

## Autonomous Steering Agents

Generated by Doxygen 1.8.17



<b>1 Intent</b>	<b>1</b>
1.1 Dependencies	1
1.2 Resources	1
1.3 Links	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>13</b>
6.1 agent Class Reference	13
6.1.1 Detailed Description	14
6.1.2 Constructor & Destructor Documentation	14
6.1.2.1 agent() [1/2]	14
6.1.2.2 agent() [2/2]	15
6.1.2.3 ~agent()	16
6.1.3 Member Function Documentation	16
6.1.3.1 draw()	16
6.1.3.2 getMass()	17
6.1.3.3 getName()	17
6.1.3.4 setFeatures()	17
6.1.3.5 setMass()	18
6.1.3.6 setName()	18
6.1.3.7 updatePosition()	18
6.1.4 Member Data Documentation	19
6.1.4.1 acceleration	19
6.1.4.2 arrive	19
6.1.4.3 desiredVelocity	19
6.1.4.4 force	20
6.1.4.5 id	20
6.1.4.6 maxForce	20
6.1.4.7 maxSpeed	20
6.1.4.8 position	20
6.1.4.9 r	21
6.1.4.10 steering	21
6.1.4.11 targetPoint	21
6.1.4.12 velocity	21

6.2 color Class Reference . . . . .	21
6.2.1 Detailed Description . . . . .	22
6.2.2 Constructor & Destructor Documentation . . . . .	22
6.2.2.1 color() [1/2] . . . . .	22
6.2.2.2 color() [2/2] . . . . .	22
6.2.3 Member Function Documentation . . . . .	23
6.2.3.1 getColor() . . . . .	23
6.2.4 Member Data Documentation . . . . .	23
6.2.4.1 B . . . . .	24
6.2.4.2 G . . . . .	24
6.2.4.3 R . . . . .	24
6.3 entity Class Reference . . . . .	24
6.3.1 Detailed Description . . . . .	25
6.3.2 Constructor & Destructor Documentation . . . . .	25
6.3.2.1 entity() . . . . .	25
6.3.3 Member Function Documentation . . . . .	25
6.3.3.1 draw() . . . . .	25
6.3.3.2 getId() . . . . .	25
6.3.3.3 getName() . . . . .	26
6.3.3.4 setId() . . . . .	26
6.3.3.5 setName() . . . . .	26
6.3.4 Member Data Documentation . . . . .	27
6.3.4.1 entityColor . . . . .	27
6.4 evade Class Reference . . . . .	27
6.4.1 Detailed Description . . . . .	27
6.4.2 Constructor & Destructor Documentation . . . . .	27
6.4.2.1 evade() . . . . .	28
6.4.3 Member Function Documentation . . . . .	28
6.4.3.1 loop() . . . . .	28
6.5 flee Class Reference . . . . .	28
6.5.1 Detailed Description . . . . .	29
6.5.2 Constructor & Destructor Documentation . . . . .	29
6.5.2.1 flee() . . . . .	29
6.5.3 Member Function Documentation . . . . .	29
6.5.3.1 loop() . . . . .	29
6.6 flock Class Reference . . . . .	30
6.6.1 Detailed Description . . . . .	30
6.6.2 Constructor & Destructor Documentation . . . . .	30
6.6.2.1 flock() . . . . .	30
6.6.3 Member Function Documentation . . . . .	30
6.6.3.1 loop() . . . . .	31
6.7 flowField Class Reference . . . . .	31

6.7.1 Detailed Description . . . . .	31
6.7.2 Constructor & Destructor Documentation . . . . .	32
6.7.2.1 flowField() [1/2] . . . . .	32
6.7.2.2 flowField() [2/2] . . . . .	32
6.7.3 Member Function Documentation . . . . .	32
6.7.3.1 getField() . . . . .	32
6.8 graphics Class Reference . . . . .	33
6.8.1 Detailed Description . . . . .	34
6.8.2 Member Function Documentation . . . . .	34
6.8.2.1 drawAgent() . . . . .	34
6.8.2.2 drawCircle() . . . . .	35
6.8.2.3 drawLine() . . . . .	35
6.8.2.4 drawPath() . . . . .	36
6.8.2.5 drawPoint() . . . . .	36
6.8.2.6 drawText() . . . . .	36
6.8.2.7 forceInScreen() . . . . .	37
6.8.2.8 getMousePosition() . . . . .	37
6.8.2.9 handleKeypress() . . . . .	38
6.8.2.10 handleResize() . . . . .	38
6.8.2.11 initGraphics() . . . . .	38
6.8.2.12 mouseButton() . . . . .	40
6.8.2.13 mouseMove() . . . . .	40
6.8.2.14 refreshScene() . . . . .	41
6.8.2.15 timerEvent() . . . . .	41
6.8.3 Member Data Documentation . . . . .	41
6.8.3.1 target_x . . . . .	42
6.8.3.2 target_y . . . . .	42
6.9 leaderFollower Class Reference . . . . .	42
6.9.1 Detailed Description . . . . .	43
6.9.2 Constructor & Destructor Documentation . . . . .	43
6.9.2.1 leaderFollower() . . . . .	43
6.9.3 Member Function Documentation . . . . .	43
6.9.3.1 loop() . . . . .	43
6.9.4 Member Data Documentation . . . . .	44
6.9.4.1 leaderPosition . . . . .	44
6.9.4.2 leaderVelocity . . . . .	44
6.10 mouseFollower Class Reference . . . . .	45
6.10.1 Detailed Description . . . . .	45
6.10.2 Constructor & Destructor Documentation . . . . .	45
6.10.2.1 mouseFollower() . . . . .	45
6.10.3 Member Function Documentation . . . . .	45
6.10.3.1 loop() . . . . .	46

6.11 obstacle Class Reference	46
6.11.1 Detailed Description	46
6.11.2 Constructor & Destructor Documentation	47
6.11.2.1 obstacle() [1/2]	47
6.11.2.2 obstacle() [2/2]	47
6.11.3 Member Function Documentation	47
6.11.3.1 draw()	48
6.11.4 Member Data Documentation	48
6.11.4.1 p	48
6.11.4.2 r	48
6.12 obstacleAvoidance Class Reference	48
6.12.1 Detailed Description	49
6.12.2 Constructor & Destructor Documentation	49
6.12.2.1 obstacleAvoidance()	49
6.12.3 Member Function Documentation	49
6.12.3.1 createObstacle()	49
6.12.3.2 loop()	50
6.12.4 Member Data Documentation	50
6.12.4.1 obstacles	50
6.13 path Class Reference	51
6.13.1 Detailed Description	51
6.13.2 Constructor & Destructor Documentation	51
6.13.2.1 path() [1/2]	51
6.13.2.2 path() [2/2]	51
6.13.3 Member Function Documentation	52
6.13.3.1 addPoint()	52
6.13.3.2 draw()	52
6.13.4 Member Data Documentation	53
6.13.4.1 points	53
6.13.4.2 width	53
6.14 pathFollower Class Reference	53
6.14.1 Detailed Description	54
6.14.2 Constructor & Destructor Documentation	54
6.14.2.1 pathFollower()	54
6.14.3 Member Function Documentation	54
6.14.3.1 createPath()	54
6.14.3.2 loop()	55
6.14.4 Member Data Documentation	55
6.14.4.1 myPath	55
6.15 point Class Reference	55
6.15.1 Detailed Description	56
6.15.2 Constructor & Destructor Documentation	56

6.15.2.1 point() [1/2]	56
6.15.2.2 point() [2/2]	57
6.15.3 Member Function Documentation	57
6.15.3.1 div()	57
6.15.3.2 getNormalPoint()	57
6.15.3.3 mul()	58
6.15.3.4 operator+() [1/2]	58
6.15.3.5 operator+() [2/2]	59
6.15.3.6 operator-()	59
6.15.3.7 operator==()	60
6.15.3.8 print()	60
6.15.4 Member Data Documentation	60
6.15.4.1 x	61
6.15.4.2 y	61
6.16 prison Class Reference	61
6.16.1 Detailed Description	61
6.16.2 Constructor & Destructor Documentation	61
6.16.2.1 prison()	62
6.16.3 Member Function Documentation	62
6.16.3.1 loop()	62
6.17 pursuit Class Reference	62
6.17.1 Detailed Description	63
6.17.2 Constructor & Destructor Documentation	63
6.17.2.1 pursuit()	63
6.17.3 Member Function Documentation	63
6.17.3.1 loop()	64
6.18 pvector Class Reference	64
6.18.1 Detailed Description	65
6.18.2 Constructor & Destructor Documentation	65
6.18.2.1 pvector() [1/2]	65
6.18.2.2 pvector() [2/2]	66
6.18.3 Member Function Documentation	66
6.18.3.1 add()	66
6.18.3.2 angleBetween()	66
6.18.3.3 div()	67
6.18.3.4 dotProduct()	67
6.18.3.5 getAngle()	68
6.18.3.6 limit()	68
6.18.3.7 magnitude()	68
6.18.3.8 mul()	69
6.18.3.9 normalize()	69
6.18.3.10 operator+() [1/2]	69

6.18.3.11 operator+() [2/2]	70
6.18.3.12 operator+=()	70
6.18.3.13 operator-() [1/2]	71
6.18.3.14 operator-() [2/2]	71
6.18.3.15 operator==()	72
6.18.3.16 print()	72
6.18.4 Member Data Documentation	72
6.18.4.1 x	72
6.18.4.2 y	73
6.19 random Class Reference	73
6.19.1 Detailed Description	73
6.19.2 Member Function Documentation	73
6.19.2.1 createRandomArray()	73
6.20 scenario Class Reference	74
6.20.1 Detailed Description	75
6.20.2 Constructor & Destructor Documentation	75
6.20.2.1 scenario()	75
6.20.3 Member Function Documentation	75
6.20.3.1 createAgent()	75
6.20.3.2 initGL()	76
6.20.3.3 refresh()	76
6.20.4 Member Data Documentation	76
6.20.4.1 agents	77
6.20.4.2 behavior	77
6.20.4.3 callback	77
6.20.4.4 name	77
6.20.4.5 view	78
6.21 steeringBehavior Class Reference	78
6.21.1 Detailed Description	79
6.21.2 Member Function Documentation	79
6.21.2.1 align()	79
6.21.2.2 avoid()	79
6.21.2.3 cohesion()	80
6.21.2.4 evade()	81
6.21.2.5 flee()	82
6.21.2.6 inFlowField()	82
6.21.2.7 pursuit()	83
6.21.2.8 seek()	83
6.21.2.9 separation()	84
6.21.2.10 setAngle()	85
6.21.2.11 stayInArea()	85
6.21.2.12 stayInPath()	86



6.21.2.13 wander()	86
6.22 wander Class Reference	87
6.22.1 Detailed Description	87
6.22.2 Constructor & Destructor Documentation	87
6.22.2.1 wander()	88
6.22.3 Member Function Documentation	88
6.22.3.1 loop()	88
6.23 windy Class Reference	88
6.23.1 Detailed Description	89
6.23.2 Constructor & Destructor Documentation	89
6.23.2.1 windy()	89
6.23.3 Member Function Documentation	89
6.23.3.1 loop()	90
6.23.4 Member Data Documentation	90
6.23.4.1 flow	90
<b>7 File Documentation</b>	<b>91</b>
7.1 include/agent.h File Reference	91
7.1.1 Detailed Description	92
7.2 include/color.h File Reference	92
7.2.1 Detailed Description	93
7.2.2 Macro Definition Documentation	93
7.2.2.1 BLACK	93
7.2.2.2 BLUE	94
7.2.2.3 CYAN	94
7.2.2.4 GREEN	94
7.2.2.5 MAGENDA	94
7.2.2.6 RED	94
7.2.2.7 WHITE	94
7.2.2.8 YELLOW	95
7.3 include/entity.h File Reference	95
7.4 include/evade.h File Reference	96
7.4.1 Detailed Description	97
7.5 include/flee.h File Reference	97
7.5.1 Detailed Description	98
7.6 include/flock.h File Reference	98
7.6.1 Detailed Description	99
7.7 include/flowField.h File Reference	100
7.7.1 Detailed Description	101
7.7.2 Macro Definition Documentation	101
7.7.2.1 FIELD_HEIGHT	101
7.7.2.2 FIELD_WIDTH	101

