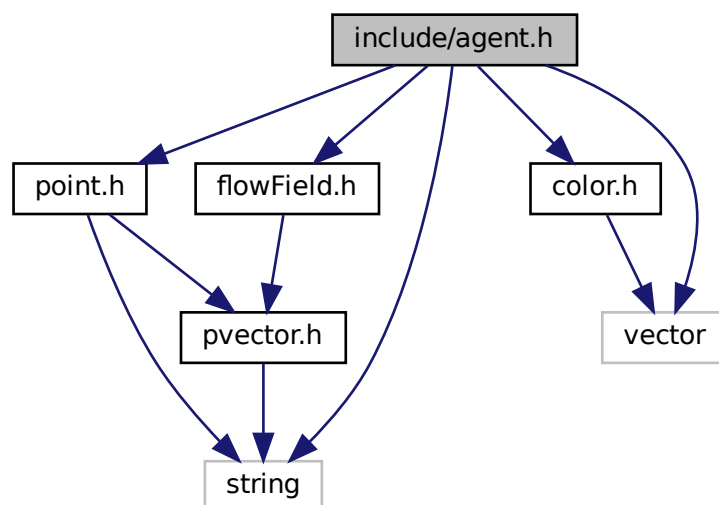
## Chapter 7

# File Documentation

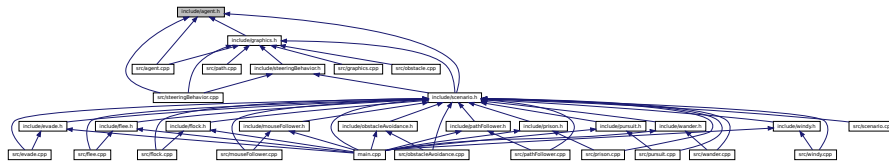
### 7.1 include/agent.h File Reference

agent class defines all agent specifications

```
#include "point.h"  
#include "color.h"  
#include "flowField.h"  
#include <vector>  
#include <string>  
Include dependency graph for agent.h:
```



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



## Classes

- class [agent](#)

### 7.1.1 Detailed Description

agent class defines all agent specifications

Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

Date

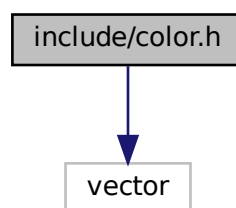
14.05.2021

## 7.2 include/color.h File Reference

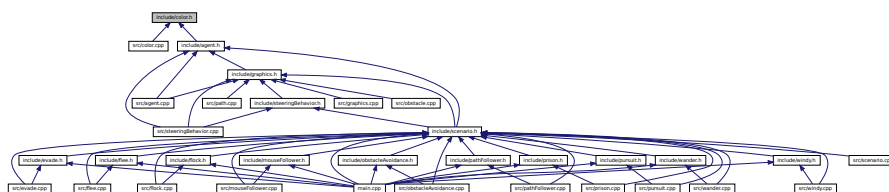
color class used for agent, path, wall etc. color

```
#include <vector>
```

Include dependency graph for color.h:



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



## Classes

- class `color`

## Enumerations

- enum `num` {  
    `BLACK=0, BLUE, GREEN, CYAN,`  
    `RED, MAGENDA, YELLOW, WHITE` }  
    *fundament list of al colors*

### 7.2.1 Detailed Description

color class used for agent, path, wall etc. color

#### Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

#### Date

13.05.2021

### 7.2.2 Enumeration Type Documentation

#### 7.2.2.1 num

enum `num`

fundament list of al colors

#### Enumerator

BLACK	
BLUE	
GREEN	
CYAN	
RED	
MAGENDA	
YELLOW	
WHITE	

Definition at line 17 of file color.h.

```
17 { BLACK=0, BLUE, GREEN, CYAN, RED, MAGENDA, YELLOW, WHITE };
```

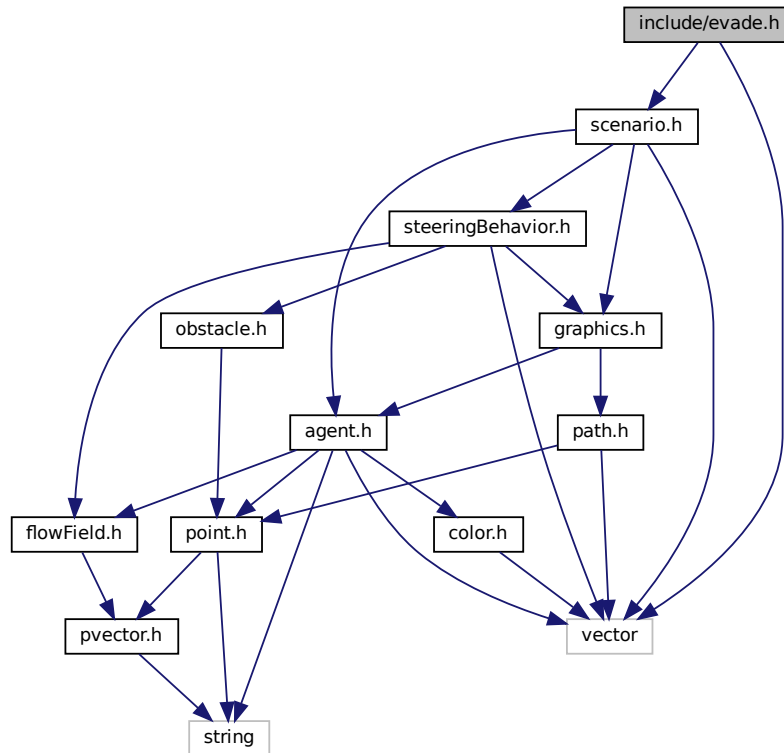
### 7.3 include/evade.h File Reference

evade class inherited from scenario class

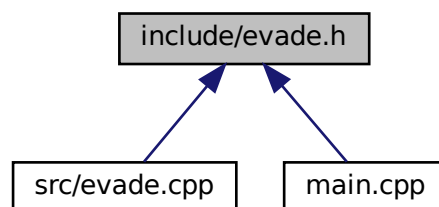
```
#include "scenario.h"
```

```
#include <vector>
```

Include dependency graph for evade.h:



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



#### Classes

- class [evade](#)



### 7.3.1 Detailed Description

evade class inherited from scenario class

Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

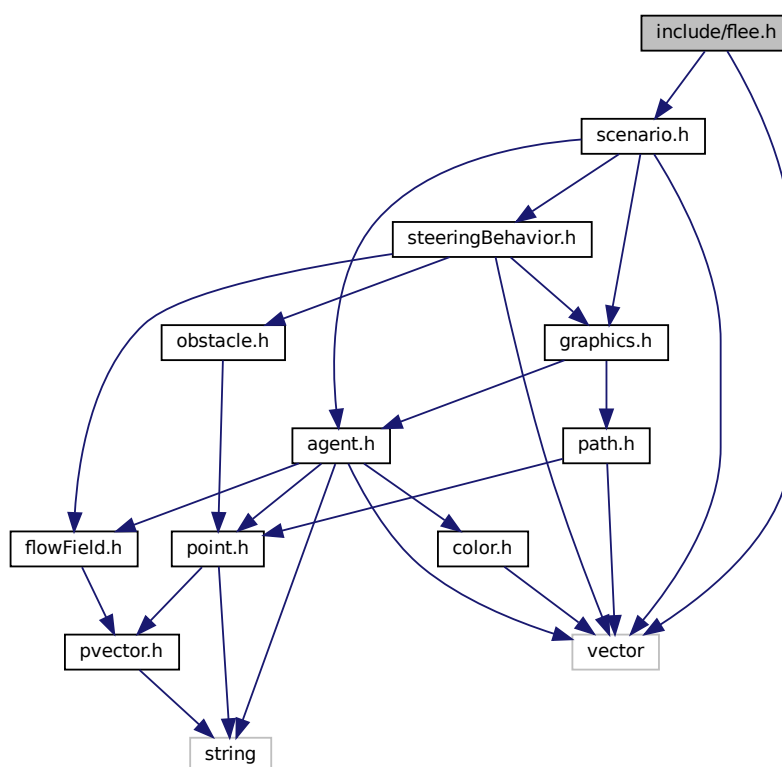
Date

15.05.2021

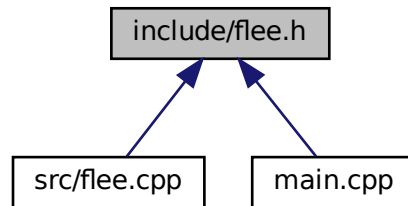
## 7.4 include/flee.h File Reference

agents flee from mouse scenario

```
#include "scenario.h"  
#include <vector>  
Include dependency graph for flee.h:
```



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



## Classes

- class [flee](#)

### 7.4.1 Detailed Description

agents flee from mouse scenario

#### Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

#### Date

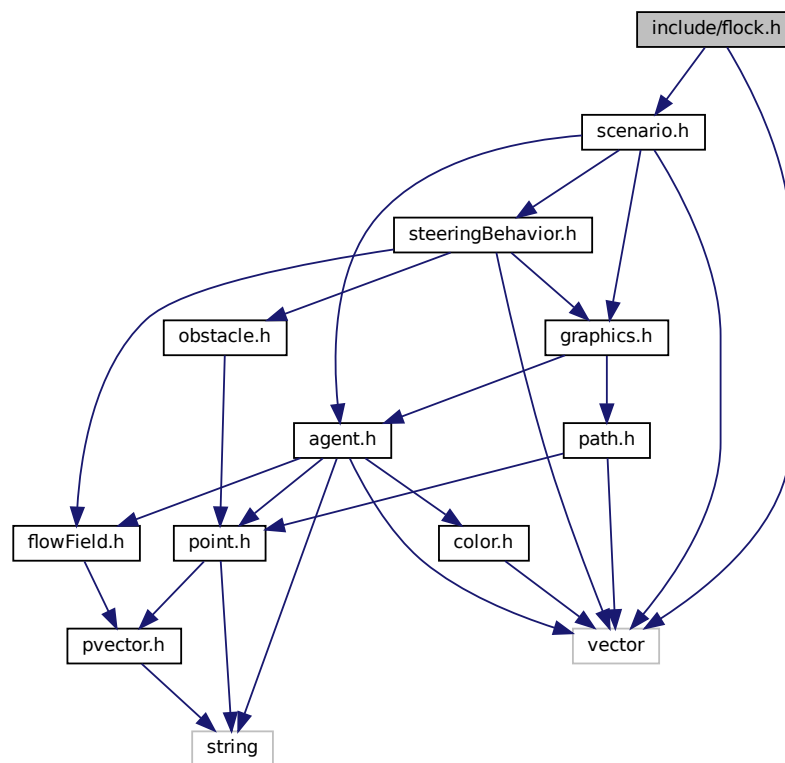
15.05.2021

## 7.5 include/flock.h File Reference

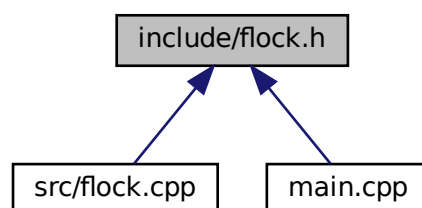
flocking agents scenario

```
#include "scenario.h"
#include <vector>
```

Include dependency graph for flock.h:



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



## Classes

- class [flock](#)

### 7.5.1 Detailed Description

flocking agents scenario

Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

Date

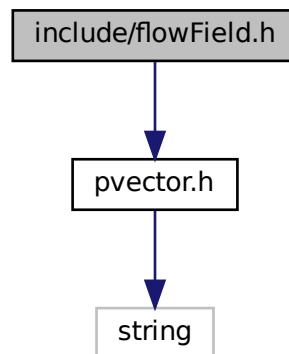
15.05.2021

## 7.6 include/flowField.h File Reference

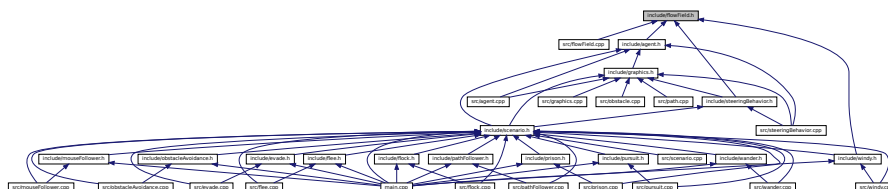
`flowField` class, screen can be filled with a force for each pixel

```
#include "pvector.h"
```

Include dependency graph for `flowField.h`:



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



## Classes

- class `flowField`

## Macros

- `#define FIELD_WIDTH 34`
- `#define FIELD_HEIGHT 34`
- `#define WIND_WEST 0.1, 0.0`
- `#define GRAVITY 0.0, -0.1`

### 7.6.1 Detailed Description

`flowField` class, screen can be filled with a force for each pixel

#### Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

#### Date

13.05.2021

### 7.6.2 Macro Definition Documentation

#### 7.6.2.1 FIELD\_HEIGHT

```
#define FIELD_HEIGHT 34
```

Definition at line 13 of file flowField.h.

#### 7.6.2.2 FIELD\_WIDTH

```
#define FIELD_WIDTH 34
```

Definition at line 12 of file flowField.h.

#### 7.6.2.3 GRAVITY

```
#define GRAVITY 0.0, -0.1
```

Definition at line 16 of file flowField.h.



## Macros

- `#define WIDTH 34`
- `#define HEIGHT 34`
- `#define ESC 27`
- `#define PI 3.14159265`

### 7.7.1 Detailed Description

graphics class, drives openGL

#### Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

#### Date

15.05.2021

### 7.7.2 Macro Definition Documentation

#### 7.7.2.1 ESC

```
#define ESC 27
```

Definition at line 16 of file graphics.h.

#### 7.7.2.2 HEIGHT

```
#define HEIGHT 34
```

Definition at line 14 of file graphics.h.

#### 7.7.2.3 PI

```
#define PI 3.14159265
```

Definition at line 17 of file graphics.h.