7.7.2.3 GRAVITY . . . . .	101
7.7.2.4 WIND_WEST . . . . .	102
7.8 include/graphics.h File Reference . . . . .	102
7.8.1 Detailed Description . . . . .	103
7.8.2 Macro Definition Documentation . . . . .	103
7.8.2.1 ESC . . . . .	103
7.8.2.2 HEIGHT . . . . .	103
7.8.2.3 PI . . . . .	103
7.8.2.4 WIDTH . . . . .	104
7.9 include/leaderFollower.h File Reference . . . . .	104
7.9.1 Detailed Description . . . . .	105
7.10 include/mouseFollower.h File Reference . . . . .	105
7.10.1 Detailed Description . . . . .	106
7.11 include/obstacle.h File Reference . . . . .	106
7.11.1 Detailed Description . . . . .	107
7.12 include/obstacleAvoidance.h File Reference . . . . .	108
7.12.1 Detailed Description . . . . .	109
7.13 include/path.h File Reference . . . . .	109
7.13.1 Detailed Description . . . . .	110
7.14 include/pathFollower.h File Reference . . . . .	110
7.14.1 Detailed Description . . . . .	111
7.15 include/point.h File Reference . . . . .	112
7.15.1 Detailed Description . . . . .	113
7.16 include/prison.h File Reference . . . . .	113
7.16.1 Detailed Description . . . . .	114
7.17 include/pursuit.h File Reference . . . . .	114
7.17.1 Detailed Description . . . . .	115
7.18 include/pvector.h File Reference . . . . .	116
7.18.1 Detailed Description . . . . .	117
7.18.2 Macro Definition Documentation . . . . .	117
7.18.2.1 PI . . . . .	117
7.19 include/random.h File Reference . . . . .	117
7.19.1 Detailed Description . . . . .	118
7.20 include/scenario.h File Reference . . . . .	118
7.20.1 Detailed Description . . . . .	119
7.20.2 Enumeration Type Documentation . . . . .	119
7.20.2.1 types . . . . .	119
7.21 include/steeringBehavior.h File Reference . . . . .	119
7.21.1 Detailed Description . . . . .	121
7.21.2 Macro Definition Documentation . . . . .	121
7.21.2.1 AVOID_OBSTACLE . . . . .	121
7.21.2.2 CIRCLE_DISTANCE . . . . .	121

7.21.2.3 CIRCLE_RADIUS . . . . .	121
7.21.2.4 EVADE . . . . .	121
7.21.2.5 FLEE . . . . .	122
7.21.2.6 FLOCK . . . . .	122
7.21.2.7 FOLLOW_MOUSE . . . . .	122
7.21.2.8 IN_FLOW_FIELD . . . . .	122
7.21.2.9 LEADER_FOLLOWER . . . . .	122
7.21.2.10 PURSUIT . . . . .	122
7.21.2.11 STAY_IN_FIELD . . . . .	123
7.21.2.12 STAY_IN_PATH . . . . .	123
7.21.2.13 WANDER . . . . .	123
7.22 include/wander.h File Reference . . . . .	123
7.22.1 Detailed Description . . . . .	124
7.23 include/windy.h File Reference . . . . .	124
7.23.1 Detailed Description . . . . .	125
7.24 main.cpp File Reference . . . . .	126
7.24.1 Detailed Description . . . . .	127
7.24.2 Function Documentation . . . . .	127
7.24.2.1 main() . . . . .	127
7.24.2.2 menu() . . . . .	128
7.24.3 Variable Documentation . . . . .	128
7.24.3.1 mode . . . . .	128
7.25 README.md File Reference . . . . .	128
7.26 src/agent.cpp File Reference . . . . .	128
7.26.1 Detailed Description . . . . .	129
7.27 src/color.cpp File Reference . . . . .	129
7.27.1 Detailed Description . . . . .	130
7.28 src/entity.cpp File Reference . . . . .	130
7.28.1 Detailed Description . . . . .	131
7.29 src/evade.cpp File Reference . . . . .	131
7.29.1 Detailed Description . . . . .	132
7.30 src/flee.cpp File Reference . . . . .	132
7.30.1 Detailed Description . . . . .	133
7.31 src/flock.cpp File Reference . . . . .	133
7.31.1 Detailed Description . . . . .	134
7.32 src/flowField.cpp File Reference . . . . .	134
7.32.1 Detailed Description . . . . .	135
7.33 src/graphics.cpp File Reference . . . . .	135
7.33.1 Detailed Description . . . . .	136
7.34 src/leaderFollower.cpp File Reference . . . . .	136
7.34.1 Detailed Description . . . . .	137
7.35 src/mouseFollower.cpp File Reference . . . . .	137

7.35.1 Detailed Description	138
7.36 src/obstacle.cpp File Reference	138
7.36.1 Detailed Description	139
7.37 src/obstacleAvoidance.cpp File Reference	139
7.37.1 Detailed Description	140
7.38 src/path.cpp File Reference	140
7.38.1 Detailed Description	141
7.39 src/pathFollower.cpp File Reference	141
7.39.1 Detailed Description	142
7.40 src/point.cpp File Reference	142
7.40.1 Detailed Description	143
7.41 src/prison.cpp File Reference	143
7.41.1 Detailed Description	144
7.41.2 Macro Definition Documentation	144
7.41.2.1 DISTANCE	145
7.41.2.2 WALL	145
7.42 src/pursuit.cpp File Reference	145
7.42.1 Detailed Description	146
7.43 src/pvector.cpp File Reference	146
7.43.1 Detailed Description	146
7.44 src/random.cpp File Reference	147
7.44.1 Detailed Description	147
7.45 src/scenario.cpp File Reference	147
7.45.1 Detailed Description	148
7.45.2 Macro Definition Documentation	148
7.45.2.1 MAX_NUMBER_OF_AGENTS	148
7.46 src/steeringBehavior.cpp File Reference	149
7.46.1 Detailed Description	149
7.47 src/wander.cpp File Reference	150
7.47.1 Detailed Description	150
7.48 src/windy.cpp File Reference	151
7.48.1 Detailed Description	151
7.49 test/test_suites.cpp File Reference	152
7.49.1 Detailed Description	153
7.49.2 Macro Definition Documentation	153
7.49.2.1 BOOST_TEST_MODULE	153
7.49.3 Function Documentation	153
7.49.3.1 BOOST_AUTO_TEST_CASE() [1/12]	153
7.49.3.2 BOOST_AUTO_TEST_CASE() [2/12]	154
7.49.3.3 BOOST_AUTO_TEST_CASE() [3/12]	154
7.49.3.4 BOOST_AUTO_TEST_CASE() [4/12]	154
7.49.3.5 BOOST_AUTO_TEST_CASE() [5/12]	154

---

7.49.3.6 BOOST_AUTO_TEST_CASE() [6/12] . . . . .	155
7.49.3.7 BOOST_AUTO_TEST_CASE() [7/12] . . . . .	155
7.49.3.8 BOOST_AUTO_TEST_CASE() [8/12] . . . . .	155
7.49.3.9 BOOST_AUTO_TEST_CASE() [9/12] . . . . .	156
7.49.3.10 BOOST_AUTO_TEST_CASE() [10/12] . . . . .	156
7.49.3.11 BOOST_AUTO_TEST_CASE() [11/12] . . . . .	156
7.49.3.12 BOOST_AUTO_TEST_CASE() [12/12] . . . . .	156

<b>Index</b>	<b>157</b>
--------------	------------



# Chapter 1

## Intent

- 1- implementing ai using autonomous steering agents
- 2- implementing smart agents using genetics algorithms
- 3- implementing smart agents using 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

Jan Schiffmann : Nature of Code

Fernando Bevilacqua : Understanding Steering Behaviors

Jer Thorp : Living in Data

### 1.3 Links

<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:

color . . . . .	21
entity . . . . .	24
agent . . . . .	13
obstacle . . . . .	46
path . . . . .	51
flowField . . . . .	31
graphics . . . . .	33
point . . . . .	55
pvector . . . . .	64
random . . . . .	73
scenario . . . . .	74
evade . . . . .	27
flee . . . . .	28
flock . . . . .	30
leaderFollower . . . . .	42
mouseFollower . . . . .	45
obstacleAvoidance . . . . .	48
pathFollower . . . . .	53
prison . . . . .	61
pursuit . . . . .	62
wander . . . . .	87
windy . . . . .	88
steeringBehavior . . . . .	78



## Chapter 4

# Class Index

### 4.1 Class List

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

agent	13
color	21
entity	24
evade	27
flee	28
flock	30
flowField	31
graphics	33
leaderFollower	42
mouseFollower	45
obstacle	46
obstacleAvoidance	48
path	51
pathFollower	53
point	55
prison	61
pursuit	62
pvector	64
random	73
scenario	74
steeringBehavior	78
wander	87
windy	88



## 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	126
<a href="#">include/agent.h</a>	
Agent class defines all agent specifications	91
<a href="#">include/color.h</a>	
Color class used for agent, path, wall etc. color	92
<a href="#">include/entity.h</a>	95
<a href="#">include/evade.h</a>	
Evade class inherited from scenario class	96
<a href="#">include/flee.h</a>	
Agents flee from mouse scenario	97
<a href="#">include/flock.h</a>	
Flocking agents scenario	98
<a href="#">include/flowField.h</a>	
FlowField class, screen can be filled with a force for each pixel	100
<a href="#">include/graphics.h</a>	
Graphics class, drives openGL	102
<a href="#">include/leaderFollower.h</a>	
Agents follow leader scenario	104
<a href="#">include/mouseFollower.h</a>	
Agents follow mouse scenario	105
<a href="#">include/obstacle.h</a>	
Circular obstacles for agent avoidance behaviors	106
<a href="#">include/obstacleAvoidance.h</a>	
Agents avoid from obstacles scenario	108
<a href="#">include/path.h</a>	
Path class used for path following steering behaviors	109
<a href="#">include/pathFollower.h</a>	
Path following scenario	110
<a href="#">include/point.h</a>	
Point class used for point operations	112
<a href="#">include/prison.h</a>	
Agents cant escape from field scenario	113
<a href="#">include/pursuit.h</a>	
One agent pursue other one scenario	114

include/ <a href="#">pvector.h</a>	Pvector class used for 2D vector operations . . . . .	116
include/ <a href="#">random.h</a>	Utility class for random operations . . . . .	117
include/ <a href="#">scenario.h</a>	Base class for all scenarios . . . . .	118
include/ <a href="#">steeringBehavior.h</a>	Functions for autonomous steering behaviors . . . . .	119
include/ <a href="#">wander.h</a>	Random wandering agents scenario . . . . .	123
include/ <a href="#">windy.h</a>	Windy air scenario . . . . .	124
src/ <a href="#">agent.cpp</a>	Implementation of the agent class . . . . .	128
src/ <a href="#">color.cpp</a>	Color class implementation . . . . .	129
src/ <a href="#">entity.cpp</a>	Entity class implementation . . . . .	130
src/ <a href="#">evade.cpp</a>	Evade class implementation . . . . .	131
src/ <a href="#">flee.cpp</a>	Flee class implementation . . . . .	132
src/ <a href="#">flock.cpp</a>	Flock class implementation . . . . .	133
src/ <a href="#">flowField.cpp</a>	FlowField class implementation . . . . .	134
src/ <a href="#">graphics.cpp</a>	Graphics class implementation . . . . .	135
src/ <a href="#">leaderFollower.cpp</a>	LeaderFollower class implementation . . . . .	136
src/ <a href="#">mouseFollower.cpp</a>	MouseFollower class implementation . . . . .	137
src/ <a href="#">obstacle.cpp</a>	Obstacle class implementation . . . . .	138
src/ <a href="#">obstacleAvoidance.cpp</a>	ObstacleAvoidance class implementation . . . . .	139
src/ <a href="#">path.cpp</a>	Path class implementation . . . . .	140
src/ <a href="#">pathFollower.cpp</a>	PathFollower class implementation . . . . .	141
src/ <a href="#">point.cpp</a>	Point class implementation file . . . . .	142
src/ <a href="#">prison.cpp</a>	Prison class implementation . . . . .	143
src/ <a href="#">pursuit.cpp</a>	Prison class implementation . . . . .	145
src/ <a href="#">pvector.cpp</a>	Pvector class implementation . . . . .	146
src/ <a href="#">random.cpp</a>	Utility class for random operations . . . . .	147
src/ <a href="#">scenario.cpp</a>	Scenario base class implementation . . . . .	147
src/ <a href="#">steeringBehavior.cpp</a>	Implementation of autonomous steering behaviors . . . . .	149
src/ <a href="#">wander.cpp</a>	Wander class implementation . . . . .	150
src/ <a href="#">windy.cpp</a>	Windy class implementation . . . . .	151



test/ <a href="#">test_suites.cpp</a>	
Unit test suites . . . . .	<a href="#">152</a>



## Chapter 6

# Class Documentation

### 6.1 agent Class Reference

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

#### 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*
- `string getName ()`  
*name attribute getter*
- `void setName (string n)`  
*name attribute setter*
- `float getMass ()`  
*mass attribute getter*
- `void setMass (float m)`  
*mass attribute setter*
- `void draw (graphics view) override`  
*agent drawing*

## Public Attributes

- [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*
- int [id](#)  
*id of the agent*
- bool [arrive](#) = false  
*has arriving behavior or not*

### 6.1.1 Detailed Description

Definition at line 21 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 37 of file agent.cpp.

```

38 {
39     position      = point(x, y);
40     velocity      = pvector(0.6, 0.0);
41     acceleration  = pvector(0.0, 0.0);
42     steering      = pvector(0.0, 0.0);
43     desiredVelocity = pvector(0.0, 0.0);
44     force         = pvector(0.0, 0.0);
45     targetPoint   = point(0.0, 0.0);
46     entityColor   = RED;
47 }
```

**6.1.2.3 ~agent()**

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

destructor

Definition at line 79 of file agent.cpp.

```

80 {
81
82 }
```

**6.1.3 Member Function Documentation****6.1.3.1 draw()**

```
void agent::draw (
    graphics view ) [override], [virtual]
```

agent drawing

**Parameters**

<i>view</i>	graphics to draw
-------------	------------------

Implements [entity](#).

Definition at line 84 of file agent.cpp.

```

84         {
85     this->updatePosition(this->arrive);
86     view.drawAgent(*this);
87 }

```

### 6.1.3.2 getMass()

```
float agent::getMass ( )
```

mass attribute getter

Definition at line 29 of file agent.cpp.

```

29     {
30     return mass;
31 }

```

### 6.1.3.3 getName()

```
string agent::getName ( )
```

name attribute getter

Definition at line 21 of file agent.cpp.

```

21     {
22     return name;
23 }

```

### 6.1.3.4 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 71 of file agent.cpp.

```

72 {
73     this->maxSpeed = s;
74     this->maxForce = f;

```

```
75     this->r = r;  
76     this->mass = m;  
77 }
```

#### 6.1.3.5 setMass()

```
void agent::setMass (  
    float m )
```

mass attribute setter

##### Parameters

<i>m</i>	set value
----------	-----------

Definition at line 33 of file agent.cpp.

```
33     {  
34         mass = m;  
35     }
```

#### 6.1.3.6 setName()

```
void agent::setName (  
    string n )
```

name attribute setter

##### Parameters

<i>n</i>	set value
----------	-----------

Definition at line 25 of file agent.cpp.

```
25     {  
26         name = n;  
27     }
```

#### 6.1.3.7 updatePosition()

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

position update calculations

##### Parameters

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



See also

[agent\(\)](#)

Definition at line 49 of file agent.cpp.

```
50 {  
51     force.limit(maxForce);  
52     acceleration = force;  
53     velocity += acceleration;  
54  
55     //arriving behavior implementation  
56     if(arrive == true){  
57         pvector diff = targetPoint - position;  
58         if(diff.magnitude() > r)  
59             velocity.limit(maxSpeed);  
60         else  
61             velocity.limit(maxSpeed * diff.magnitude() / r);  
62     }  
63     else{  
64         velocity.limit(maxSpeed);  
65     }  
66  
67     position = position + velocity;  
68     force = pvector(0,0);  
69 }
```

## 6.1.4 Member Data Documentation

### 6.1.4.1 acceleration

`pvector agent::acceleration`

acceleration of the agent

Definition at line 124 of file agent.h.

### 6.1.4.2 arrive

`bool agent::arrive = false`

has arriving behavior or not

Definition at line 144 of file agent.h.

### 6.1.4.3 desiredVelocity

`pvector agent::desiredVelocity`

desired velocity of the agent

Definition at line 129 of file agent.h.

#### 6.1.4.4 force

```
pvector agent::force
```

force of the agent

Definition at line 119 of file agent.h.

#### 6.1.4.5 id

```
int agent::id
```

id of the agent

Definition at line 139 of file agent.h.

#### 6.1.4.6 maxForce

```
float agent::maxForce
```

maximum force of the agent

Definition at line 109 of file agent.h.

#### 6.1.4.7 maxSpeed

```
float agent::maxSpeed
```

maximum speed of the agent

Definition at line 104 of file agent.h.

#### 6.1.4.8 position

```
point agent::position
```

position of the agent

Definition at line 89 of file agent.h.

#### 6.1.4.9 r

```
float agent::r
```

radius of the agent

Definition at line 134 of file agent.h.

#### 6.1.4.10 steering

```
pvector agent::steering
```

steering force of the apply

Definition at line 114 of file agent.h.

#### 6.1.4.11 targetPoint

```
point agent::targetPoint
```

target of the agent

Definition at line 99 of file agent.h.

#### 6.1.4.12 velocity

```
pvector agent::velocity
```

velocity of the agent

Definition at line 94 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>
```

## Public Member Functions

- [color](#) ()  
*default constructor.*
- [color](#) (float r, float g, float b)  
*constructor.*

## Static Public Member Functions

- static [color](#) [getColor](#) (int index)  
*gets colorbar colors*

## Public Attributes

- float [R](#)  
*portion of red color*
- float [G](#)  
*portion of green color*
- float [B](#)  
*portion of blue color*

### 6.2.1 Detailed Description

Definition at line 23 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 13 of file color.cpp.

```
14 {  
15  
16 }
```

#### 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 19 of file color.cpp.

```
20 {  
21     R = r;  
22     G = g;  
23     B = b;  
24 }
```

## 6.2.3 Member Function Documentation

### 6.2.3.1 getColor()

```
color color::getColor (  
    int index ) [static]
```

gets colorbar colors

### Parameters

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

Definition at line 26 of file color.cpp.

```
26 {  
27     switch (index)  
28     {  
29         case 0: return WHITE; break;  
30         case 1: return BLUE; break;  
31         case 2: return RED; break;  
32         case 3: return YELLOW; break;  
33         case 4: return GREEN; break;  
34         case 5: return BLACK; break;  
35         case 6: return CYAN; break;  
36         case 7: return MAGENDA; break;  
37     }  
38     return RED;  
39 }  
40 }
```

## 6.2.4 Member Data Documentation

#### 6.2.4.1 B

```
float color::B
```

portion of blue color

Definition at line 53 of file color.h.

#### 6.2.4.2 G

```
float color::G
```

portion of green color

Definition at line 48 of file color.h.

#### 6.2.4.3 R

```
float color::R
```

portion of red color

Definition at line 43 of file color.h.

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

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

## 6.3 entity Class Reference

```
#include <entity.h>
```

### Public Member Functions

- [entity](#) ()  
*default constructor.*
- string [getName](#) ()  
*getter of the name*
- void [setName](#) (string name)  
*name attribute setter*
- int [getId](#) ()  
*getter of the id attribute*
- void [setId](#) (int id)  
*id attribute setter*
- virtual void [draw](#) ([graphics](#) view)=0  
*overriden by child classes*

## Public Attributes

- [color](#) `entityColor`  
*color of the entity*

### 6.3.1 Detailed Description

Definition at line 10 of file `entity.h`.

### 6.3.2 Constructor & Destructor Documentation

#### 6.3.2.1 `entity()`

```
entity::entity ( )
```

default constructor.

Definition at line 10 of file `entity.cpp`.

```
10         {  
11     entityColor = RED;  
12 }
```

### 6.3.3 Member Function Documentation

#### 6.3.3.1 `draw()`

```
virtual void entity::draw (  
    graphics view ) [pure virtual]
```

overridden by child classes

##### Parameters

<code>view</code>	<code>graphics</code>
-------------------	-----------------------

Implemented in [agent](#), [obstacle](#), and [path](#).

#### 6.3.3.2 `getId()`

```
int entity::getId ( )
```

getter of the id attribute

Definition at line 22 of file entity.cpp.

```
22         {  
23     return id;  
24 }
```

#### 6.3.3.3 getName()

```
string entity::getName ( )
```

getter of the name

Definition at line 14 of file entity.cpp.

```
14         {  
15     return name;  
16 }
```

#### 6.3.3.4 setId()

```
void entity::setId (  
    int id )
```

id attribute setter

##### Parameters

<i>id</i>	setter
-----------	--------

Definition at line 26 of file entity.cpp.

```
26         {  
27     this->id = id;  
28 }
```

#### 6.3.3.5 setName()

```
void entity::setName (  
    string name )
```

name attribute setter

##### Parameters

<i>name</i>	setter
-------------	--------

Definition at line 18 of file entity.cpp.

```
18 {
```



```
19     this->name = name;  
20 }
```

## 6.3.4 Member Data Documentation

### 6.3.4.1 entityColor

`color` `entity::entityColor`

color of the entity

Definition at line 48 of file `entity.h`.

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

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

## 6.4 evade Class Reference

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

### Public Member Functions

- `evade ()`  
*default constructor.*

### Static Public Member Functions

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

### Additional Inherited Members

#### 6.4.1 Detailed Description

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

#### 6.4.2 Constructor & Destructor Documentation

### 6.4.2.1 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 }
```

## 6.4.3 Member Function Documentation

### 6.4.3.1 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).getName() == "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](#)

## 6.5 flee Class Reference

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

### Public Member Functions

- [flee \(\)](#)  
*default constructor.*

## Static Public Member Functions

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

## Additional Inherited Members

### 6.5.1 Detailed Description

Definition at line 14 of file flee.h.

### 6.5.2 Constructor & Destructor Documentation

#### 6.5.2.1 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 }
```

### 6.5.3 Member Function Documentation

#### 6.5.3.1 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](#)

## 6.6 flock Class Reference

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

### Public Member Functions

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

### Static Public Member Functions

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

### Additional Inherited Members

#### 6.6.1 Detailed Description

Definition at line 15 of file flock.h.

#### 6.6.2 Constructor & Destructor Documentation

##### 6.6.2.1 flock()

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

default constructor.

Definition at line 38 of file flock.cpp.