### 7.7.2.4 WIDTH

```
#define WIDTH 34
```

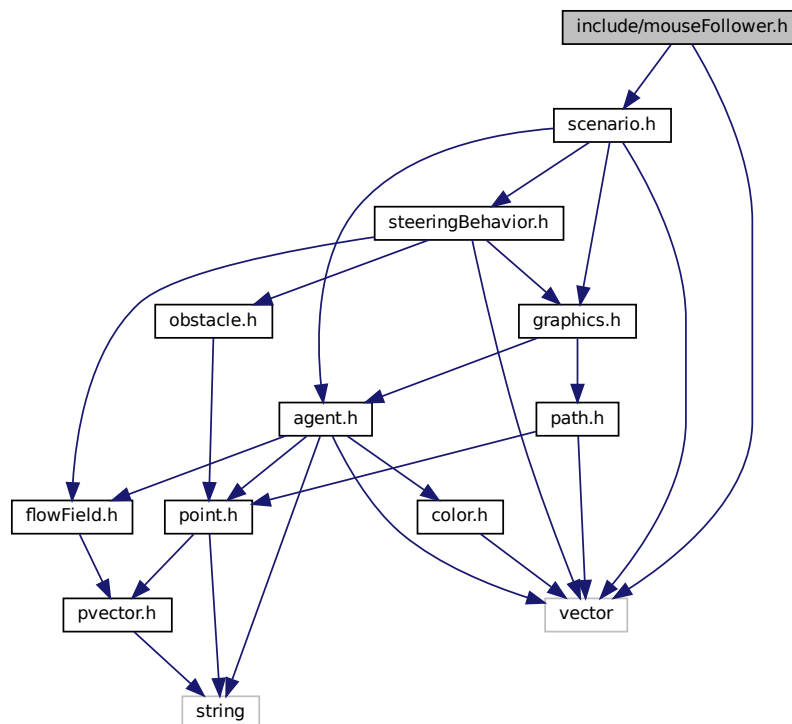
Definition at line 13 of file graphics.h.

## 7.8 include/mouseFollower.h File Reference

agents follow mouse scenario

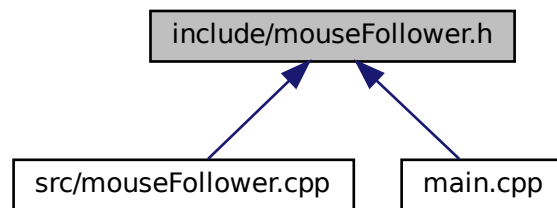
```
#include "scenario.h"
#include <vector>
```

Include dependency graph for mouseFollower.h:





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



## Classes

- class `mouseFollower`

### 7.8.1 Detailed Description

agents follow mouse scenario

#### Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

#### Date

15.05.2021

## 7.9 include/obstacle.h File Reference

circular obstacles for agent avoidance behaviors

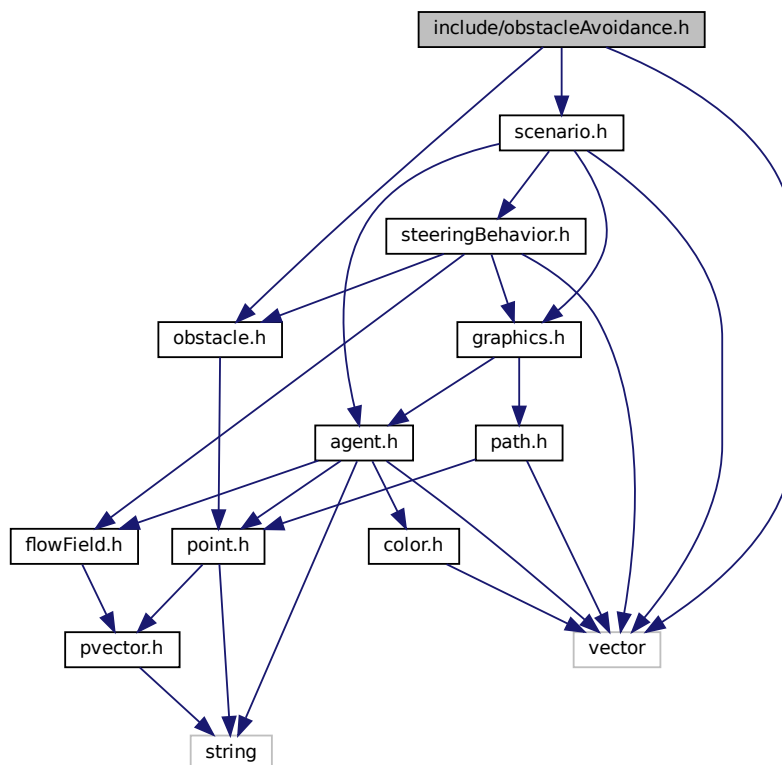


## 7.10 include/obstacleAvoidance.h File Reference

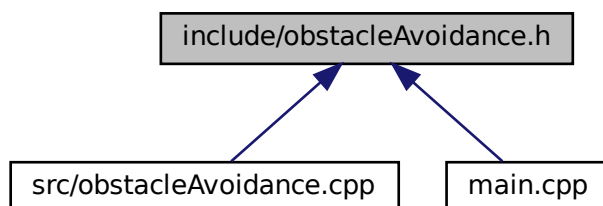
agents avoid from obstacles scenario

```
#include "scenario.h"
#include "obstacle.h"
#include <vector>
```

Include dependency graph for obstacleAvoidance.h:



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



## Classes

- class [obstacleAvoidance](#)

### 7.10.1 Detailed Description

agents avoid from obstacles scenario

Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

Date

15.05.2021

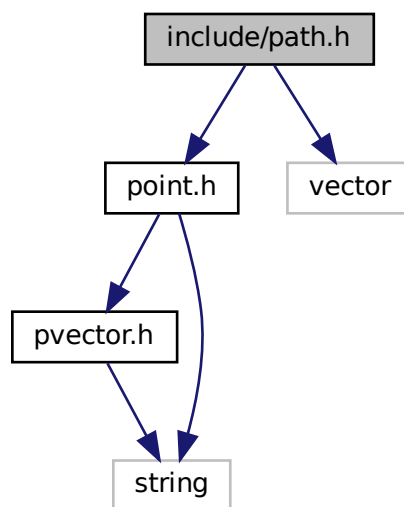
## 7.11 include/path.h File Reference

path class used for path following steering behaviors.

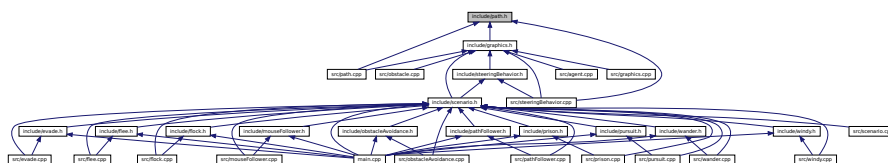
```
#include "point.h"
```

```
#include <vector>
```

Include dependency graph for path.h:



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



## Classes

- class [path](#)

### 7.11.1 Detailed Description

path class used for path following steering behaviors.

#### Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

#### Date

12.05.2021

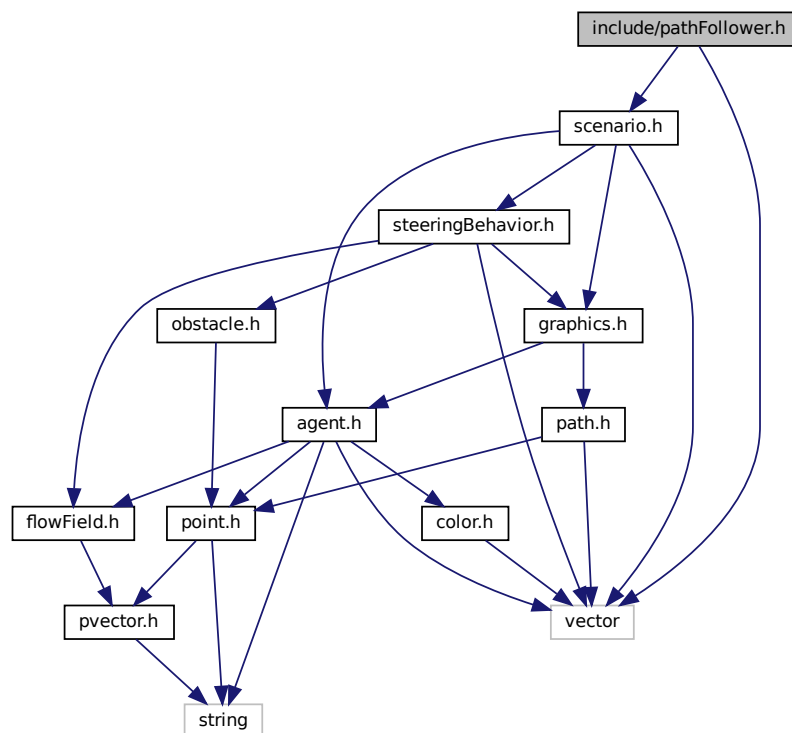
## 7.12 include/pathFollower.h File Reference

path following scenario

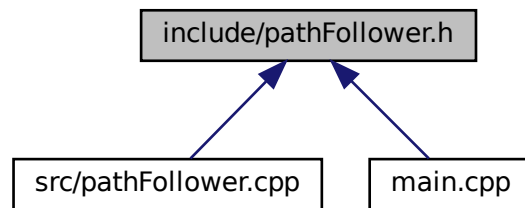
```
#include "scenario.h"
```

```
#include <vector>
```

Include dependency graph for pathFollower.h:



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



## Classes

- class [pathFollower](#)

### 7.12.1 Detailed Description

path following scenario

#### Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

#### Date

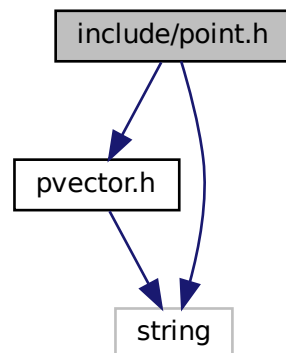
15.05.2021

## 7.13 include/point.h File Reference

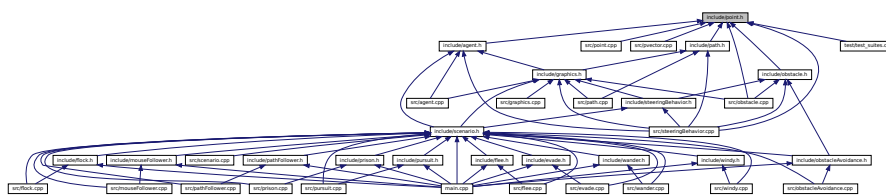
point class used for point operations

```
#include "pvector.h"  
#include <string>
```

Include dependency graph for point.h:



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



## Classes

- class [point](#)

### 7.13.1 Detailed Description

point class used for point operations

#### Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

#### Date

15.05.2021

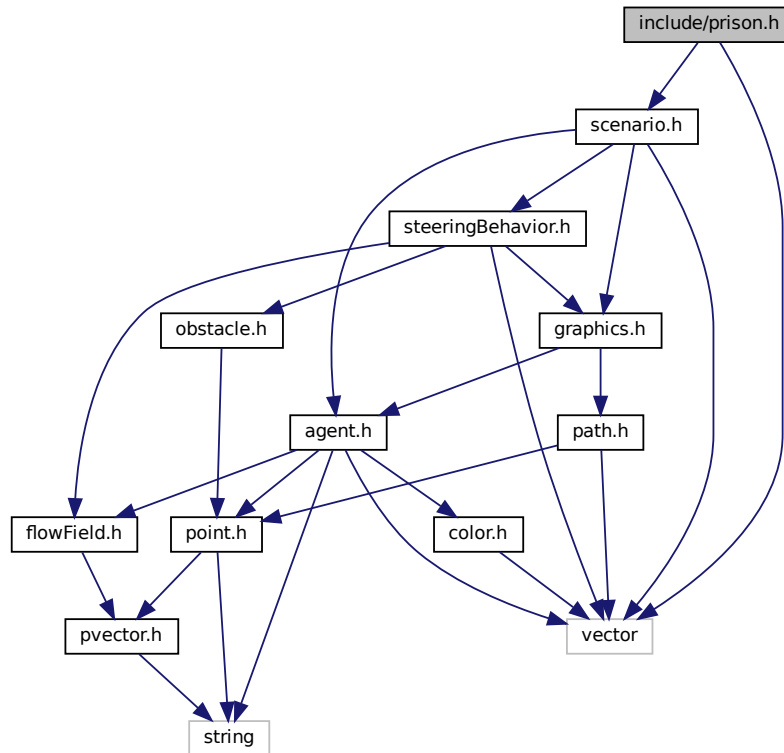
## 7.14 include/prison.h File Reference

agents cant escape from field scenario

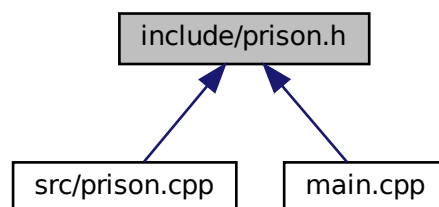
```
#include "scenario.h"
```

```
#include <vector>
```

Include dependency graph for prison.h:



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



### Classes

- class [prison](#)



### 7.14.1 Detailed Description

agents cant escape from field scenario

Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

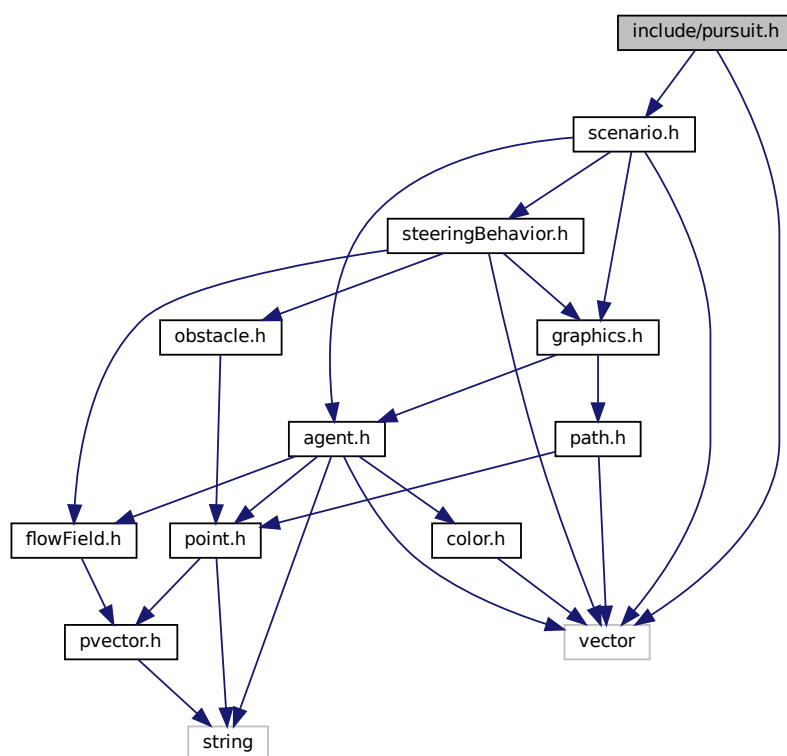
Date

15.05.2021

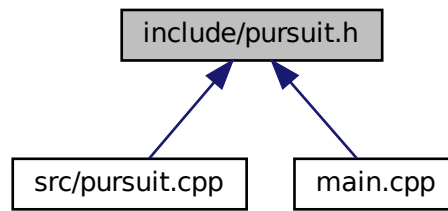
## 7.15 include/pursuit.h File Reference

one agent pursue other one scenario

```
#include "scenario.h"
#include <vector>
Include dependency graph for pursuit.h:
```