```
39 {  
40     int agentCount = 50;  
41     float maxForce = 0.3;  
42     float maxSpeed = 0.8;  
43     name = "flocking agents";  
44     createAgent(RANDOM, &agentCount, &maxForce, &maxSpeed);  
45     callback = reinterpret_cast <void(*)()> ( (void *)(&loop) );  
46 }
```

#### 6.6.3 Member Function Documentation

### 6.6.3.1 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).targetPoint = view.getMousePosition();
28         pvector seek = behavior.seek(*it);
29         seek.mul(0.01);
30
31         (*it).force = sep + ali + coh + seek;
32         (*it).arrive = true;
33     }
34
35     refresh();
36 }
```

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

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

## 6.7 flowField Class Reference

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

### Public Member Functions

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

### 6.7.1 Detailed Description

Definition at line 18 of file flowField.h.

## 6.7.2 Constructor & Destructor Documentation

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

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

default constructor.

See also

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

Definition at line 15 of file flowField.cpp.

```
16 {  
17  
18 }
```

### 6.7.2.2 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 }
```

## 6.7.3 Member Function Documentation

### 6.7.3.1 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 }
```

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

- include/flowField.h
- src/flowField.cpp

## 6.8 graphics Class Reference

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

### Public Member Functions

- void [drawAgent](#) ([agent](#) &[agent](#))  
*drawing with corresponding angle*
- void [drawLine](#) ([point](#) p1, [point](#) p2, [color](#) cl)  
*drawing line*
- void [drawPath](#) ([path](#) &[path](#))  
*draws path*
- void [drawPoint](#) ([point](#) p)  
*draws point*
- void [drawCircle](#) ([point](#) p, float radius, [color](#) color)  
*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*

### 6.8.1 Detailed Description

Definition at line 22 of file graphics.h.

### 6.8.2 Member Function Documentation

#### 6.8.2.1 drawAgent()

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

drawing with corresponding angle

#### Parameters

<a href="#">agent</a>	instance to change
-----------------------	--------------------

Definition at line 159 of file graphics.cpp.

```
160 {
161     glPushMatrix();
162     glTranslatef(agent.position.x, agent.position.y, 0.0f);
163     glRotatef(agent.velocity.getAngle\(\), 0.0f, 0.0f, 1.0f);
164     glBegin(GL_TRIANGLES);
165     glColor3f( agent.entityColor.R, agent.entityColor.G, agent.entityColor.B);
166     glVertex3f( 1.0f, 0.0f, 0.0f);
167     glVertex3f(-1.0f, 0.5f, 0.0f);
168     glVertex3f(-1.0f, -0.5f, 0.0f);
169     glEnd();
170     glPopMatrix();
```



```
171 }
```

### 6.8.2.2 drawCircle()

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

draws circle

#### Parameters

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

Definition at line 136 of file graphics.cpp.

```
137 {
138     glColor3f(color.R, color.G, color.B);
139     glBegin(GL_LINE_STRIP);
140     glLineWidth(2);
141     for (int i = 0; i <= 300; i++) {
142         float angle = 2 * PI * i / 300;
143         float x = cos(angle) * radius;
144         float y = sin(angle) * radius;
145         glVertex2d(p.x + x, p.y + y);
146     }
147     glEnd();
148 }
```

### 6.8.2.3 drawLine()

```
void graphics::drawLine (
    point p1,
    point p2,
    color cl )
```

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 126 of file graphics.cpp.

```
127 {
128     glColor3f( cl.R, cl.G, cl.B);
129     glLineWidth(2);
130     glBegin(GL_LINES);
131     glVertex2f(p1.x, p1.y);
132     glVertex2f(p2.x, p2.y);
```

```

133     glEnd();
134 }

```

#### 6.8.2.4 drawPath()

```

void graphics::drawPath (
    path & path )

```

draws path

##### Parameters

<i>path</i>	to draw
-------------	---------

Definition at line 112 of file graphics.cpp.

```

113 {
114     point p1, p2;
115     for(auto it = path.points.begin(); it < path.points.end()-1; it++){
116         p1 = point((*it).x, (*it).y - path.width/2) ;
117         p2 = point((*it+1).x, (*it+1).y - path.width/2);
118         drawLine(p1, p2, path.entityColor);
119
120         p1 = point((*it).x, (*it).y + path.width/2) ;
121         p2 = point((*it+1).x, (*it+1).y + path.width/2);
122         drawLine(p1, p2, path.entityColor);
123     }
124 }

```

#### 6.8.2.5 drawPoint()

```

void graphics::drawPoint (
    point p )

```

draws point

##### Parameters

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

Definition at line 150 of file graphics.cpp.

```

151 {
152     glColor3f(1,1,1);
153     glPointSize(4.0);
154     glBegin(GL_POINTS);
155     glVertex2f(p.x, p.y);
156     glEnd();
157 }

```

#### 6.8.2.6 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(p.x, p.y);
25     for ( string::iterator it=text.begin(); it!=text.end(); ++it){
26         glutBitmapCharacter(GLUT_BITMAP_9_BY_15, *it);
27     }
28 }
```

#### 6.8.2.7 forceInScreen()

```

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

changes agent position so that it stays in screen

#### Parameters

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

Definition at line 61 of file graphics.cpp.

```

62 {
63     if (agent.position.x > WIDTH)
64         agent.position.x -= 2 * WIDTH;
65     if (agent.position.x < -WIDTH)
66         agent.position.x += 2 * WIDTH;
67     if (agent.position.y > HEIGHT)
68         agent.position.y -= 2 * HEIGHT;
69     if (agent.position.y < -HEIGHT)
70         agent.position.y += 2 * HEIGHT;
71 }
```

#### 6.8.2.8 getMousePosition()

```

point graphics::getMousePosition ( )
```

gets mouse position

#### Returns

mouse point

Definition at line 56 of file graphics.cpp.

```

57 {
58     return point (graphics::target_x, graphics::target_y);
59 }
```

### 6.8.2.9 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 105 of file graphics.cpp.

```
106 {
107     if (key == ESC){
108         exit(0);
109     }
110 }
```

### 6.8.2.10 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 81 of file graphics.cpp.

```
82 {
83     glViewport(0, 0, w, h); //Tell OpenGL how to convert from coordinates to pixel values
84     glMatrixMode(GL_PROJECTION); //Switch to setting the camera perspective
85     glLoadIdentity(); //Reset the camera
86     //Set the camera perspective
87     gluPerspective(45.0, //The camera angle
88                   (double)w / (double)h, //The width-to-height ratio
89                   1.0, //The near z clipping coordinate
90                   200.0); //The far z clipping coordinate
91 }
```

### 6.8.2.11 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 39 of file graphics.cpp.

```

40 {
41     glutInit(argv, argc);
42     glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
43     glutInitWindowSize(400, 400);
44     glutCreateWindow("Autonomous Steering Agents");
45     glClearColor(0.7f, 0.7f, 0.7f, 1.0f); //set background color
46     glEnable(GL_DEPTH_TEST);
47     glutDisplayFunc(*callback);
48     glutMouseFunc(graphics::mouseButton);
49     glutPassiveMotionFunc(graphics::mouseMove);
50     glutKeyboardFunc(graphics::handleKeypress);
51     glutReshapeFunc(graphics::handleResize);
52     glutTimerFunc(20, graphics::timerEvent, 0);
53     glutMainLoop();
54 }
```

### 6.8.2.12 mouseButton()

```

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

mouse press event

## Parameters

<i>button</i>	mouse key pressed
<i>state</i>	down/up etc.
<i>x</i>	unused but required for openGL
<i>y</i>	unused but required for openGL

Definition at line 99 of file graphics.cpp.

```

100 {
101     if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN){
102     }
103 }
```

### 6.8.2.13 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 73 of file graphics.cpp.

```

74 {
75     //TODO: mouse position to glut
76     //TODO: magic numbers
77     graphics::target_x = x / 5.88 - 34;
78     graphics::target_y = 34 - y / 5.88;
79 }
```

### 6.8.2.14 refreshScene()

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

update agent position

Definition at line 30 of file graphics.cpp.

```

31 {
32     glutSwapBuffers();
33     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
34     glMatrixMode(GL_MODELVIEW); //Switch to the drawing perspective
35     glLoadIdentity(); //Reset the drawing perspective
36     glTranslatef(0.0f, 0.0f, -85.0f); //Move to the center of the triangle
37 }
```

### 6.8.2.15 timerEvent()

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

periodic timer event

## Parameters

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

Definition at line 93 of file graphics.cpp.

```

94 {
95     glutPostRedisplay(); //Tell GLUT that the display has changed
96     glutTimerFunc(value, timerEvent, 20);
97 }
```

## 6.8.3 Member Data Documentation

### 6.8.3.1 target\_x

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

mouse position x

Definition at line 130 of file graphics.h.

### 6.8.3.2 target\_y

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

mouse position y

Definition at line 135 of file graphics.h.

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

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

## 6.9 leaderFollower Class Reference

```
#include <leaderFollower.h>
```

### Public Member Functions

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

### Static Public Member Functions

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

### Static Public Attributes

- static [pvector](#) [leaderVelocity](#)  
*leader velocity*
- static [point](#) [leaderPosition](#)  
*leader position*



## Additional Inherited Members

### 6.9.1 Detailed Description

Definition at line 14 of file leaderFollower.h.

### 6.9.2 Constructor & Destructor Documentation

#### 6.9.2.1 leaderFollower()

```
leaderFollower::leaderFollower ( )
```

default constructor.

Definition at line 53 of file leaderFollower.cpp.

```
54 {  
55     int agentCount = 10;  
56     float maxForce = 0.4;  
57     float maxSpeed = 0.4;  
58     name = "leader following";  
59  
60     //todo: refactor leader creation  
61     agent agent1 {-10.0, 0.0};  
62     agent1.id = 1;  
63     agent1.setName("leader");  
64     agent1.entityColor = BLUE;  
65     agent1.setFeatures(0.8, 0.4, 5, 1);  
66     agents.push_back(agent1);  
67  
68  
69     createAgent(RANDOM, &agentCount, &maxForce, &maxSpeed);  
70     callback = reinterpret_cast<void(*)()> ( (void *)(&loop) );  
71 }
```

### 6.9.3 Member Function Documentation

#### 6.9.3.1 loop()

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

leader following scenario loop function

**Note**

opengl callback forces that function to be static

Definition at line 18 of file leaderFollower.cpp.

```

19 {
20     for(auto it = agents.begin(); it < agents.end(); it++){
21         if((*it).getName() == "leader"){
22             (*it).targetPoint = view.getMousePosition();
23             (*it).force = behavior.seek(*it);
24             leaderVelocity = (*it).velocity;
25             leaderVelocity.mul(-1);
26             leaderVelocity.normalize().mul(20);
27             leaderPosition = (*it).position;
28         }
29         else{
30             (*it).targetPoint = leaderPosition + leaderVelocity;
31             view.drawCircle((*it).targetPoint, 8, RED);
32
33             pvector sep = behavior.separation(agents, *it);
34             sep.mul(15);
35             (*it).force = sep;
36
37             pvector leaderDiff = (*it).position - leaderPosition;
38             if(leaderDiff.magnitude() < 5){
39                 pvector fle = behavior.evade(agents, *it, view, "leader");
40                 fle.mul(40);
41                 (*it).force += fle;
42             }
43
44             pvector diff = (*it).position - (*it).targetPoint;
45             if(diff.magnitude() > 5) { (*it).force += behavior.seek(*it); }
46             else { (*it).velocity = pvector(0,0); }
47         }
48         (*it).arrive = true;
49     }
50     refresh();
51 }
```

## 6.9.4 Member Data Documentation

### 6.9.4.1 leaderPosition

```
pvector leaderFollower::leaderPosition [static]
```

leader position

Definition at line 24 of file leaderFollower.h.

### 6.9.4.2 leaderVelocity

```
pvector leaderFollower::leaderVelocity [static]
```

leader velocity

Definition at line 19 of file leaderFollower.h.

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

- include/leaderFollower.h
- src/leaderFollower.cpp

## 6.10 mouseFollower Class Reference

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

### Public Member Functions

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

### Static Public Member Functions

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

### Additional Inherited Members

#### 6.10.1 Detailed Description

Definition at line 14 of file mouseFollower.h.

#### 6.10.2 Constructor & Destructor Documentation

##### 6.10.2.1 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 }
```

#### 6.10.3 Member Function Documentation

### 6.10.3.1 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

## 6.11 obstacle Class Reference

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

### Public Member Functions

- [obstacle](#) ()  
*default constructor.*
- [obstacle](#) (point p, float r)  
*constructor*
- void [draw](#) (graphics view) override  
*overriden draw implementation*

### Public Attributes

- [point](#) p  
*center point of the obstacle*
- float r  
*radius of the obstacle*

### 6.11.1 Detailed Description

Definition at line 15 of file obstacle.h.

## 6.11.2 Constructor & Destructor Documentation

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

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

default constructor.

See also

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

Definition at line 16 of file obstacle.cpp.

```
17 {  
18     p = point(0,0);  
19     r = 5;  
20     entityColor = RED;  
21 }
```

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

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

constructor

Parameters

$p$	center of the circular obstacle
$r$	radius of the obstacle

See also

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

Definition at line 23 of file obstacle.cpp.

```
24 {  
25     this->p = p;  
26     this->r = r;  
27     entityColor = RED;  
28 }
```

## 6.11.3 Member Function Documentation

### 6.11.3.1 draw()

```
void obstacle::draw (
    graphics view ) [override], [virtual]
```

overriden draw implementation

Implements [entity](#).

Definition at line 30 of file obstacle.cpp.

```
30     {
31         view.drawCircle(p, r, entityColor);
32     }
```

## 6.11.4 Member Data Documentation

### 6.11.4.1 p

```
point obstacle::p
```

center point of the obstacle

Definition at line 34 of file obstacle.h.

### 6.11.4.2 r

```
float obstacle::r
```

radius of the obstacle

Definition at line 39 of file obstacle.h.

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

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

## 6.12 obstacleAvoidance Class Reference

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

### 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

### 6.12.1 Detailed Description

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

### 6.12.2 Constructor & Destructor Documentation

#### 6.12.2.1 obstacleAvoidance()

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

default constructor.

Definition at line 42 of file `obstacleAvoidance.cpp`.

```
43 {
44     name = "avoid obstacles";
45     createAgent(STATIC, nullptr, nullptr, nullptr);
46     createObstacle(obstacles);
47     callback = reinterpret_cast <void(*)()> ( (void *)(&loop) );
48 }
```

### 6.12.3 Member Function Documentation

#### 6.12.3.1 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 35 of file obstacleAvoidance.cpp.

```

36 {
37     obstacles.push_back(obstacle(point(0,0), 8));
38     obstacles.push_back(obstacle(point(-20,0), 3));
39     obstacles.push_back(obstacle(point(20,-10), 4));
40 }
```

### 6.12.3.2 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         (*it).targetPoint = view.getMousePosition();
21         pvector seek = behavior.seek(*it);
22         seek.mul(0.5);
23
24         pvector avoid = behavior.avoid(obstacles, *it);
25         (*it).force = avoid + seek;
26         (*it).arrive = true;
27
28         for(auto it = obstacles.begin(); it < obstacles.end(); it++){
29             (*it).draw(view);
30         }
31     }
32     refresh();
33 }
```

## 6.12.4 Member Data Documentation

### 6.12.4.1 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](#)



## 6.13 path Class Reference

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

### Public Member Functions

- [path](#) ()  
*default constructor.*
- [path](#) (float [width](#))  
*donstructor.*
- void [addPoint](#) ([point](#) p)  
*adds a new point to the path*
- void [draw](#) ([graphics](#) view)  
*overriden draw implementation*

### Public Attributes

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

#### 6.13.1 Detailed Description

Definition at line 17 of file path.h.

#### 6.13.2 Constructor & Destructor Documentation

##### 6.13.2.1 [path\(\)](#) [1/2]

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

default constructor.

See also

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

Definition at line 16 of file path.cpp.

```
17 {  
18     entityColor = BLUE;  
19     width = 8;  
20 }
```

##### 6.13.2.2 [path\(\)](#) [2/2]

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

donstructor.

**Parameters**

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

**See also**

[path\(\)](#)

Definition at line 22 of file path.cpp.

```

23 {
24     this->width = width;
25     entityColor = BLUE;
26 }
```

**6.13.3 Member Function Documentation****6.13.3.1 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 }
```

**6.13.3.2 draw()**

```

void path::draw (
    graphics view ) [virtual]
```

overriden draw implementation

Implements [entity](#).

Definition at line 28 of file path.cpp.

```

28     {
29     view.drawPath(*this);
30 }
```

## 6.13.4 Member Data Documentation

### 6.13.4.1 points

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

list of points added to the path

Definition at line 41 of file path.h.

### 6.13.4.2 width

```
int path::width
```

width of the path

Definition at line 46 of file path.h.

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

- include/path.h
- src/path.cpp

## 6.14 pathFollower Class Reference

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

### 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

### 6.14.1 Detailed Description

Definition at line 14 of file pathFollower.h.

### 6.14.2 Constructor & Destructor Documentation

#### 6.14.2.1 pathFollower()

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

default constructor.

Definition at line 38 of file pathFollower.cpp.

```
39 {
40     int agentCount = 40;
41     float maxForce = 0.2;
42     float maxSpeed = 0.4;
43     myPath = path(8);
44     createPath(myPath);
45     name = "path following";
46     createAgent(RANDOM, &agentCount, &maxForce, &maxSpeed);
47     callback = reinterpret_cast<void(*)()> ( (void *)(&loop) );
48 }
```

### 6.14.3 Member Function Documentation

#### 6.14.3.1 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 30 of file pathFollower.cpp.

```
31 {
32     p.addPoint(point(-40, 5));
33     p.addPoint(point(-14, 15));
```

```

34     p.addPoint(point( 10, 7));
35     p.addPoint(point( 40, 12));
36 }

```

### 6.14.3.2 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         pvector flwpth = behavior.stayInPath(*it, myPath, view);
21         pvector sep = behavior.separation(agents, *it);
22         sep.mul(5);
23         (*it).force = sep + flwpth;
24     }
25     myPath.draw(view);
26 }
27 refresh();
28 }

```

## 6.14.4 Member Data Documentation

### 6.14.4.1 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

## 6.15 point Class Reference

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

## 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*

### 6.15.1 Detailed Description

Definition at line 15 of file [point.h](#).

### 6.15.2 Constructor & Destructor Documentation

#### 6.15.2.1 [point\(\)](#) [1/2]

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

default constructor

See also

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

Definition at line 21 of file [point.cpp](#).

```
22 {  
23     x = 0;  
24     y = 0;  
25 }
```

### 6.15.2.2 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 }
```

## 6.15.3 Member Function Documentation

### 6.15.3.1 div()

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

divide point

#### Parameters

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

Definition at line 42 of file point.cpp.

```
43 {
44     x = x / d;
45     y = y / d;
46 }
```

### 6.15.3.2 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 71 of file point.cpp.

```

72 {
73     pvector a = predicted - start;
74     pvector b = end - start;
75     b.normalize();
76     float a_dot_b = a.dotProduct(b);
77     b.mul(a_dot_b);
78     point normalPoint = start + b;
79     this->x = normalPoint.x;
80     this->y = normalPoint.y;
81 }

```

#### 6.15.3.3 mul()

```

void point::mul (
    float d )

```

multiply point

#### Parameters

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

Definition at line 48 of file point.cpp.

```

49 {
50     x = x * d;
51     y = y * d;
52 }

```

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

```

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

```

overloaded + operator

#### Parameters

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



**Returns**

sum

Definition at line 55 of file point.cpp.

```
56 {  
57     point res;  
58     res.x = x + obj.x;  
59     res.y = y + obj.y;  
60     return res;  
61 }
```

**6.15.3.5 operator+() [2/2]**

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

overloaded + operator

**Parameters**

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

**Returns**

sum

Definition at line 27 of file point.cpp.

```
28 {  
29     point res;  
30     res.x = x + obj.x;  
31     res.y = y + obj.y;  
32     return res;  
33 }
```

**6.15.3.6 operator-()**

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

overloaded - operator

**Parameters**

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

**Returns**

difference

Definition at line 63 of file point.cpp.

```

64 {
65     pvector res;
66     res.x = x - obj.x;
67     res.y = y - obj.y;
68     return res;
69 }

```

### 6.15.3.7 operator==()

```

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

```

overloaded == operator

#### Parameters

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

#### Returns

comparison result

Definition at line 35 of file point.cpp.

```

36 {
37     if(x == obj.x && y == obj.y)
38         return true;
39     return false;
40 }

```

### 6.15.3.8 print()

```

void point::print (
    const string & s )

```

debug function

#### Parameters

<i>s</i>	explanation string of the log
----------	-------------------------------

Definition at line 83 of file point.cpp.

```

84 {
85     cout << " " << s << " " << x << " " << y << endl;
86 }

```

## 6.15.4 Member Data Documentation

#### 6.15.4.1 x

```
float point::x
```

x position

Definition at line 88 of file point.h.

#### 6.15.4.2 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](#)

## 6.16 prison Class Reference

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

### Public Member Functions

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

### Static Public Member Functions

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

### Additional Inherited Members

#### 6.16.1 Detailed Description

Definition at line 15 of file prison.h.

#### 6.16.2 Constructor & Destructor Documentation

### 6.16.2.1 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 }
```

## 6.16.3 Member Function Documentation

### 6.16.3.1 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), BLUE);
22         view.drawLine(point(WALL, WALL), point(WALL, -WALL), BLUE);
23         view.drawLine(point(WALL, -WALL), point(-WALL, -WALL), BLUE);
24         view.drawLine(point(-WALL, WALL), point(-WALL, -WALL), BLUE);
25         (*it).force = behavior.stayInArea(*it, WALL - DISTANCE);
26         (*it).force += behavior.separation(agents, *it);
27     }
28     refresh();
29 }
```

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

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

## 6.17 pursuit Class Reference

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

## Public Member Functions

- [pursuit](#) ()

*default constructor.*

## Static Public Member Functions

- static void [loop](#) ()

*pursuing scenario loop function*

## Additional Inherited Members

### 6.17.1 Detailed Description

Definition at line 14 of file pursuit.h.

### 6.17.2 Constructor & Destructor Documentation

#### 6.17.2.1 [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 }
```

### 6.17.3 Member Function Documentation

### 6.17.3.1 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).getName() == "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](#)

## 6.18 pvector Class Reference

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

### 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*

### 6.18.1 Detailed Description

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

### 6.18.2 Constructor & Destructor Documentation

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

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

default constructor

See also

`pvector(float x, float y)`

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

```
36 {
37     x = 0;
38     y = 0;
39 }
```

### 6.18.2.2 pvector() [2/2]

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

constructor

#### Parameters

<i>x</i>	magnitude of the vector
<i>y</i>	magnitude of the vector

See also

[pvector\(\)](#)

Definition at line 41 of file pvector.cpp.

```
42 {
43     this->x = x;
44     this->y = y;
45 }
```

## 6.18.3 Member Function Documentation

### 6.18.3.1 add()

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

addition of vectors

#### Parameters

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

Definition at line 59 of file pvector.cpp.