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



## Classes

- class `pursuit`

### 7.15.1 Detailed Description

one agent pursue other one scenario

Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

Date

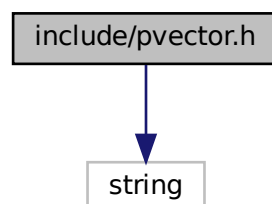
15.05.2021

## 7.16 include/pvector.h File Reference

pvector class used for 2D vector operations

```
#include <string>
```

Include dependency graph for pvector.h:



- class **pvector**

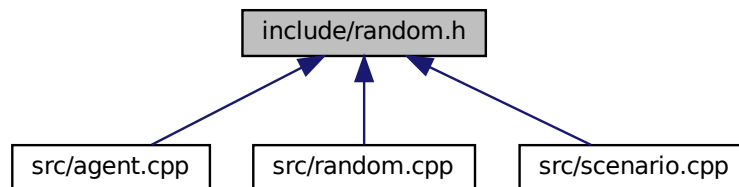
- #define PI 3.14159265

Generated by Doxygen

## 7.17 include/random.h File Reference

utility class for random operations

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



### Classes

- class [random](#)

### 7.17.1 Detailed Description

utility class for random operations

#### Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

#### Date

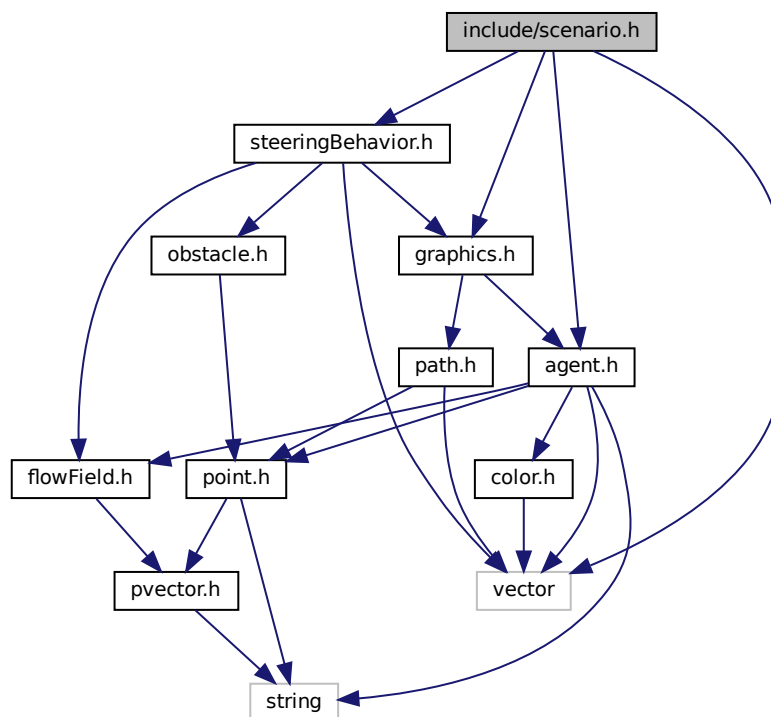
15.05.2021

## 7.18 include/scenario.h File Reference

base class for all scenarios

```
#include "agent.h"  
#include "graphics.h"  
#include "steeringBehavior.h"
```

Include dependency graph for scenario.h:

[illegible]

- class **scenario**

- enum `types` { `RANDOM` =0, `STATIC`, `TROOP` }

base class for all scenarios

**Author**

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

**Date**

15.05.2021

## 7.18.2 Enumeration Type Documentation

### 7.18.2.1 types

enum `types`

**Enumerator**

RANDOM	
STATIC	
TROOP	

Definition at line 17 of file scenario.h.

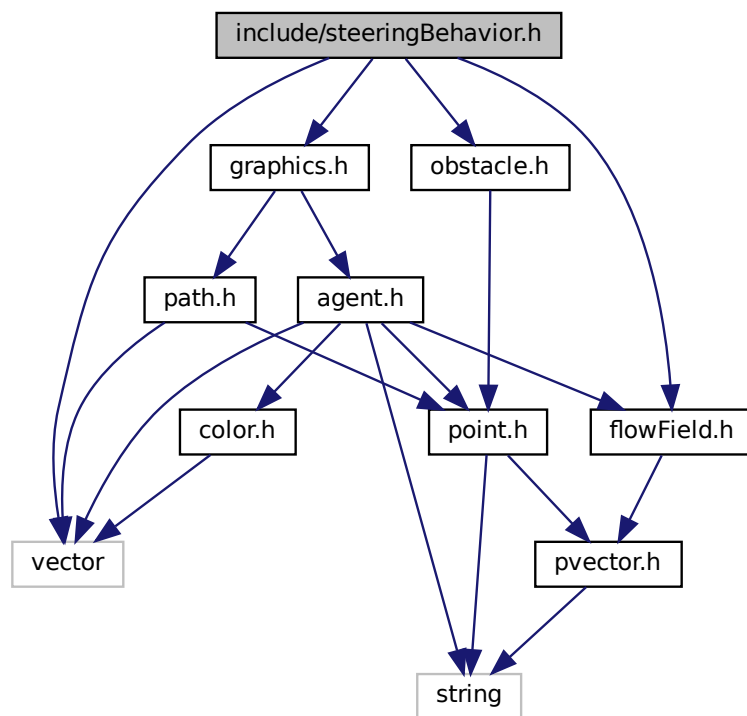
```
17 { RANDOM=0, STATIC, TROOP };
```

## 7.19 include/steeringBehavior.h File Reference

functions for autonomous steering behaviors

```
#include "flowField.h"
#include <vector>
#include "graphics.h"
#include "obstacle.h"
```

Include dependency graph for steeringBehavior.h:



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



## Classes

- class **steeringBehavior**

## Macros

- #define CIRCLE\_DISTANCE 0.1
- #define CIRCLE\_RADIUS 0.4
- #define FOLLOW\_MOUSE 1
- #define STAY\_IN\_FIELD 2
- #define IN\_FLOW\_FIELD 3
- #define AVOID\_OBSTACLE 4
- #define STAY\_IN\_PATH 5
- #define FLOCK 6
- #define WANDER 7
- #define FLEE 8
- #define PURSUIT 9
- #define EVADE 10

### 7.19.1 Detailed Description

functions for autonomous steering behaviors

#### Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

#### Date

15.05.2021

### 7.19.2 Macro Definition Documentation

#### 7.19.2.1 AVOID\_OBSTACLE

```
#define AVOID_OBSTACLE 4
```

Definition at line 21 of file steeringBehavior.h.

#### 7.19.2.2 CIRCLE\_DISTANCE

```
#define CIRCLE_DISTANCE 0.1
```

Definition at line 15 of file steeringBehavior.h.

#### 7.19.2.3 CIRCLE\_RADIUS

```
#define CIRCLE_RADIUS 0.4
```

Definition at line 16 of file steeringBehavior.h.

#### 7.19.2.4 EVADE

```
#define EVADE 10
```

Definition at line 27 of file steeringBehavior.h.



#### 7.19.2.5 FLEE

```
#define FLEE 8
```

Definition at line 25 of file steeringBehavior.h.

#### 7.19.2.6 FLOCK

```
#define FLOCK 6
```

Definition at line 23 of file steeringBehavior.h.

#### 7.19.2.7 FOLLOW\_MOUSE

```
#define FOLLOW_MOUSE 1
```

Definition at line 18 of file steeringBehavior.h.

#### 7.19.2.8 IN\_FLOW\_FIELD

```
#define IN_FLOW_FIELD 3
```

Definition at line 20 of file steeringBehavior.h.

#### 7.19.2.9 PURSUIT

```
#define PURSUIT 9
```

Definition at line 26 of file steeringBehavior.h.

#### 7.19.2.10 STAY\_IN\_FIELD

```
#define STAY_IN_FIELD 2
```

Definition at line 19 of file steeringBehavior.h.

### 7.19.2.11 STAY\_IN\_PATH

```
#define STAY_IN_PATH 5
```

Definition at line 22 of file steeringBehavior.h.

### 7.19.2.12 WANDER

```
#define WANDER 7
```

Definition at line 24 of file steeringBehavior.h.

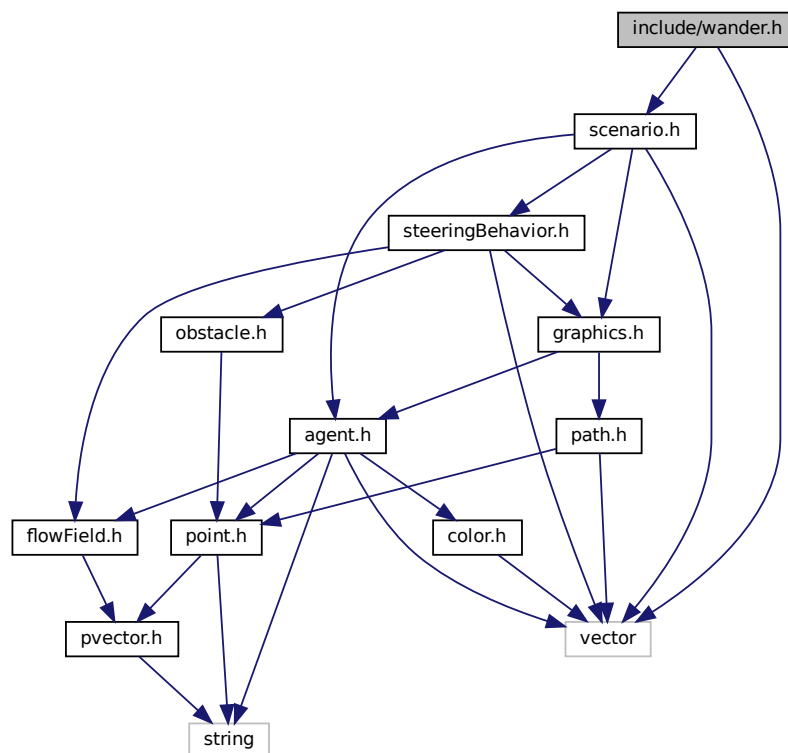
## 7.20 include/wander.h File Reference

random wandering agents scenario

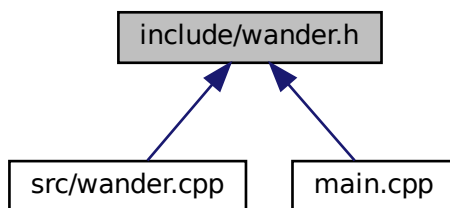
```
#include "scenario.h"
```

```
#include <vector>
```

Include dependency graph for wander.h:



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



## Classes

- class [wander](#)

### 7.20.1 Detailed Description

random wandering agents scenario

#### Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

#### Date

15.05.2021

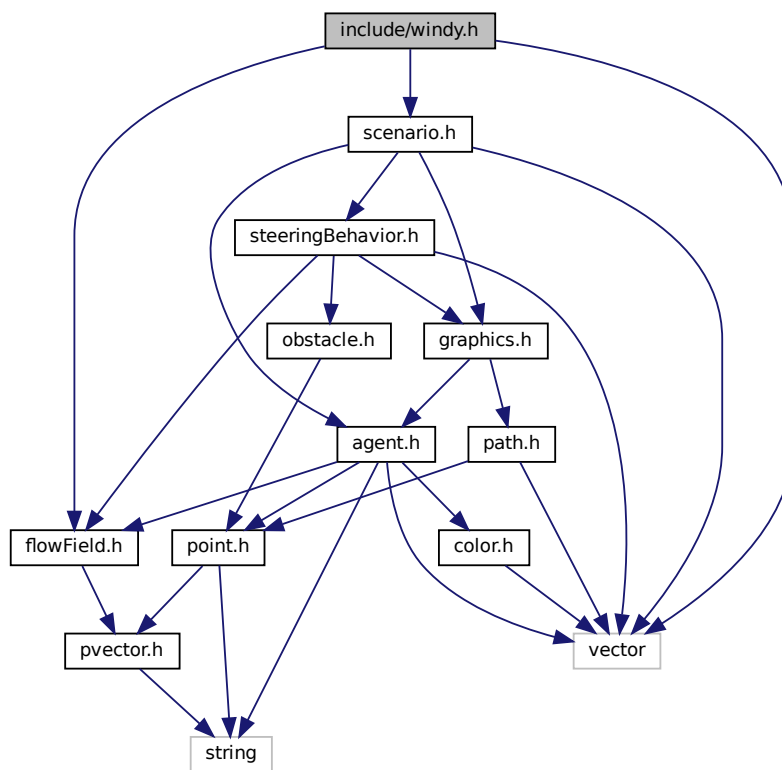
## 7.21 include/windy.h File Reference

windy air scenario

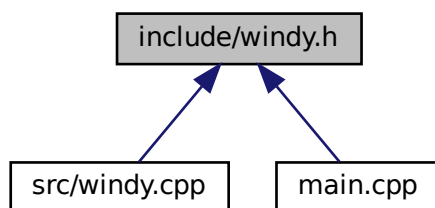
```
#include "scenario.h"
#include "flowField.h"
```

```
#include <vector>
```