```
60 {
61     x = x + p.x;
62     y = y + p.y;
63 }
```

### 6.18.3.2 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 }
```

**6.18.3.3 div()**

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

vector division

**Parameters**

$i$	scalar value to divide
-----	------------------------

Definition at line 47 of file pvector.cpp.

```
48 {  
49     x = x / i;  
50     y = y / i;  
51 }
```

**6.18.3.4 dotProduct()**

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

dot product of two vectors

**Parameters**

$v$	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 }

```

### 6.18.3.5 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 }

```

### 6.18.3.6 limit()

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

vector limitation

#### Parameters

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

Definition at line 84 of file pvector.cpp.

```

85 {
86     this->normalize();
87     this->mul(limit);
88 }

```

### 6.18.3.7 magnitude()

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

calculates magnitude of the vector

#### Returns

magnitude of the vector

Definition at line 65 of file pvector.cpp.

```

66 {
67     return sqrt((this->x * this->x) + (this->y * this->y));
68 }

```

### 6.18.3.8 mul()

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

vector multiplication

#### Parameters

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

Definition at line 53 of file pvector.cpp.

```
54 {
55     x = x * i;
56     y = y * i;
57 }
```

### 6.18.3.9 normalize()

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

normalize

#### Returns

normalized vector

Definition at line 70 of file pvector.cpp.

```
71 {
72     float magnitude = this->magnitude();
73     if(magnitude != 0){
74         this->x = this->x / magnitude;
75         this->y = this->y / magnitude;
76     }
77     else{
78         this->x = 0;
79         this->y = 0;
80     }
81     return *this;
82 }
```

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

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

overloaded + operator

#### Parameters

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

**Returns**

sum

Definition at line 112 of file pvector.cpp.

```
113 {  
114     pvector res;  
115     res.x = x + obj.x;  
116     res.y = y + obj.y;  
117     return res;  
118 }
```

**6.18.3.11 operator+() [2/2]**

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

overloaded + operator

**Parameters**

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

**Returns**

sum

Definition at line 90 of file pvector.cpp.

```
91 {  
92     pvector res;  
93     res.x = x + obj.x;  
94     res.y = y + obj.y;  
95     return res;  
96 }
```

**6.18.3.12 operator+=()**

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

overloaded += operator

**Parameters**

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

**Returns**

sum

Definition at line 98 of file pvector.cpp.

```
99 {  
100     x = x + obj.x;  
101     y = y + obj.y;  
102     return *this;  
103 }
```

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

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

overloaded - operator

#### Parameters

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

#### Returns

difference

Definition at line 120 of file pvector.cpp.

```
121 {  
122     pvector res;  
123     res.x = x - obj.x;  
124     res.y = y - obj.y;  
125     return res;  
126 }
```

### 6.18.3.14 operator-() [2/2]

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

overloaded - operator

#### Parameters

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

#### Returns

difference

Definition at line 133 of file pvector.cpp.

```
134 {  
135     pvector res;  
136     res.x = x - obj.x;  
137     res.y = y - obj.y;  
138     return res;  
139 }
```

### 6.18.3.15 operator==()

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

overloaded == operator

#### Parameters

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

#### Returns

comparison result

Definition at line 105 of file pvector.cpp.

```
106 {
107     if (x == obj.x && y == obj.y)
108         return true;
109     return false;
110 }
```

### 6.18.3.16 print()

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

debug function

#### Parameters

<i>s</i>	identification text
----------	---------------------

Definition at line 128 of file pvector.cpp.

```
129 {
130     cout << s << " " << x << " " << y << endl;
131 }
```

## 6.18.4 Member Data Documentation

### 6.18.4.1 x

```
float pvector::x
```

x magnitude of the vector

Definition at line 140 of file pvector.h.

### 6.18.4.2 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](#)

## 6.19 random Class Reference

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

### Static Public Member Functions

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

### 6.19.1 Detailed Description

Definition at line 9 of file random.h.

### 6.19.2 Member Function Documentation

#### 6.19.2.1 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](#)

## 6.20 scenario Class Reference

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

### 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 string [name](#)  
*scenario name*



## 6.20.1 Detailed Description

Definition at line 19 of file scenario.h.

## 6.20.2 Constructor & Destructor Documentation

### 6.20.2.1 scenario()

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

default constructor.

Definition at line 28 of file scenario.cpp.

```
29 {
30     view = graphics();
31 }
```

## 6.20.3 Member Function Documentation

### 6.20.3.1 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("gazelle", "lion");
119     }
120     else{
```

```

121         //error message
122     }
123 }

```

### 6.20.3.2 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 23 of file scenario.cpp.

```

24 {
25     view.initGraphics(argc, argv, callback);
26 }

```

### 6.20.3.3 refresh()

```

void scenario::refresh ( ) [static]

```

refreshes all items

#### Note

opengl callback forces that function to be static

Definition at line 33 of file scenario.cpp.

```

34 {
35     point textPosition = point(-34, 32.25);
36
37     for(auto it = agents.begin(); it < agents.end(); it++){
38         (*it).draw(view);
39     }
40
41     view.drawText(name, textPosition);
42     view.refreshScene();
43 }

```

## 6.20.4 Member Data Documentation

#### 6.20.4.1 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.

#### 6.20.4.2 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.

#### 6.20.4.3 callback

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

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

Definition at line 75 of file scenario.h.

#### 6.20.4.4 name

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

scenario name

##### Note

opengl callback forces that function to be static

Definition at line 70 of file scenario.h.

#### 6.20.4.5 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

## 6.21 steeringBehavior Class Reference

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

### 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*

### 6.21.1 Detailed Description

Definition at line 36 of file steeringBehavior.h.

### 6.21.2 Member Function Documentation

#### 6.21.2.1 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 120 of file steeringBehavior.cpp.

```
121 {
122     float neighborDist = 30;
123     pvector sum {0,0};
124     int count = 0;
125     for(auto it = boids.begin(); it < boids.end(); it++){
126         float d = (agent.position - (*it).position).magnitude();
127         if( (d > 0) && (d < neighborDist) ){
128             sum += (*it).velocity;
129             count++;
130         }
131     }
132     if(count > 0){
133         sum.div(count);
134         sum.normalize().mul(agent.maxSpeed);
135         agent.steering = sum - agent.velocity;
136         return agent.steering;
137     }
138     return pvector(0,0);
139 }
```

#### 6.21.2.2 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 184 of file steeringBehavior.cpp.

```

185 {
186     float dynamic_length = agent.velocity.magnitude() / agent.maxSpeed;
187     pvector vel = agent.velocity;
188     vel.normalize().mul(dynamic_length);
189     pvector ahead = vel + agent.position;
190     vel.mul(6);
191     pvector ahead2 = vel + agent.position;
192     //view.drawPoint(point(ahead.x, ahead.y));
193     //view.drawPoint(point(ahead2.x, ahead2.y));
194
195     for(auto it = obstacles.begin(); it < obstacles.end(); it++){
196         float dist = (ahead - (*it).p).magnitude();
197         float dist2 = (ahead2 - (*it).p).magnitude();
198         if(dist < (*it).r + 2 || dist2 < (*it).r + 2){
199             pvector avoidance = ahead - (*it).p;
200             avoidance.normalize().mul(20);
201             /*a = point(avoidance.x, avoidance.y);
202             view.drawLine(agent.position, agent.position + a, color(0,1,0));*/
203             return avoidance;
204         }
205     }
206     return pvector(0,0);
207 }
```

## 6.21.2.3 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 141 of file steeringBehavior.cpp.

```

142 {
143     float neighborDist = 20;
144     point sum {0,0};
145     int count = 0;
146     for(auto it = boids.begin(); it < boids.end(); it++){
147         float d = (agent.position - (*it).position).magnitude();
148         if( (d > 0) && (d < neighborDist) ){
```

```

149         sum = sum + (*it).position;
150         count++;
151     }
152 }
153 if(count>0){
154     sum.div(count);
155     agent.targetPoint = sum;
156     return seek(agent);
157 }
158 return pvector(0,0);
159 }

```

### 6.21.2.4 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).getName() == name){
52             target = *it;
53         }
54     }
55
56     point p = point(evader.position.x + 2, evader.position.y - 2);
57     //view.drawText(evader.getName(), p);
58     p = point(target.position.x + 2, target.position.y - 2);
59     //view.drawText(target.getName(), 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 }

```

### 6.21.2.5 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 }
```

### 6.21.2.6 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 239 of file steeringBehavior.cpp.



```

240 {
241     //pos_x, pos_y must be non negative integer
242     int pos_x = abs((int)agent.position.x) % WIDTH;
243     int pos_y = abs((int)agent.position.y) % HEIGHT;
244     //TODO: modification required for non uniform fields
245     return flow.getField(pos_x, pos_y);
246 }

```

### 6.21.2.7 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).getName() == name){
79             target = *it;
80         }
81     }
82
83     point p = point(target.position.x + 2, target.position.y - 2);
84     view.drawText(target.getName(), p);
85
86     p = point(pursuer.position.x + 2, pursuer.position.y - 2);
87     view.drawText(pursuer.getName(), p);
88
89     float dist = (target.position - pursuer.position).magnitude();
90     float t = dist / target.maxSpeed;
91
92     pvector targetVel = target.velocity;
93     targetVel.mul(t);
94     point futurePos = target.position + targetVel;
95     pursuer.targetPoint = futurePos;
96     return seek(pursuer);
97 }

```

### 6.21.2.8 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 209 of file steeringBehavior.cpp.

```

210 {
211     agent.desiredVelocity = agent.targetPoint - agent.position;
212     agent.steering = agent.desiredVelocity - agent.velocity;
213     return agent.steering;
214 }
```

**6.21.2.9 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 161 of file steeringBehavior.cpp.

```

162 {
163     float desiredSeparation = 3;
164     pvector sum = pvector(0,0);
165     int count = 0;
166     for(auto it = agents.begin(); it < agents.end(); it++){
167         float d = (agent.position - (*it).position).magnitude();
168         if( (d > 0) && (d < desiredSeparation) ){
169             pvector diff = agent.position - (*it).position;
170             diff.normalize().div(d);
171             sum = sum + diff;
172             count++;
173         }
174     }
175     if(count > 0){
176         sum.div(count);
177         sum.normalize().mul(agent.maxSpeed);
178         agent.steering = sum - agent.velocity;
179         return agent.steering;
180     }
181     return pvector(0,0);
182 }
```

**6.21.2.10 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 }
```

**6.21.2.11 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 248 of file steeringBehavior.cpp.

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

```

270     return pvector(0,0);
271 }

```

### 6.21.2.12 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 216 of file steeringBehavior.cpp.

```

217 {
218     float worldRecord = 1000000;
219     point normalPoint, predictedPos, start, end;
220     pvector distance;
221     for(auto it = path.points.begin(); it < path.points.end()-1; it++){
222         start = point((*it).x, (*it).y);
223         end = point((*it+1).x, (*it+1).y);
224         predictedPos = agent.position + agent.velocity;
225         normalPoint.getNormalPoint(predictedPos, start, end);
226         if (normalPoint.x < start.x || normalPoint.x > end.x){
227             normalPoint = end;
228         }
229         distance = predictedPos - normalPoint;
230         if (distance.magnitude() < worldRecord){
231             worldRecord = distance.magnitude();
232             agent.targetPoint = end;
233         }
234         view.drawPoint(agent.targetPoint);
235     }
236     return seek(agent);
237 }

```

### 6.21.2.13 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 99 of file steeringBehavior.cpp.

```

100 {
101     pvector circleCenter = agent.velocity;
102     circleCenter.normalize().mul(CIRCLE_DISTANCE + CIRCLE_RADIUS);
103
104     int wanderAngle = (rand() % 360);
105     pvector displacement {0, 1};
106     setAngle(displacement, wanderAngle);
107     displacement.mul(CIRCLE_RADIUS);
108
109     agent.desiredVelocity = displacement + circleCenter;
110     agent.steering = agent.desiredVelocity - agent.velocity;
111
112     //move it to the center when it is out of screen
113     if(agent.position.x > WIDTH || agent.position.x < -WIDTH ||
114        agent.position.y > HEIGHT || agent.position.y < -HEIGHT)
115         agent.position = point(0,0);
116
117     return agent.steering;
118 }
```

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

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

## 6.22 wander Class Reference

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

### Public Member Functions

- [wander \(\)](#)  
*default constructor*

### Static Public Member Functions

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

### Additional Inherited Members

#### 6.22.1 Detailed Description

Definition at line 14 of file wander.h.

#### 6.22.2 Constructor & Destructor Documentation

### 6.22.2.1 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 }
```

## 6.22.3 Member Function Documentation

### 6.22.3.1 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

## 6.23 windy Class Reference

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

## 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

### 6.23.1 Detailed Description

Definition at line 15 of file windy.h.

### 6.23.2 Constructor & Destructor Documentation

#### 6.23.2.1 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 }
```

### 6.23.3 Member Function Documentation

### 6.23.3.1 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 }
```

## 6.23.4 Member Data Documentation

### 6.23.4.1 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

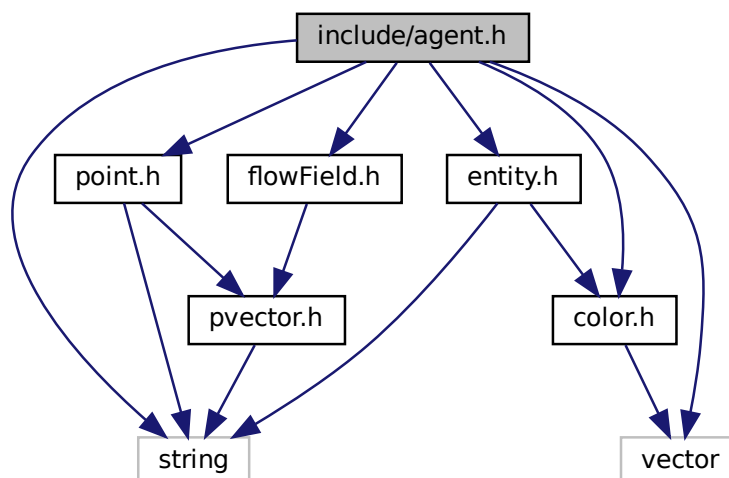
# File Documentation

### 7.1 include/agent.h File Reference

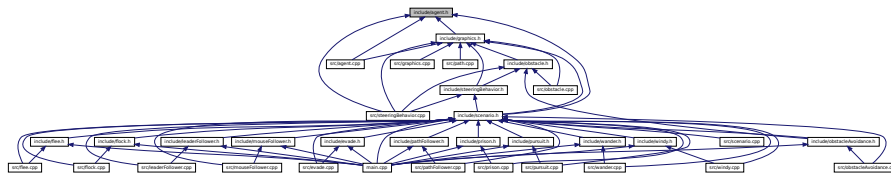
agent class defines all agent specifications

```
#include "point.h"  
#include "color.h"  
#include "entity.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

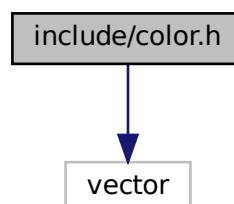
14.05.2021

## 7.2 include/color.h File Reference

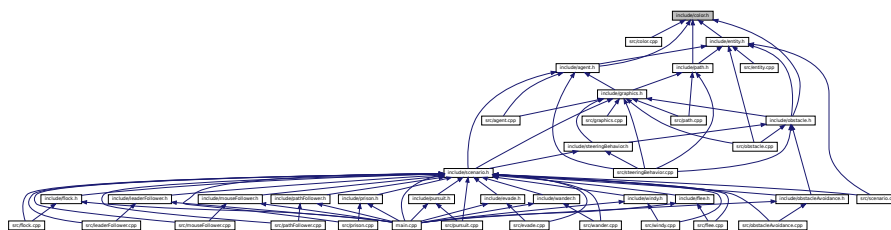
color class used for agent, path, wall etc. color

```
#include <vector>
```

Include dependency graph for color.h:



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



## Classes

- class [color](#)

## Macros

- `#define` [BLACK](#) [color](#)(0,0,0)
- `#define` [BLUE](#) [color](#)(0,0,1)
- `#define` [GREEN](#) [color](#)(0,1,0)
- `#define` [CYAN](#) [color](#)(0,1,1)
- `#define` [RED](#) [color](#)(1,0,0)
- `#define` [YELLOW](#) [color](#)(1,1,0)
- `#define` [MAGENDA](#) [color](#)(1,0,1)
- `#define` [WHITE](#) [color](#)(1,1,1)

### 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 Macro Definition Documentation

#### 7.2.2.1 BLACK

```
#define BLACK color(0,0,0)
```

Definition at line 10 of file color.h.

#### 7.2.2.2 BLUE

```
#define BLUE color(0,0,1)
```

Definition at line 11 of file color.h.

#### 7.2.2.3 CYAN

```
#define CYAN color(0,1,1)
```

Definition at line 13 of file color.h.

#### 7.2.2.4 GREEN

```
#define GREEN color(0,1,0)
```

Definition at line 12 of file color.h.

#### 7.2.2.5 MAGENDA

```
#define MAGENDA color(1,0,1)
```

Definition at line 16 of file color.h.

#### 7.2.2.6 RED

```
#define RED color(1,0,0)
```

Definition at line 14 of file color.h.

#### 7.2.2.7 WHITE

```
#define WHITE color(1,1,1)
```

Definition at line 17 of file color.h.

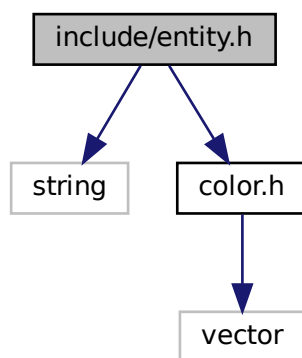
## 7.2.2.8 YELLOW

```
#define YELLOW color(1,1,0)
```

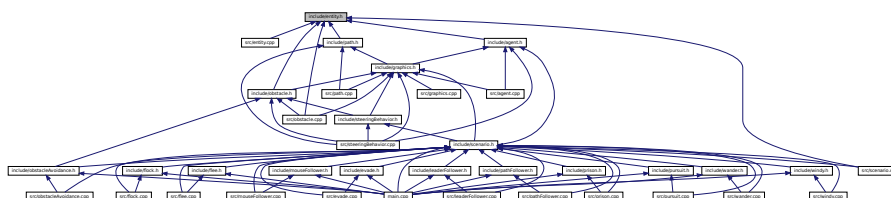
Definition at line 15 of file color.h.

## 7.3 include/entity.h File Reference

```
#include <string>
#include "color.h"
Include dependency graph for entity.h:
```



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



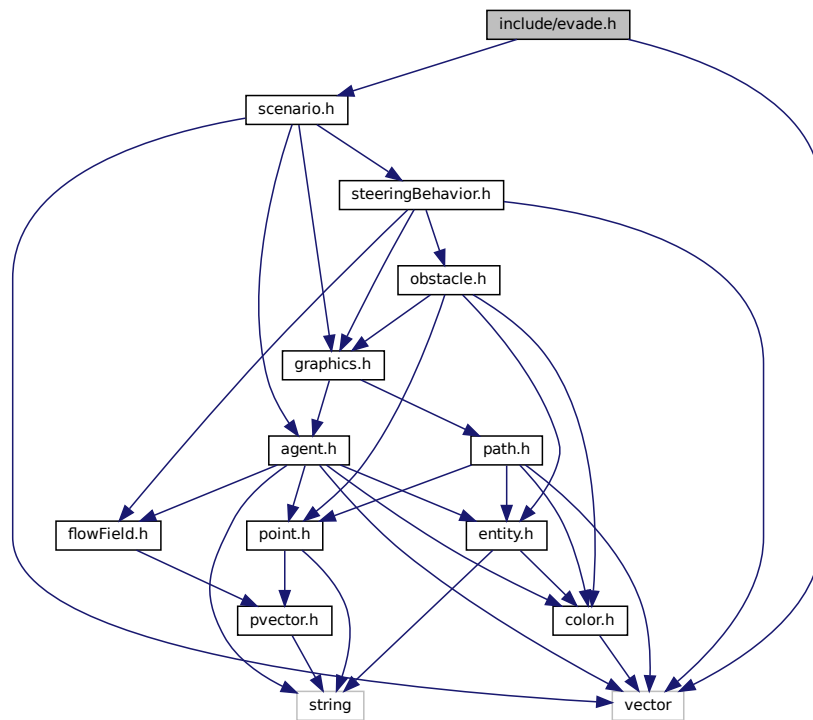
## Classes

- class `entity`

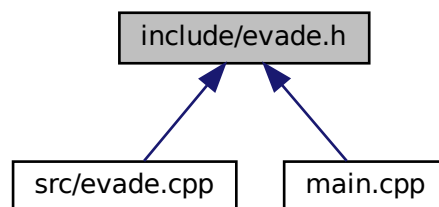
## 7.4 include/evade.h File Reference

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](#)

### 7.4.1 Detailed Description

evade class inherited from scenario class

#### Author

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

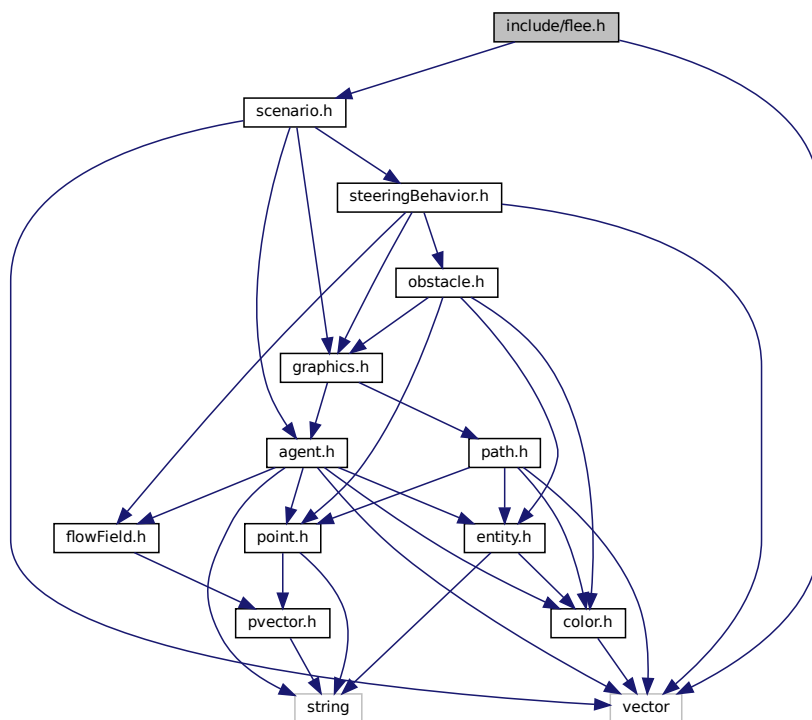
#### Date

15.05.2021

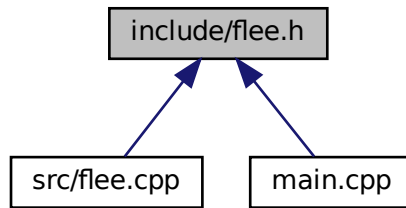
## 7.5 include/flee.h File Reference

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](#)

### 7.5.1 Detailed Description

agents flee from mouse scenario

#### Author

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

#### Date

15.05.2021

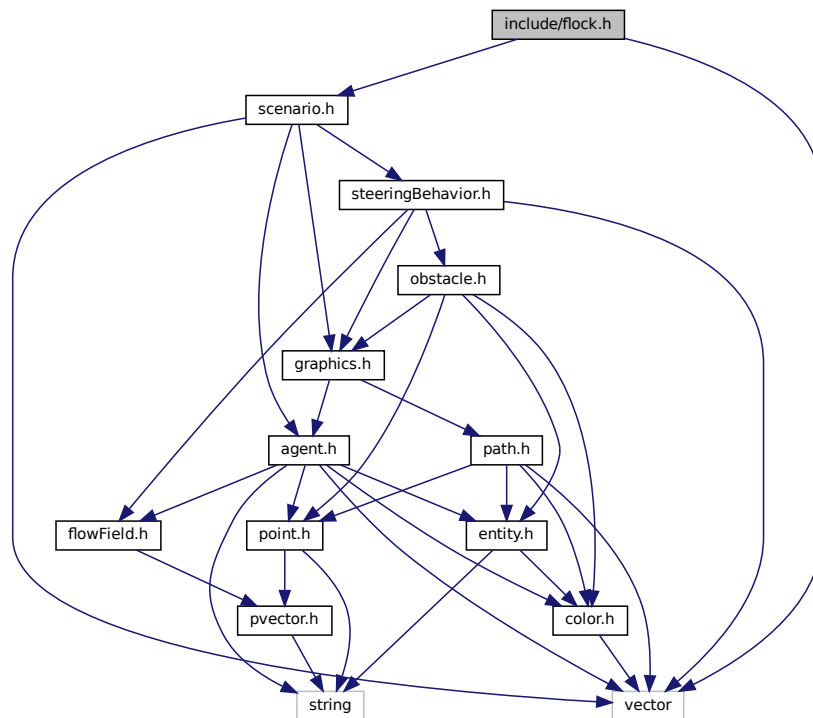
## 7.6 include/flock.h File Reference

flocking agents scenario

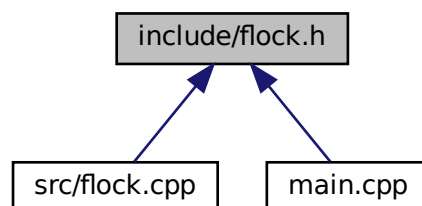
```
#include "scenario.h"
#include <vector>
```



Include dependency graph for flock.h:



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



## Classes

- class [flock](#)

### 7.6.1 Detailed Description

flocking agents scenario

## Author

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

## Date

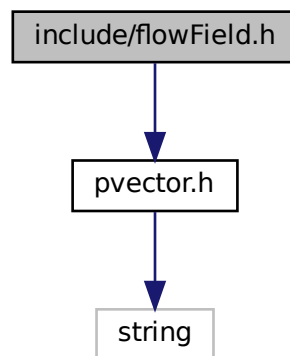
15.05.2021

## 7.7 include/flowField.h File Reference

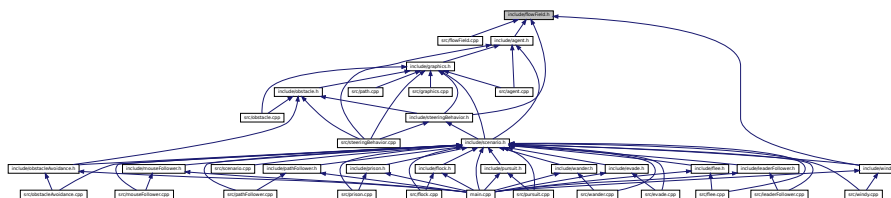
[flowField](#) class, screen can be filled with a force for each pixel