Include dependency graph for windy.h:



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



## Classes

- class [windy](#)

### 7.21.1 Detailed Description

windy air scenario

Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

Date

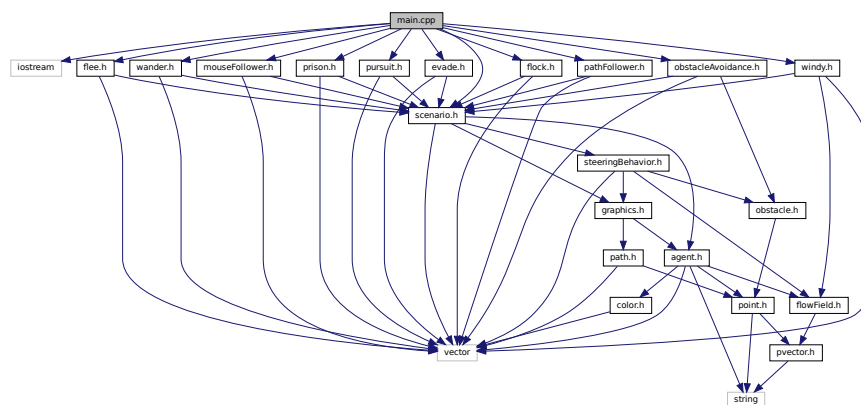
15.05.2021

## 7.22 main.cpp File Reference

client code

```
#include <iostream>
#include "mouseFollower.h"
#include "prison.h"
#include "windy.h"
#include "wander.h"
#include "pursuit.h"
#include "flee.h"
#include "scenario.h"
#include "evade.h"
#include "flock.h"
#include "pathFollower.h"
#include "obstacleAvoidance.h"
```

Include dependency graph for main.cpp:



### Functions

- void [menu](#) ()  
*displays menu*
- int [main](#) (int argc, char \*\*argv)  
*main routine*

## Variables

- int `mode`  
*specifies user selected scenario*

### 7.22.1 Detailed Description

client code

#### Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

#### Date

15.05.2021

### 7.22.2 Function Documentation

#### 7.22.2.1 main()

```
int main (  
    int argc,  
    char ** argv )
```

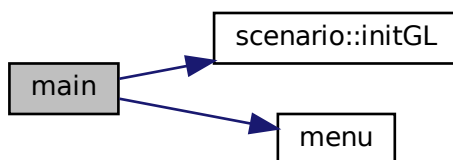
main routine

Definition at line 48 of file main.cpp.

```
48     {  
49     menu();  
50  
51     scenario* sc;  
52  
53     if(mode == FOLLOW_MOUSE) {  
54         *sc = mouseFollower();  
55     }  
56     else if(mode == STAY_IN_FIELD) {  
57         *sc = prison();  
58     }  
59     else if(mode == IN_FLOW_FIELD) {  
60         *sc = windy();  
61     }  
62     else if(mode == WANDER) {  
63         *sc = wander();  
64     }  
65     else if(mode == PURSUIT) {  
66         *sc = pursuit();  
67     }  
68     else if(mode == FLEE) {  
69         *sc = flee();  
70     }  
71     else if(mode == EVADE) {  
72         *sc = evade();  
73     }  
74     else if(mode == FLOCK) {  
75         *sc = flock();  
76     }  
77     else if(mode == STAY_IN_PATH) {  
78         *sc = pathFollower();  
79     }  
80     else if(mode == AVOID_OBSTACLE) {
```

```
81     *sc = obstacleAvoidance();
82 }
83
84 sc->initGL(&argc, argv);
85
86 return 0;
87 }
```

Here is the call graph for this function:



### 7.22.2.2 menu()

```
void menu ( )
```

displays menu

Definition at line 31 of file main.cpp.

```
31 {
32     cout << "Follow Mouse      : 1" << endl;
33     cout << "Stay in Field       : 2" << endl;
34     cout << "In Flow Field      : 3" << endl;
35     cout << "OBSTACLE AVOIDANCE : 4" << endl;
36     cout << "Stay in Path        : 5" << endl;
37     cout << "FLOCK              : 6" << endl;
38     cout << "WANDER             : 7" << endl;
39     cout << "FLEE               : 8" << endl;
40     cout << "PURSUIT            : 9" << endl;
41     cout << "EVADE              : 10" << endl;
42     cin >> mode;
43 }
```

Here is the caller graph for this function:



## 7.22.3 Variable Documentation

### 7.22.3.1 mode

```
int mode
```

specifies user selected scenario

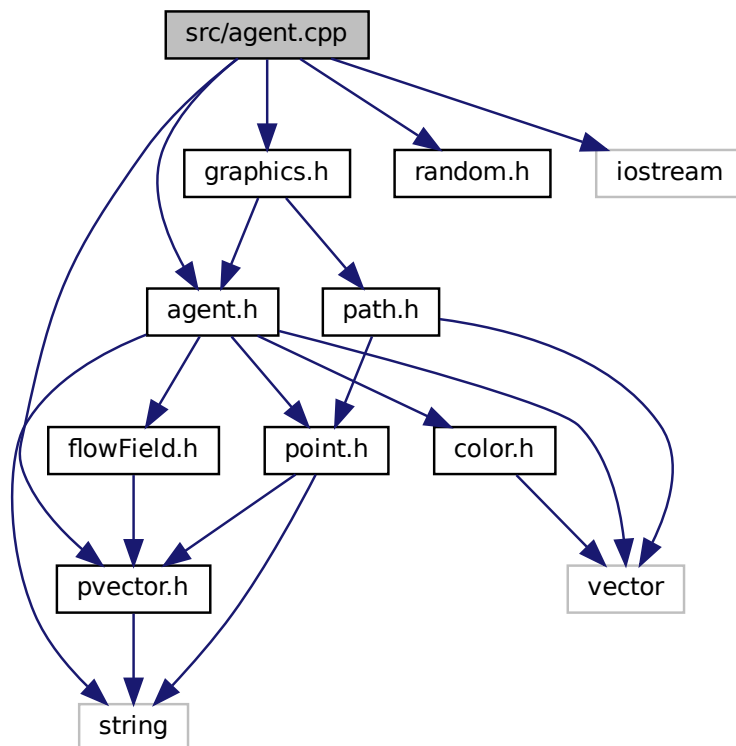
Definition at line 26 of file main.cpp.

## 7.23 README.md File Reference

## 7.24 src/agent.cpp File Reference

implementation of the agent class

```
#include "agent.h"  
#include "pvector.h"  
#include "graphics.h"  
#include "random.h"  
#include <iostream>  
Include dependency graph for agent.cpp:
```





### 7.24.1 Detailed Description

implementation of the agent class

Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

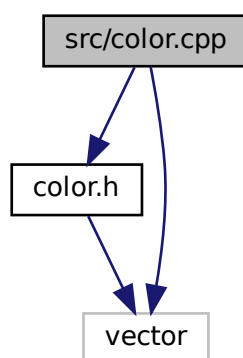
Date

14.05.2021

## 7.25 src/color.cpp File Reference

color class implementation

```
#include "color.h"  
#include <vector>  
Include dependency graph for color.cpp:
```



### 7.25.1 Detailed Description

color class implementation

Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

Date

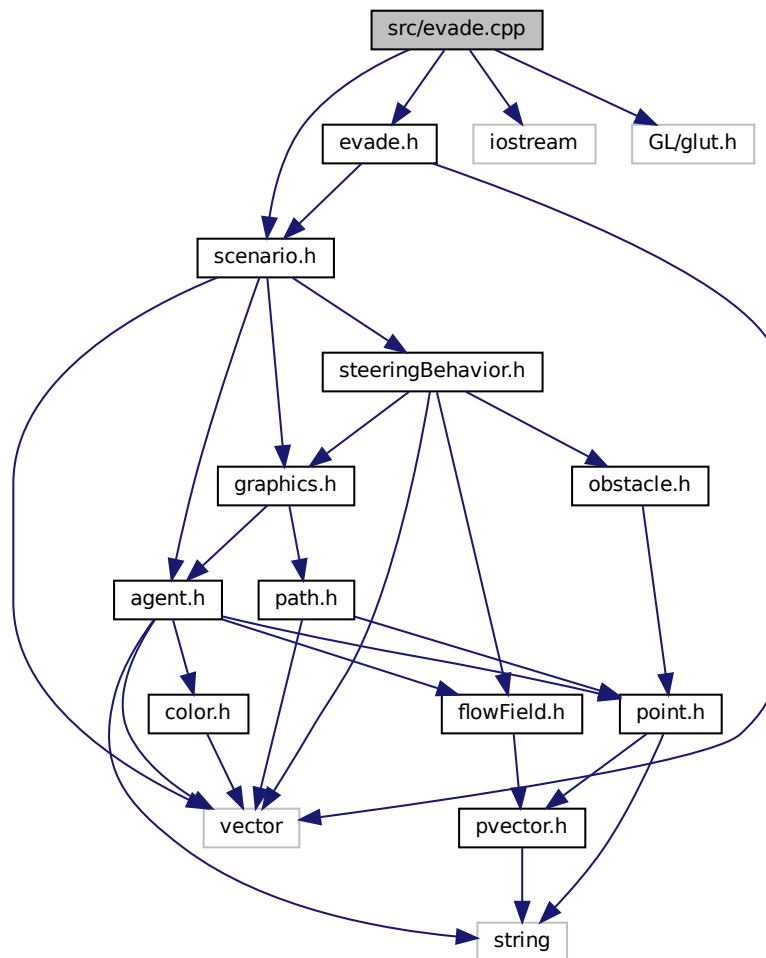
13.05.2021

## 7.26 src/evade.cpp File Reference

evade class implementation

```
#include "scenario.h"
#include "evade.h"
#include <iostream>
#include <GL/glut.h>
```

Include dependency graph for evade.cpp:



### 7.26.1 Detailed Description

evade class implementation

Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

Date

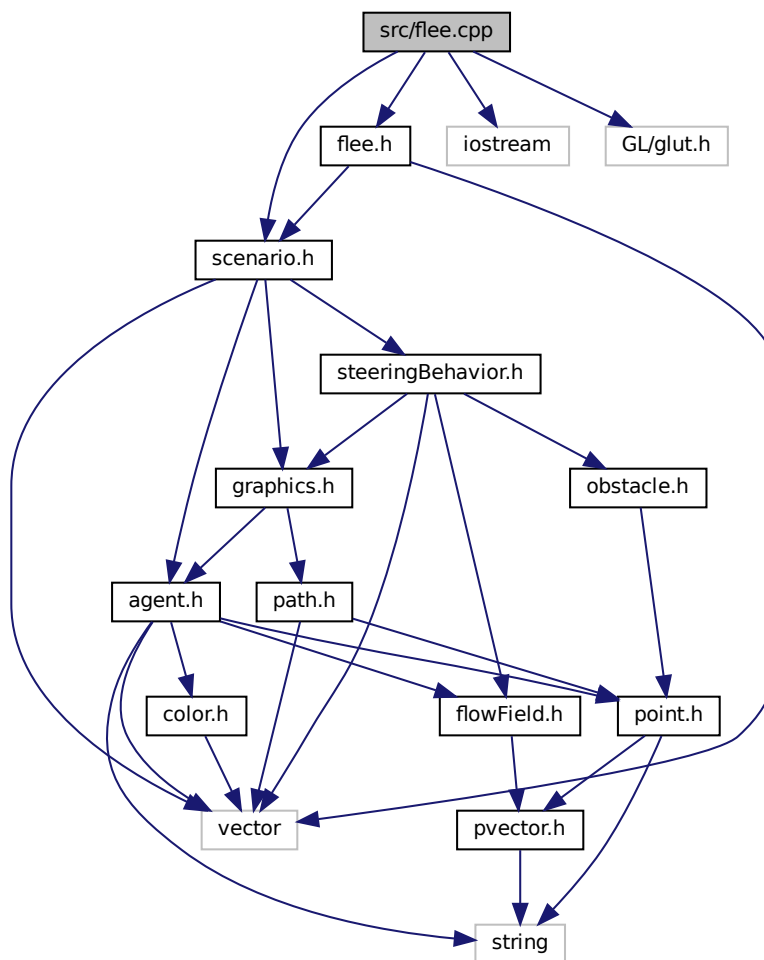
15.05.2021

## 7.27 src/flee.cpp File Reference

flee class implementation

```
#include "scenario.h"
#include "flee.h"
#include <iostream>
#include <GL/glut.h>
```

Include dependency graph for flee.cpp:



### 7.27.1 Detailed Description

flee class implementation

Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

Date

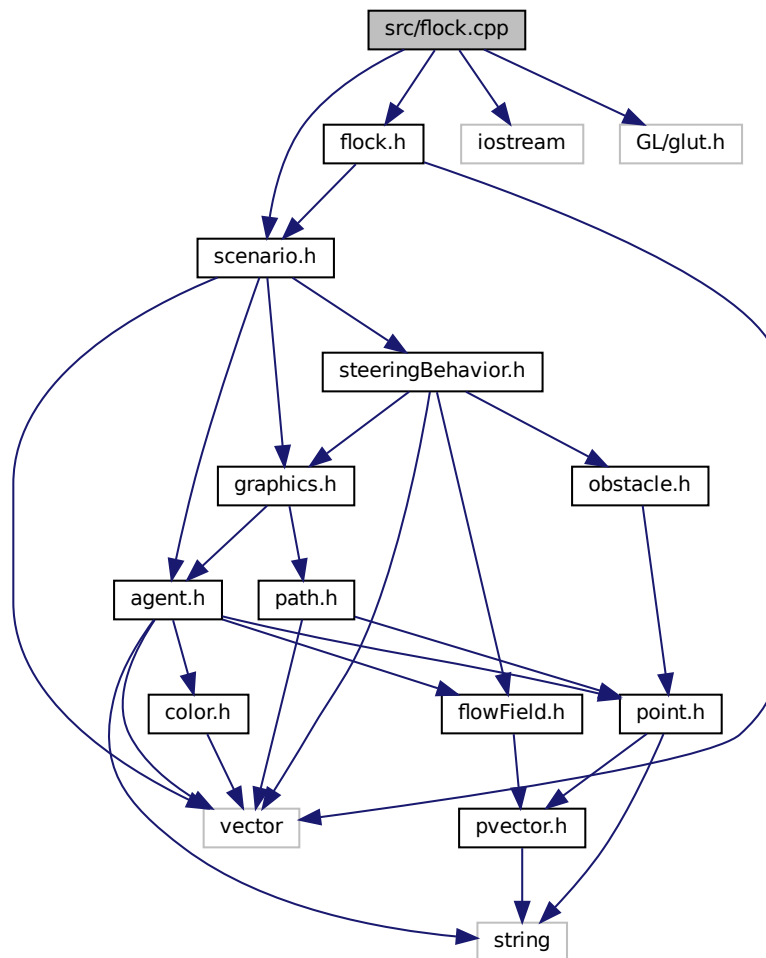
15.05.2021

## 7.28 src/flock.cpp File Reference

flock class implementation

```
#include "scenario.h"
#include "flock.h"
#include <iostream>
#include <GL/glut.h>
```

Include dependency graph for flock.cpp:



### 7.28.1 Detailed Description

flock class implementation

Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

Date

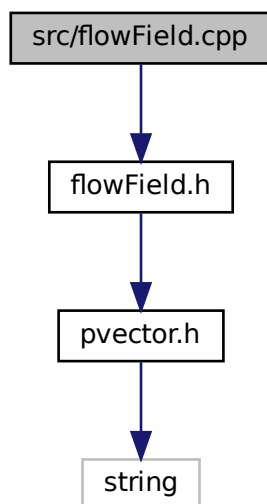
15.05.2021

## 7.29 src/flowField.cpp File Reference

[flowField](#) class implementation

```
#include "flowField.h"
```

Include dependency graph for flowField.cpp:



### 7.29.1 Detailed Description

[flowField](#) class implementation

Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

Date

13.05.2021

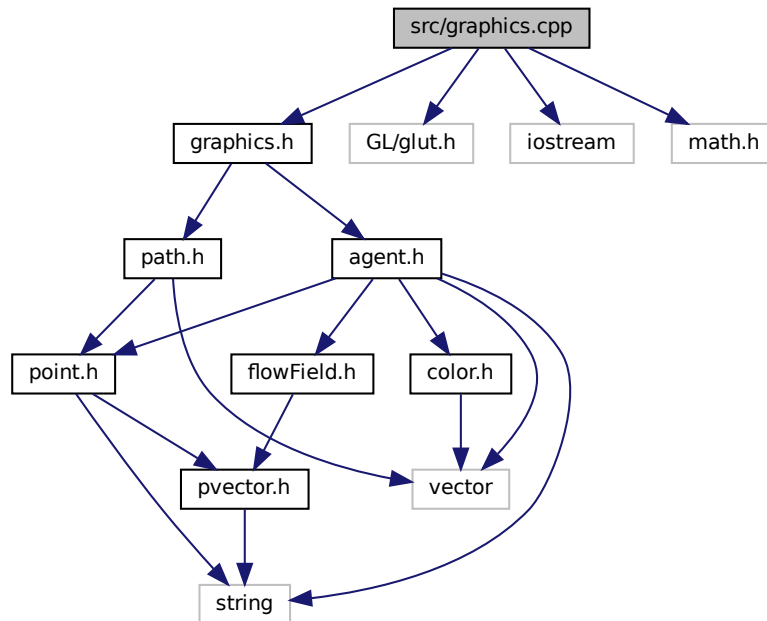
## 7.30 src/graphics.cpp File Reference

`graphics` class implementation

```
#include "graphics.h"
#include <GL/glut.h>
#include <iostream>
```

```
#include "math.h"
```

Include dependency graph for graphics.cpp:



### 7.30.1 Detailed Description

graphics class implementation

Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

Date

15.05.2021

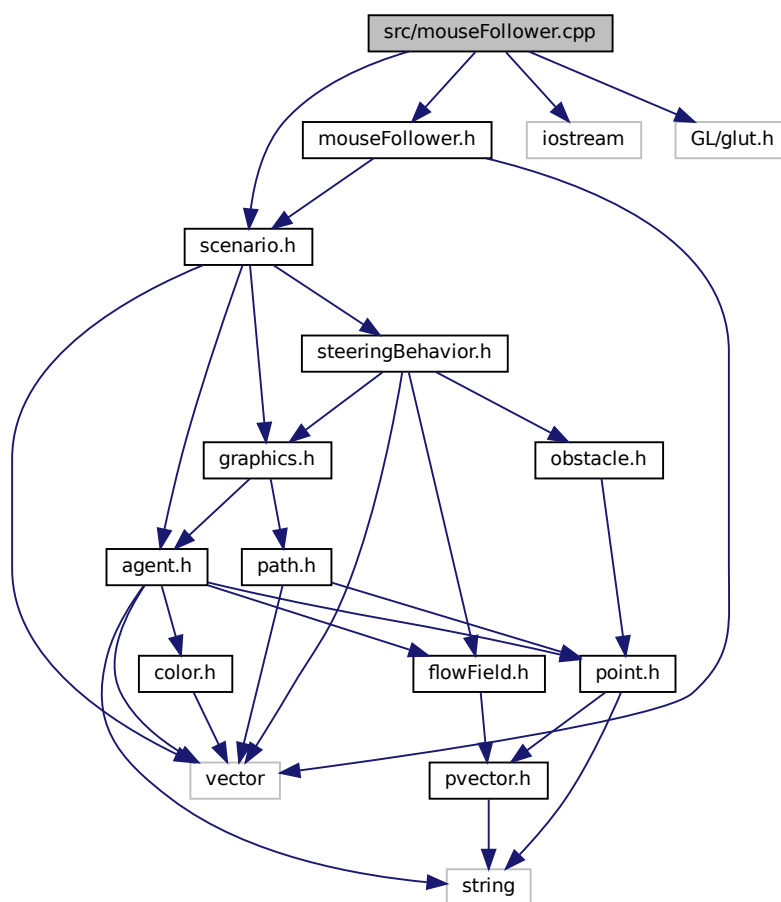
## 7.31 src/mouseFollower.cpp File Reference

`mouseFollower` class implementation

```
#include "scenario.h"
#include "mouseFollower.h"
#include <iostream>
```

```
#include <GL/glut.h>
```

Include dependency graph for mouseFollower.cpp:



### 7.31.1 Detailed Description

[mouseFollower](#) class implementation

Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

Date

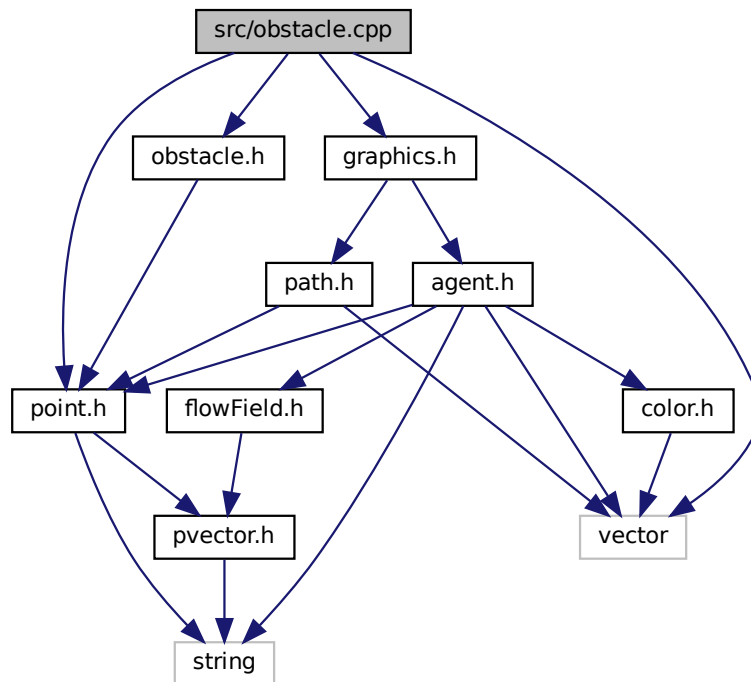
15.05.2021

## 7.32 src/obstacle.cpp File Reference

obstacle class implementation

```
#include "obstacle.h"
#include "graphics.h"
#include "point.h"
#include <vector>
```

Include dependency graph for obstacle.cpp:



### 7.32.1 Detailed Description

obstacle class implementation

Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

Date

12.05.2021

## 7.33 src/obstacleAvoidance.cpp File Reference

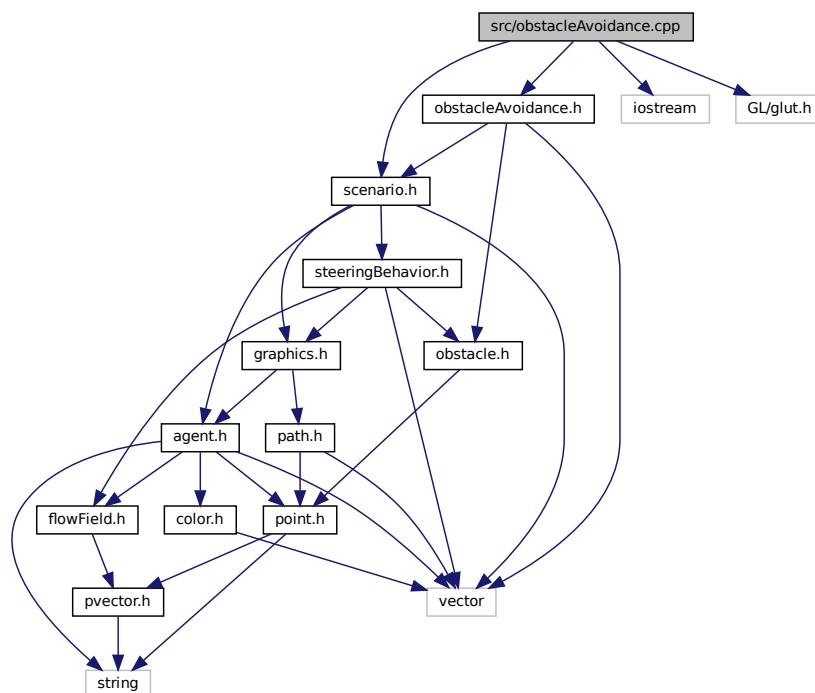
`obstacleAvoidance` class implementation

```
#include "scenario.h"
#include "obstacleAvoidance.h"
```



```
#include <iostream>
#include <GL/glut.h>
```

Include dependency graph for obstacleAvoidance.cpp:



### 7.33.1 Detailed Description

[obstacleAvoidance](#) class implementation

Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

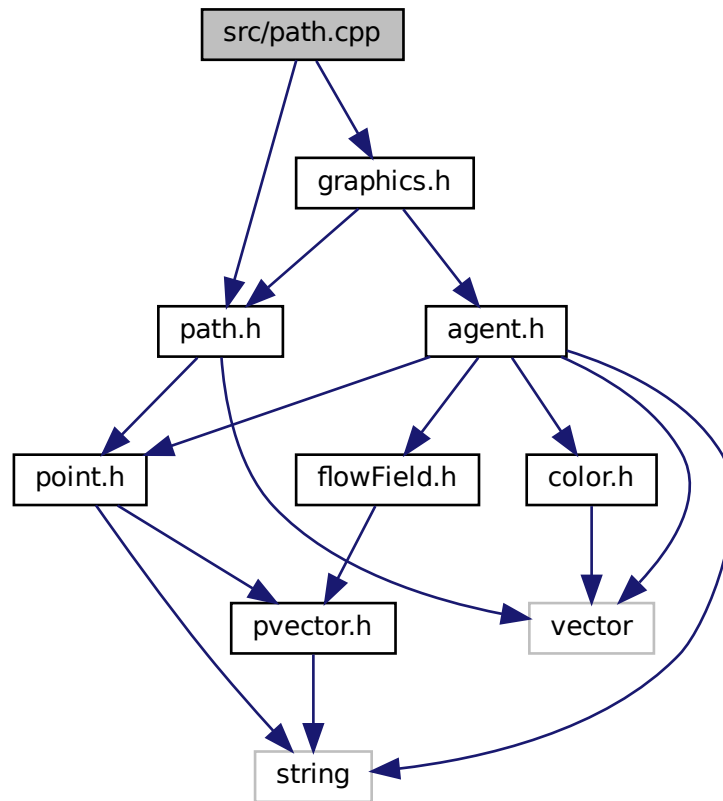
Date

15.05.2021

## 7.34 src/path.cpp File Reference

[path](#) class implementation

```
#include "path.h"
#include "graphics.h"
Include dependency graph for path.cpp:
```



### 7.34.1 Detailed Description

path class implementation

Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

Date

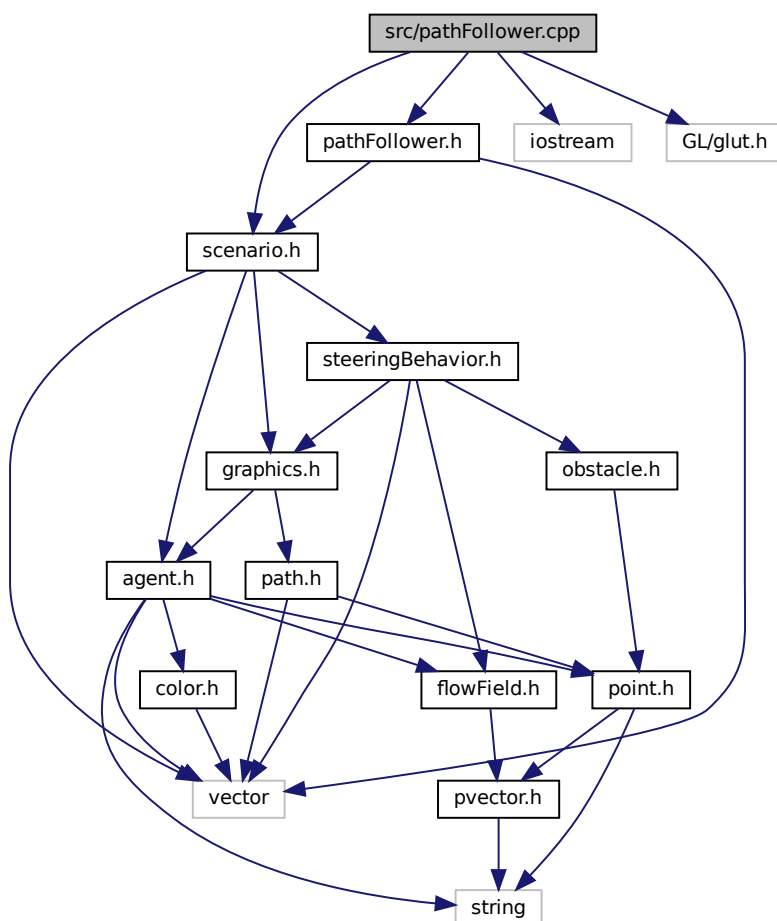
12.05.2021

## 7.35 src/pathFollower.cpp File Reference

[pathFollower](#) class implementation

```
#include "scenario.h"
#include "pathFollower.h"
#include <iostream>
#include <GL/glut.h>
```

Include dependency graph for pathFollower.cpp:



### 7.35.1 Detailed Description

[pathFollower](#) class implementation

Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

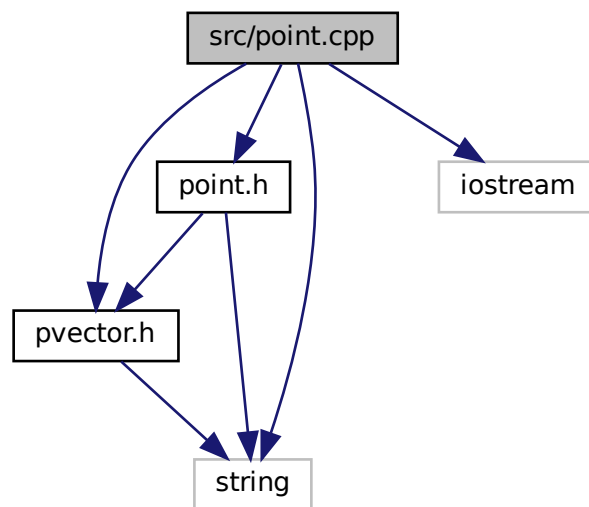
Date

15.05.2021

## 7.36 src/point.cpp File Reference

point class implementation file

```
#include "point.h"  
#include "pvector.h"  
#include <string>  
#include <iostream>  
Include dependency graph for point.cpp:
```



### 7.36.1 Detailed Description

point class implementation file

Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

Date

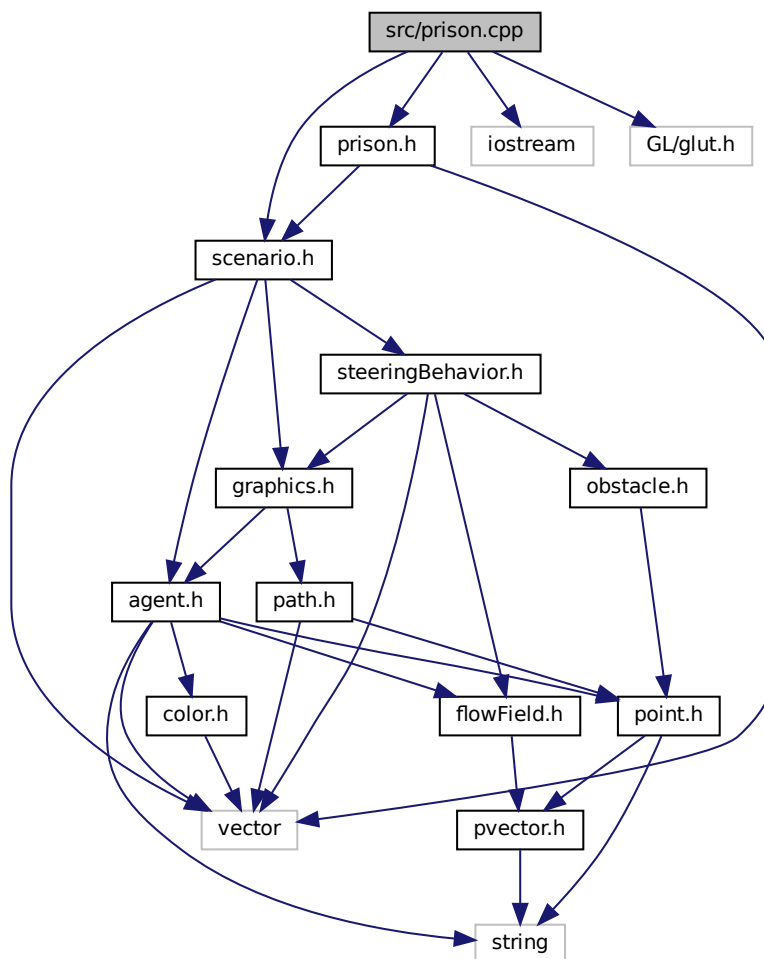
15.05.2021

## 7.37 src/prison.cpp File Reference

prison class implementation

```
#include "scenario.h"  
#include "prison.h"
```

```
#include <iostream>
#include <GL/glut.h>
Include dependency graph for prison.cpp:
```



## Macros

- `#define WALL 30`
- `#define DISTANCE 2`

### 7.37.1 Detailed Description

prison class implementation

Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

Date

15.05.2021

## 7.37.2 Macro Definition Documentation

### 7.37.2.1 DISTANCE

```
#define DISTANCE 2
```

Definition at line 14 of file prison.cpp.

### 7.37.2.2 WALL

```
#define WALL 30
```

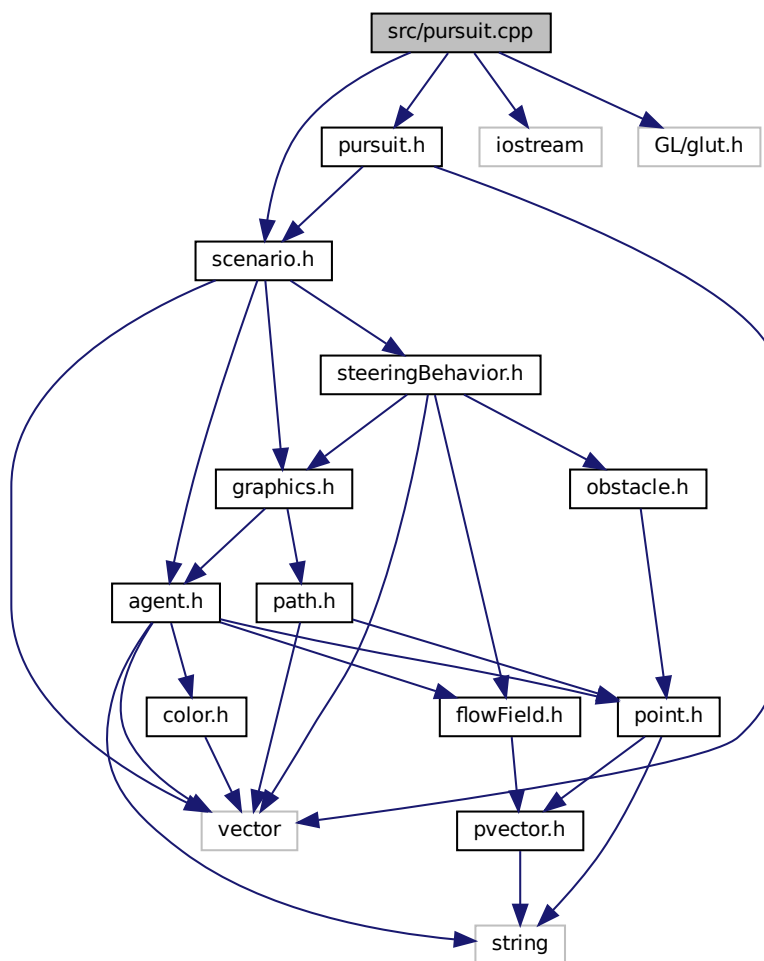
Definition at line 13 of file prison.cpp.

## 7.38 src/pursuit.cpp File Reference

prison class implementation

```
#include "scenario.h"  
#include "pursuit.h"  
#include <iostream>  
#include <GL/glut.h>
```

Include dependency graph for pursuit.cpp:



### 7.38.1 Detailed Description

prison class implementation

Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

Date

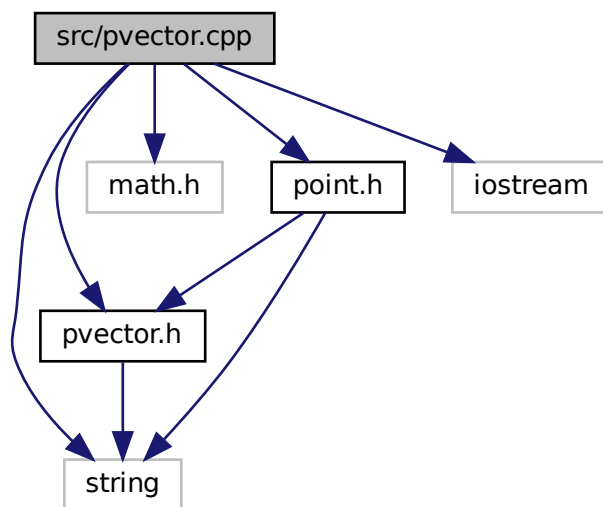
15.05.2021

## 7.39 src/pvector.cpp File Reference

pvector class implementation

```
#include "pvector.h"  
#include "math.h"  
#include "point.h"  
#include <iostream>  
#include <string>
```

Include dependency graph for pvector.cpp:



### 7.39.1 Detailed Description

pvector class implementation

Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

Date

15.05.2021

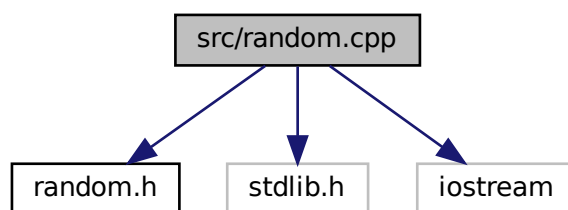


## 7.40 src/random.cpp File Reference

utility class for random operations

```
#include "random.h"
#include <stdlib.h>
#include <iostream>
```

Include dependency graph for random.cpp:



### 7.40.1 Detailed Description

utility class for random operations

Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

Date

15.05.2021

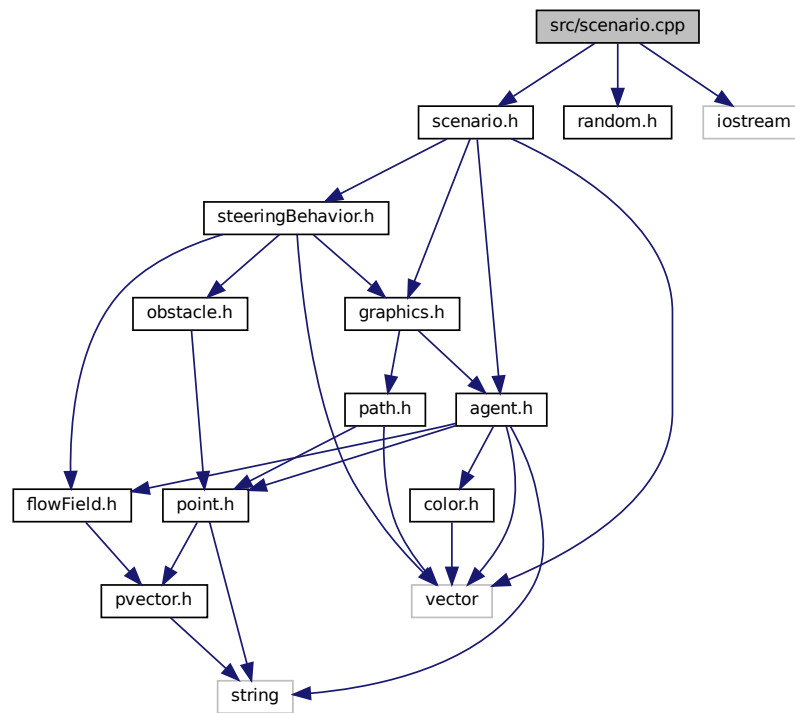
## 7.41 src/scenario.cpp File Reference

scenario base class implementation

```
#include "scenario.h"
#include "random.h"
```

```
#include <iostream>
```

Include dependency graph for scenario.cpp:



## Macros

- `#define MAX_NUMBER_OF_AGENTS 50`

### 7.41.1 Detailed Description

scenario base class implementation

#### Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

#### Date

15.05.2021

### 7.41.2 Macro Definition Documentation

### 7.41.2.1 MAX\_NUMBER\_OF\_AGENTS

```
#define MAX_NUMBER_OF_AGENTS 50
```

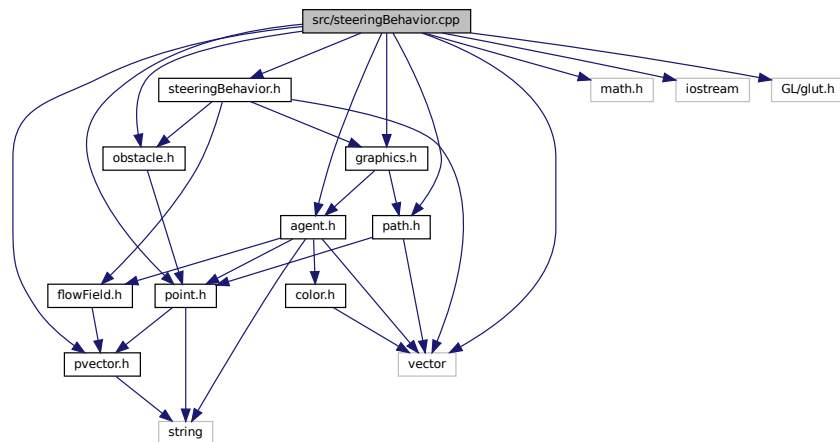
Definition at line 12 of file scenario.cpp.

## 7.42 src/steeringBehavior.cpp File Reference

implementation of autonomous steering behaviors

```
#include "steeringBehavior.h"
#include "pvector.h"
#include "agent.h"
#include "path.h"
#include "point.h"
#include <vector>
#include "graphics.h"
#include "math.h"
#include "obstacle.h"
#include <iostream>
#include <GL/glut.h>
```

Include dependency graph for steeringBehavior.cpp:



### 7.42.1 Detailed Description

implementation of autonomous steering behaviors

Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

Date

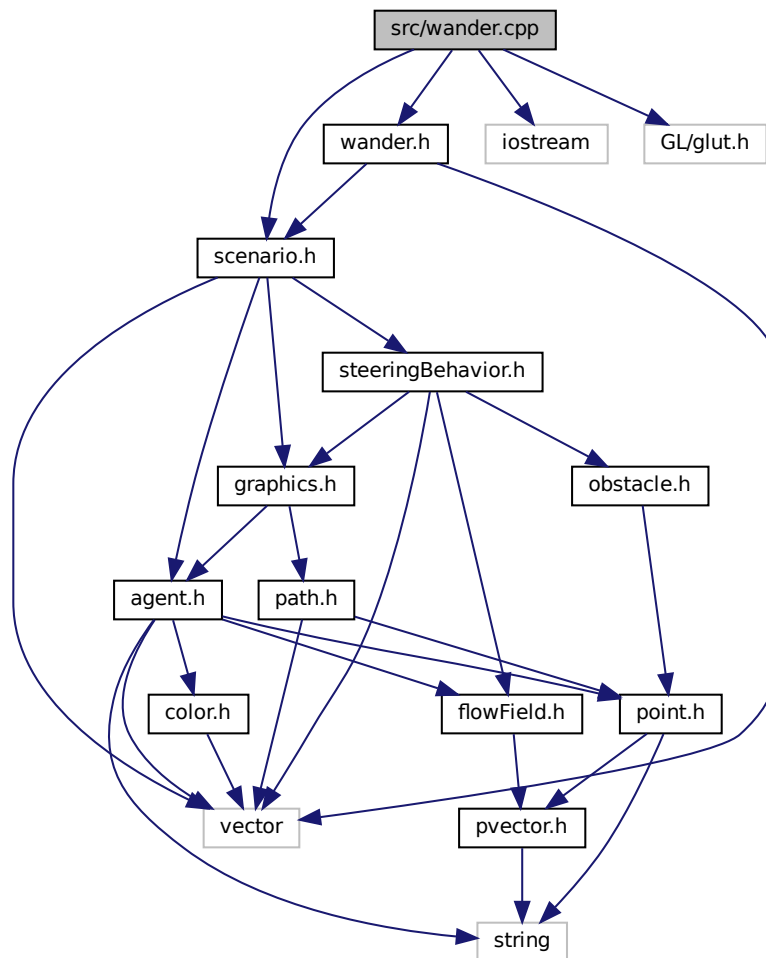
15.05.2021

## 7.43 src/wander.cpp File Reference

wander class implementation

```
#include "scenario.h"
#include "wander.h"
#include <iostream>
#include <GL/glut.h>
```

Include dependency graph for wander.cpp:



### 7.43.1 Detailed Description

wander class implementation

Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

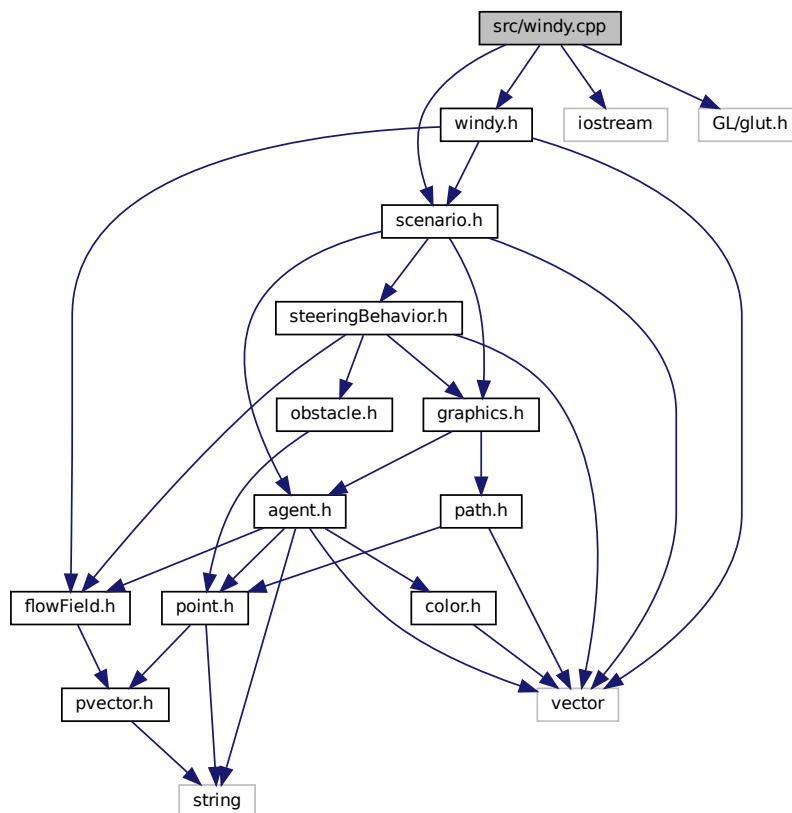
Date

15.05.2021

## 7.44 src/windy.cpp File Reference

windy class implementation

```
#include "scenario.h"
#include "windy.h"
#include <iostream>
#include <GL/glut.h>
Include dependency graph for windy.cpp:
```



### 7.44.1 Detailed Description

windy class implementation

Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

Date

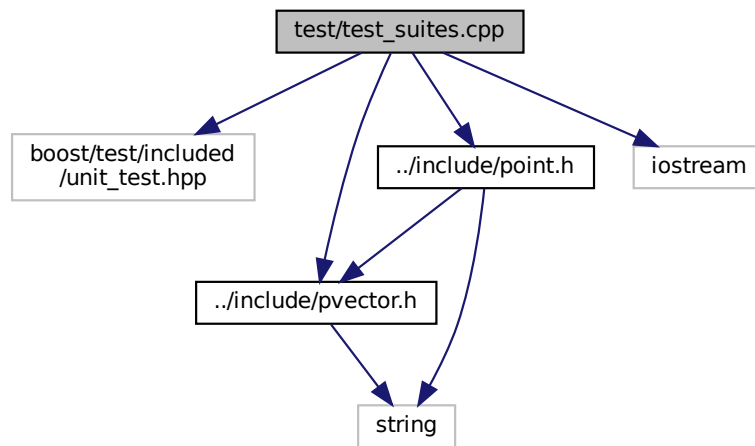
15.05.2021

## 7.45 test/test\_suites.cpp File Reference

unit test suites

```
#include <boost/test/included/unit_test.hpp>
#include "../include/pvector.h"
#include "../include/point.h"
#include <iostream>
```

Include dependency graph for test\_suites.cpp:



### Macros

- #define [BOOST\\_TEST\\_MODULE](#) test\_suites

### Functions

- [BOOST\\_AUTO\\_TEST\\_CASE](#) (s1t1)  
*pvector magnitude test case*
- [BOOST\\_AUTO\\_TEST\\_CASE](#) (s1t2)  
*pvector mul test case*
- [BOOST\\_AUTO\\_TEST\\_CASE](#) (s1t3)  
*pvector div test case*
- [BOOST\\_AUTO\\_TEST\\_CASE](#) (s1t4)  
*pvector dotproduct test case*
- [BOOST\\_AUTO\\_TEST\\_CASE](#) (s1t5)  
*pvector angle between vectors test case*
- [BOOST\\_AUTO\\_TEST\\_CASE](#) (s1t6)  
*pvector get vector angle test case*
- [BOOST\\_AUTO\\_TEST\\_CASE](#) (s1t7)  
*pvector normalize test case*
- [BOOST\\_AUTO\\_TEST\\_CASE](#) (s1t8)

- pvector limit test case*
  - [BOOST\\_AUTO\\_TEST\\_CASE](#) (s1t9)
- pvector overloaded operators test case*
  - [BOOST\\_AUTO\\_TEST\\_CASE](#) (s2t1)
- point multiplication test case*
  - [BOOST\\_AUTO\\_TEST\\_CASE](#) (s2t2)
- point division test case*
  - [BOOST\\_AUTO\\_TEST\\_CASE](#) (s2t3)
- point overloaded operators test case*

### 7.45.1 Detailed Description

unit test suites

#### Author

Mehmet Rıza Öz - [mehmetrizaoz@gmail.com](mailto:mehmetrizaoz@gmail.com)

#### Date

15.05.2021

### 7.45.2 Macro Definition Documentation

#### 7.45.2.1 BOOST\_TEST\_MODULE

```
#define BOOST_TEST_MODULE test_suites
```

Definition at line 8 of file test\_suites.cpp.

### 7.45.3 Function Documentation

### 7.45.3.1 BOOST\_AUTO\_TEST\_CASE() [1/12]

```
BOOST_AUTO_TEST_CASE (
    slt1 )
```

pvector magnitude test case

Definition at line 22 of file test\_suites.cpp.

```
23 {
24     pvector p1 = pvector(0, 4);
25     pvector p2 = pvector(3, 0);
26     pvector p3 = p1 + p2;
27     BOOST_CHECK(p3.magnitude() == 5);
28 }
```

Here is the call graph for this function:



### 7.45.3.2 BOOST\_AUTO\_TEST\_CASE() [2/12]

```
BOOST_AUTO_TEST_CASE (
    slt2 )
```

pvector mul test case

Definition at line 33 of file test\_suites.cpp.

```
34 {
35     pvector p1 = pvector(1, 1);
36     p1.mul(3);
37     pvector p2 = pvector(3, 3);
38     BOOST_CHECK(p1 == p2);
39 }
```

Here is the call graph for this function:





### 7.45.3.3 BOOST\_AUTO\_TEST\_CASE() [3/12]

```
BOOST_AUTO_TEST_CASE (
    slt3 )
```

pvector div test case

Definition at line 44 of file test\_suites.cpp.

```
45 {
46     pvector p1 = pvector(5, 5);
47     p1.div(5);
48     pvector p2 = pvector(1, 1);
49     BOOST_CHECK(p1 == p2);
50 }
```

Here is the call graph for this function:



### 7.45.3.4 BOOST\_AUTO\_TEST\_CASE() [4/12]

```
BOOST_AUTO_TEST_CASE (
    slt4 )
```

pvector dotproduct test case

Definition at line 55 of file test\_suites.cpp.

```
56 {
57     pvector p1 = pvector(1, 4);
58     pvector p2 = pvector(3, 2);
59     float dotProduct = p1.dotProduct(p2);
60     BOOST_CHECK(dotProduct == 11);
61 }
```

Here is the call graph for this function:



### 7.45.3.5 BOOST\_AUTO\_TEST\_CASE() [5/12]

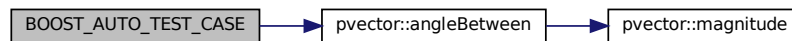
```
BOOST_AUTO_TEST_CASE (
    slt5 )
```

pvector angle between vectors test case

Definition at line 66 of file test\_suites.cpp.

```
67 {
68     pvector p1 = pvector(10, 10);
69     pvector p2 = pvector(0, 10);
70     float angle = p1.angleBetween(p2);
71     BOOST_CHECK(angle == 45);
72 }
```

Here is the call graph for this function:



### 7.45.3.6 BOOST\_AUTO\_TEST\_CASE() [6/12]

```
BOOST_AUTO_TEST_CASE (
    slt6 )
```

pvector get vector angle test case

Definition at line 77 of file test\_suites.cpp.

```
78 {
79     pvector p1 = pvector(3, 4);
80     float angle = p1.getAngle();
81     BOOST_CHECK(angle < 53.2 && angle > 52.8);
82 }
```

Here is the call graph for this function:



**7.45.3.7 BOOST\_AUTO\_TEST\_CASE()** [7/12]

```
BOOST_AUTO_TEST_CASE (
    slt7 )
```

pvector normalize test case

Definition at line 87 of file test\_suites.cpp.

```
88 {
89     pvector p1 = pvector(2, 2);
90     p1.normalize();
91     float range = 0.01;
92     BOOST_CHECK_CLOSE_FRACTION(0.707, p1.x, range);
93     BOOST_CHECK_CLOSE_FRACTION(0.707, p1.y, range);
94 }
```

Here is the call graph for this function:

**7.45.3.8 BOOST\_AUTO\_TEST\_CASE()** [8/12]

```
BOOST_AUTO_TEST_CASE (
    slt8 )
```

pvector limit test case

Definition at line 99 of file test\_suites.cpp.

```
100 {
101     pvector p1 = pvector(2, 2);
102     p1.limit(3);
103     float range = 0.01;
104     BOOST_CHECK_CLOSE_FRACTION(2.12, p1.x, range);
105     BOOST_CHECK_CLOSE_FRACTION(2.12, p1.y, range);
106 }
```

Here is the call graph for this function:



**7.45.3.9 BOOST\_AUTO\_TEST\_CASE()** [9/12]

```
BOOST_AUTO_TEST_CASE (
    s1t9 )
```

pvector overloaded operators test case

Definition at line 111 of file test\_suites.cpp.

```
112 {
113     pvector p1 = pvector(1, 1);
114     p1 += pvector(1, 1);
115     BOOST_CHECK(p1 == pvector(2, 2));
116     p1 = pvector(1, 1) + pvector(3, 3);
117     BOOST_CHECK(p1 == pvector(4, 4));
118     p1 = pvector(4, 1) - pvector(3, 3);
119     BOOST_CHECK(p1 == pvector(1, -2));
120     p1 = pvector(4, 1) - point(3, 3);
121     BOOST_CHECK(p1 == pvector(1, -2));
122     p1 = pvector(4, 1) + point(3, 3);
123     BOOST_CHECK(p1 == pvector(7, 4));
124 }
```

Here is the call graph for this function:

**7.45.3.10 BOOST\_AUTO\_TEST\_CASE()** [10/12]

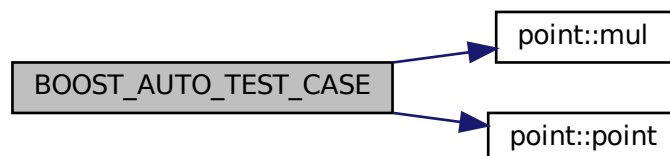
```
BOOST_AUTO_TEST_CASE (
    s2t1 )
```

point multiplication test case

Definition at line 133 of file test\_suites.cpp.

```
134 {
135     point p1 = point(1, 1);
136     p1.mul(3);
137     point p2 = point(3, 3);
138     BOOST_CHECK(p1 == p2);
139 }
```

Here is the call graph for this function:



### 7.45.3.11 BOOST\_AUTO\_TEST\_CASE() [11/12]

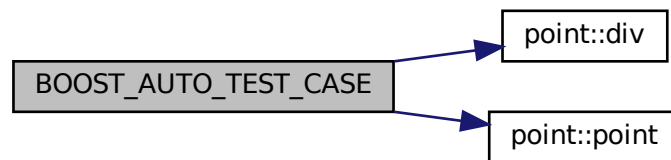
```
BOOST_AUTO_TEST_CASE (
    s2t2 )
```

point division test case

Definition at line 144 of file test\_suites.cpp.

```
145 {
146     point p1 = point(4, 4);
147     p1.div(4);
148     point p2 = point(1, 1);
149     BOOST_CHECK(p1 == p2);
150 }
```

Here is the call graph for this function:



### 7.45.3.12 BOOST\_AUTO\_TEST\_CASE() [12/12]

```
BOOST_AUTO_TEST_CASE (
    s2t3 )
```

point overloaded operators test case

Definition at line 155 of file test\_suites.cpp.

```
156 {
157     point p1 = point(1,1) + point(3,3);
158     BOOST_CHECK(p1 == point(4,4));
159     p1 = point(1,1) + pvector(3,3);
160     BOOST_CHECK(p1 == point(4,4));
161     pvector p2 = point(1,1) - point(3,3);
162     BOOST_CHECK(p2 == pvector(-2,-2));
163 }
```

Here is the call graph for this function:





# Index

- ~agent
  - agent, [13](#)
- acceleration
  - agent, [15](#)
- add
  - pvector, [81](#)
- addPoint
  - path, [58](#)
- agent, [11](#)
  - ~agent, [13](#)
  - acceleration, [15](#)
  - agent, [13](#)
  - arrive, [15](#)
  - desiredVelocity, [15](#)
  - fillColor, [16](#)
  - force, [16](#)
  - id, [16](#)
  - mass, [16](#)
  - maxForce, [16](#)
  - maxSpeed, [17](#)
  - name, [17](#)
  - position, [17](#)
  - r, [17](#)
  - setFeatures, [14](#)
  - steering, [17](#)
  - targetPoint, [18](#)
  - updatePosition, [14](#)
  - velocity, [18](#)
- agents
  - scenario, [97](#)
- align
  - steeringBehavior, [100](#)
- angleBetween
  - pvector, [81](#)
- arrive
  - agent, [15](#)
- avoid
  - steeringBehavior, [101](#)
- AVOID\_OBSTACLE
  - steeringBehavior.h, [148](#)
- B
  - color, [22](#)
- behavior
  - scenario, [97](#)
- BLACK
  - color.h, [123](#)
- BLUE
  - color.h, [123](#)
- BOOST\_AUTO\_TEST\_CASE
  - test\_suites.cpp, [179–185](#)
- BOOST\_TEST\_MODULE
  - test\_suites.cpp, [179](#)
- callback
  - scenario, [98](#)
- CIRCLE\_DISTANCE
  - steeringBehavior.h, [148](#)
- CIRCLE\_RADIUS
  - steeringBehavior.h, [148](#)
- cohesion
  - steeringBehavior, [102](#)
- color, [18](#)
  - B, [22](#)
  - color, [20](#)
  - colors, [22](#)
  - createColors, [21](#)
  - G, [22](#)
  - getColor, [21](#)
  - R, [22](#)
- color.h
  - BLACK, [123](#)
  - BLUE, [123](#)
  - CYAN, [123](#)
  - GREEN, [123](#)
  - MAGENDA, [123](#)
  - num, [123](#)
  - RED, [123](#)
  - WHITE, [123](#)
  - YELLOW, [123](#)
- colors
  - color, [22](#)
- createAgent
  - scenario, [95](#)
- createColors
  - color, [21](#)
- createObstacle
  - obstacleAvoidance, [53](#)
- createPath
  - pathFollower, [61](#)
- createRandomArray
  - random, [92](#)
- CYAN
  - color.h, [123](#)
- desiredVelocity
  - agent, [15](#)
- DISTANCE
  - prison.cpp, [170](#)

- div
  - point, 66
  - pvector, 82
- dotProduct
  - pvector, 83
- drawAgent
  - graphics, 36
- drawCircle
  - graphics, 37
- drawLine
  - graphics, 37
- drawPath
  - graphics, 38
- drawPoint
  - graphics, 39
- drawText
  - graphics, 39
- ESC
  - graphics.h, 131
- EVADE
  - steeringBehavior.h, 148
- evade, 23
  - evade, 25
  - loop, 25
  - steeringBehavior, 103
- FIELD\_HEIGHT
  - flowField.h, 129
- FIELD\_WIDTH
  - flowField.h, 129
- fillColor
  - agent, 16
- FLEE
  - steeringBehavior.h, 148
- flee, 26
  - flee, 28
  - loop, 28
  - steeringBehavior, 104
- FLOCK
  - steeringBehavior.h, 149
- flock, 29
  - flock, 31
  - loop, 31
- flow
  - windy, 119
- flowField, 32
  - flowField, 33
  - getField, 34
- flowField.h
  - FIELD\_HEIGHT, 129
  - FIELD\_WIDTH, 129
  - GRAVITY, 129
  - WIND\_WEST, 129
- FOLLOW\_MOUSE
  - steeringBehavior.h, 149
- force
  - agent, 16
- forceInScreen
  - graphics, 40
- G
  - color, 22
- getAngle
  - pvector, 83
- getColor
  - color, 21
- getField
  - flowField, 34
- getMousePosition
  - graphics, 40
- getNormalPoint
  - point, 67
- graphics, 35
  - drawAgent, 36
  - drawCircle, 37
  - drawLine, 37
  - drawPath, 38
  - drawPoint, 39
  - drawText, 39
  - forceInScreen, 40
  - getMousePosition, 40
  - handleKeypress, 41
  - handleResize, 41
  - initGraphics, 42
  - mouseButton, 43
  - mouseMove, 44
  - refreshScene, 44
  - target\_x, 45
  - target\_y, 45
  - timerEvent, 45
- graphics.h
  - ESC, 131
  - HEIGHT, 131
  - PI, 131
  - WIDTH, 131
- GRAVITY
  - flowField.h, 129
- GREEN
  - color.h, 123
- handleKeypress
  - graphics, 41
- handleResize
  - graphics, 41
- HEIGHT
  - graphics.h, 131
- id
  - agent, 16
- IN\_FLOW\_FIELD
  - steeringBehavior.h, 149
- include/agent.h, 121
- include/color.h, 122
- include/evade.h, 124
- include/flee.h, 125
- include/flock.h, 126
- include/flowField.h, 128



- include/graphics.h, 130
- include/mouseFollower.h, 132
- include/obstacle.h, 133
- include/obstacleAvoidance.h, 135
- include/path.h, 136
- include/pathFollower.h, 137
- include/point.h, 138
- include/prison.h, 140
- include/pursuit.h, 141
- include/pvector.h, 142
- include/random.h, 144
- include/scenario.h, 144
- include/steeringBehavior.h, 146
- include/wander.h, 150
- include/windy.h, 151
- inFlowField
  - steeringBehavior, 105
- initGL
  - scenario, 96
- initGraphics
  - graphics, 42
- limit
  - pvector, 84
- loop
  - evade, 25
  - flee, 28
  - flock, 31
  - mouseFollower, 48
  - obstacleAvoidance, 54
  - pathFollower, 62
  - prison, 74
  - pursuit, 78
  - wander, 115
  - windy, 118
- MAGENDA
  - color.h, 123
- magnitude
  - pvector, 85
- main
  - main.cpp, 154
- main.cpp, 153
  - main, 154
  - menu, 155
  - mode, 155
- mass
  - agent, 16
- MAX\_NUMBER\_OF\_AGENTS
  - scenario.cpp, 174
- maxForce
  - agent, 16
- maxSpeed
  - agent, 17
- menu
  - main.cpp, 155
- mode
  - main.cpp, 155
- mouseButton
  - graphics, 43
- mouseFollower, 46
  - loop, 48
  - mouseFollower, 48
- mouseMove
  - graphics, 44
- mul
  - point, 68
  - pvector, 86
- myColor
  - scenario, 98
- myPath
  - pathFollower, 63
- name
  - agent, 17
  - scenario, 98
- normalize
  - pvector, 87
- num
  - color.h, 123
- obstacle, 49
  - obstacle, 50
  - p, 50
  - r, 51
- obstacleAvoidance, 51
  - createObstacle, 53
  - loop, 54
  - obstacleAvoidance, 53
  - obstacles, 55
- obstacles
  - obstacleAvoidance, 55
- operator+
  - point, 69
  - pvector, 88
- operator+=
  - pvector, 89
- operator-
  - point, 70
  - pvector, 89, 90
- operator==
  - point, 70
  - pvector, 90
- p
  - obstacle, 50
- path, 55
  - addPoint, 58
  - path, 57
  - points, 58
  - width, 58
- pathFollower, 59
  - createPath, 61
  - loop, 62
  - myPath, 63
  - pathFollower, 61
- PI
  - graphics.h, 131

- pvector.h, 143
- point, 63
  - div, 66
  - getNormalPoint, 67
  - mul, 68
  - operator+, 69
  - operator-, 70
  - operator==, 70
  - point, 64, 65
  - print, 71
  - x, 71
  - y, 71
- points
  - path, 58
- position
  - agent, 17
- print
  - point, 71
  - pvector, 91
- prison, 72
  - loop, 74
  - prison, 74
- prison.cpp
  - DISTANCE, 170
  - WALL, 170
- PURSUIT
  - steeringBehavior.h, 149
- pursuit, 75
  - loop, 78
  - pursuit, 78
  - steeringBehavior, 106
- pvector, 79
  - add, 81
  - angleBetween, 81
  - div, 82
  - dotProduct, 83
  - getAngle, 83
  - limit, 84
  - magnitude, 85
  - mul, 86
  - normalize, 87
  - operator+, 88
  - operator+==, 89
  - operator-, 89, 90
  - operator==, 90
  - print, 91
  - pvector, 80
  - x, 91
  - y, 91
- pvector.h
  - PI, 143
- R
  - color, 22
- r
  - agent, 17
  - obstacle, 51
- RANDOM
  - scenario.h, 146
- random, 92
  - createRandomArray, 92
- README.md, 156
- RED
  - color.h, 123
- refresh
  - scenario, 96
- refreshScene
  - graphics, 44
- scenario, 93
  - agents, 97
  - behavior, 97
  - callback, 98
  - createAgent, 95
  - initGL, 96
  - myColor, 98
  - name, 98
  - refresh, 96
  - scenario, 95
  - view, 98
- scenario.cpp
  - MAX\_NUMBER\_OF\_AGENTS, 174
- scenario.h
  - RANDOM, 146
  - STATIC, 146
  - TROOP, 146
  - types, 146
- seek
  - steeringBehavior, 107
- separation
  - steeringBehavior, 107
- setAngle
  - steeringBehavior, 109
- setFeatures
  - agent, 14
- src/agent.cpp, 156
- src/color.cpp, 157
- src/evade.cpp, 158
- src/flee.cpp, 159
- src/flock.cpp, 160
- src/flowField.cpp, 161
- src/graphics.cpp, 161
- src/mouseFollower.cpp, 162
- src/obstacle.cpp, 163
- src/obstacleAvoidance.cpp, 164
- src/path.cpp, 165
- src/pathFollower.cpp, 166
- src/point.cpp, 168
- src/prison.cpp, 168
- src/pursuit.cpp, 170
- src/pvector.cpp, 172
- src/random.cpp, 173
- src/scenario.cpp, 173
- src/steeringBehavior.cpp, 175
- src/wander.cpp, 176
- src/windy.cpp, 177
- STATIC
  - scenario.h, 146

- STAY\_IN\_FIELD
  - steeringBehavior.h, 149
- STAY\_IN\_PATH
  - steeringBehavior.h, 149
- stayInArea
  - steeringBehavior, 110
- stayInPath
  - steeringBehavior, 110
- steering
  - agent, 17
- steeringBehavior, 99
  - align, 100
  - avoid, 101
  - cohesion, 102
  - evade, 103
  - flee, 104
  - inFlowField, 105
  - pursuit, 106
  - seek, 107
  - separation, 107
  - setAngle, 109
  - stayInArea, 110
  - stayInPath, 110
  - wander, 111
- steeringBehavior.h
  - AVOID\_OBSTACLE, 148
  - CIRCLE\_DISTANCE, 148
  - CIRCLE\_RADIUS, 148
  - EVADE, 148
  - FLEE, 148
  - FLOCK, 149
  - FOLLOW\_MOUSE, 149
  - IN\_FLOW\_FIELD, 149
  - PURSUIT, 149
  - STAY\_IN\_FIELD, 149
  - STAY\_IN\_PATH, 149
  - WANDER, 150
- target\_x
  - graphics, 45
- target\_y
  - graphics, 45
- targetPoint
  - agent, 18
- test/test\_suites.cpp, 178
- test\_suites.cpp
  - BOOST\_AUTO\_TEST\_CASE, 179–185
  - BOOST\_TEST\_MODULE, 179
- timerEvent
  - graphics, 45
- TROOP
  - scenario.h, 146
- types
  - scenario.h, 146
- updatePosition
  - agent, 14
- velocity
  - agent, 18
- view
  - scenario, 98
- WALL
  - prison.cpp, 170
- WANDER
  - steeringBehavior.h, 150
- wander, 112
  - loop, 115
  - steeringBehavior, 111
  - wander, 115
- WHITE
  - color.h, 123
- WIDTH
  - graphics.h, 131
- width
  - path, 58
- WIND\_WEST
  - flowField.h, 129
- windy, 116
  - flow, 119
  - loop, 118
  - windy, 118
- x
  - point, 71
  - pvector, 91
- y
  - point, 71
  - pvector, 91
- YELLOW
  - color.h, 123