```
#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`

### 7.7.1 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

### 7.7.2 Macro Definition Documentation

#### 7.7.2.1 FIELD\_HEIGHT

```
#define FIELD_HEIGHT 34
```

Definition at line 13 of file `flowField.h`.

#### 7.7.2.2 FIELD\_WIDTH

```
#define FIELD_WIDTH 34
```

Definition at line 12 of file `flowField.h`.

#### 7.7.2.3 GRAVITY

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

Definition at line 16 of file `flowField.h`.

### 7.7.2.4 WIND\_WEST

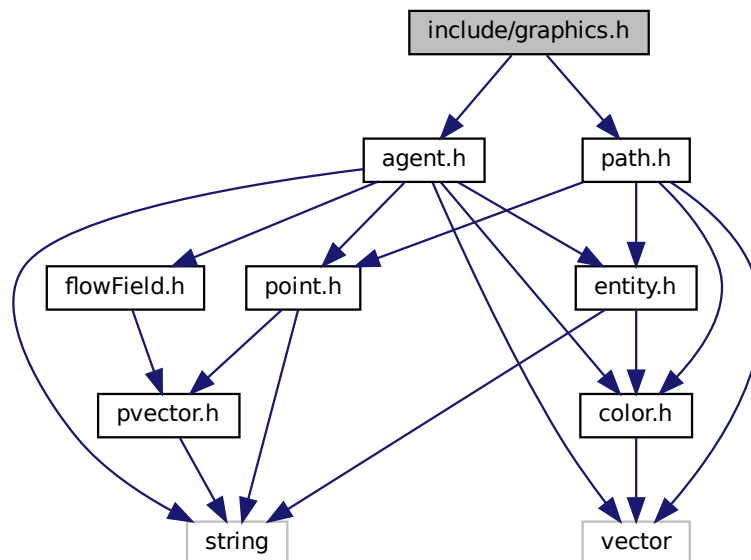
```
#define WIND_WEST 0.1, 0.0
```

Definition at line 15 of file flowField.h.

## 7.8 include/graphics.h File Reference

graphics class, drives OpenGL

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



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



## Classes

- class [graphics](#)

## Macros

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

### 7.8.1 Detailed Description

graphics class, drives openGL

#### Author

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

#### Date

15.05.2021

### 7.8.2 Macro Definition Documentation

#### 7.8.2.1 ESC

```
#define ESC 27
```

Definition at line 16 of file graphics.h.

#### 7.8.2.2 HEIGHT

```
#define HEIGHT 34
```

Definition at line 14 of file graphics.h.

#### 7.8.2.3 PI

```
#define PI 3.14159265
```

Definition at line 17 of file graphics.h.

### 7.8.2.4 WIDTH

```
#define WIDTH 34
```

Definition at line 13 of file graphics.h.

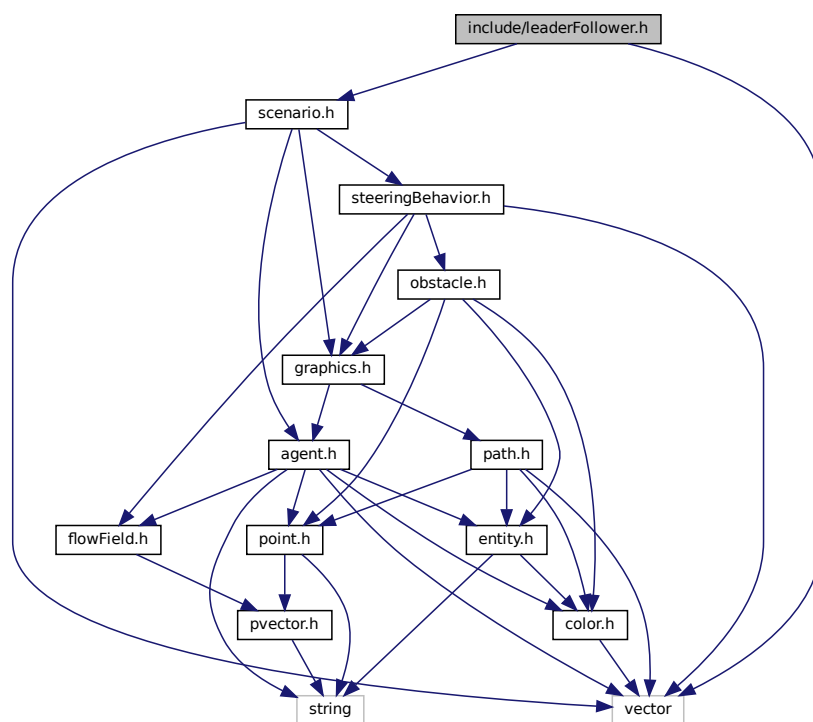
## 7.9 include/leaderFollower.h File Reference

agents follow leader scenario

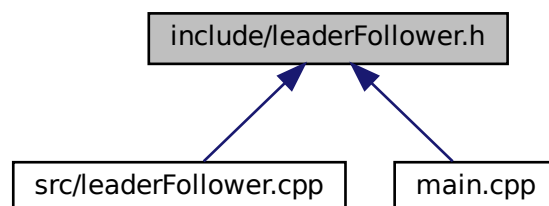
```
#include "scenario.h"
```

```
#include <vector>
```

Include dependency graph for leaderFollower.h:



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



## Classes

- class [leaderFollower](#)

### 7.9.1 Detailed Description

agents follow leader scenario

#### Author

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

#### Date

19.05.2021

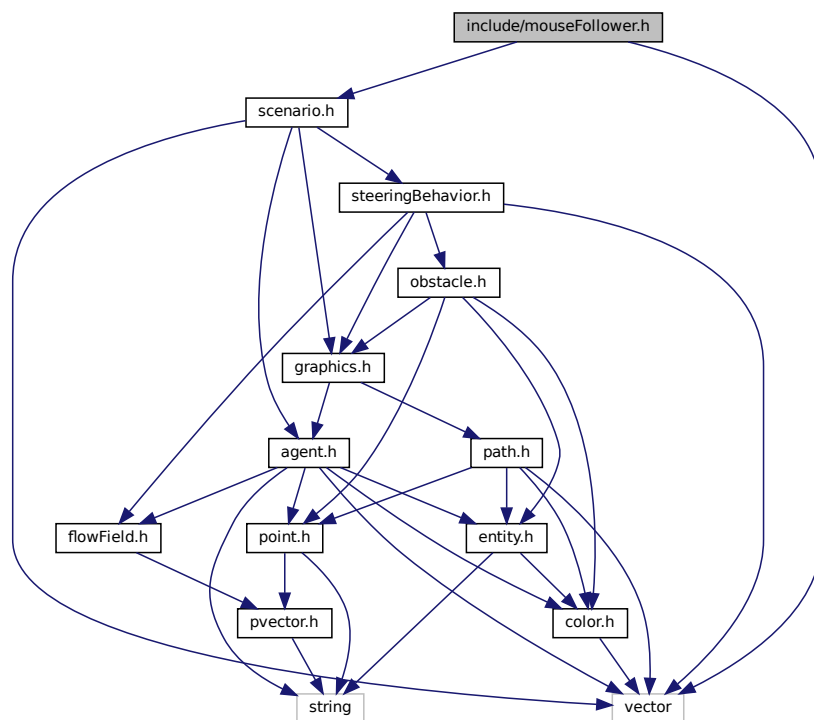
## 7.10 include/mouseFollower.h File Reference

agents follow mouse scenario

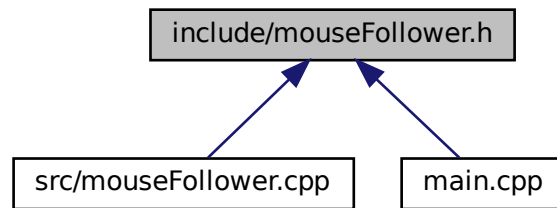
```
#include "scenario.h"
```

```
#include <vector>
```

Include dependency graph for mouseFollower.h:



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



## Classes

- class [mouseFollower](#)

### 7.10.1 Detailed Description

agents follow mouse scenario

#### Author

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

#### Date

15.05.2021

## 7.11 include/obstacle.h File Reference

circular obstacles for agent avoidance behaviors

```
#include "point.h"
#include "graphics.h"
#include "color.h"
```



Include dependency graph for obstacle.h:



- class obstacle

circular obstacles for agent avoidance behaviors

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

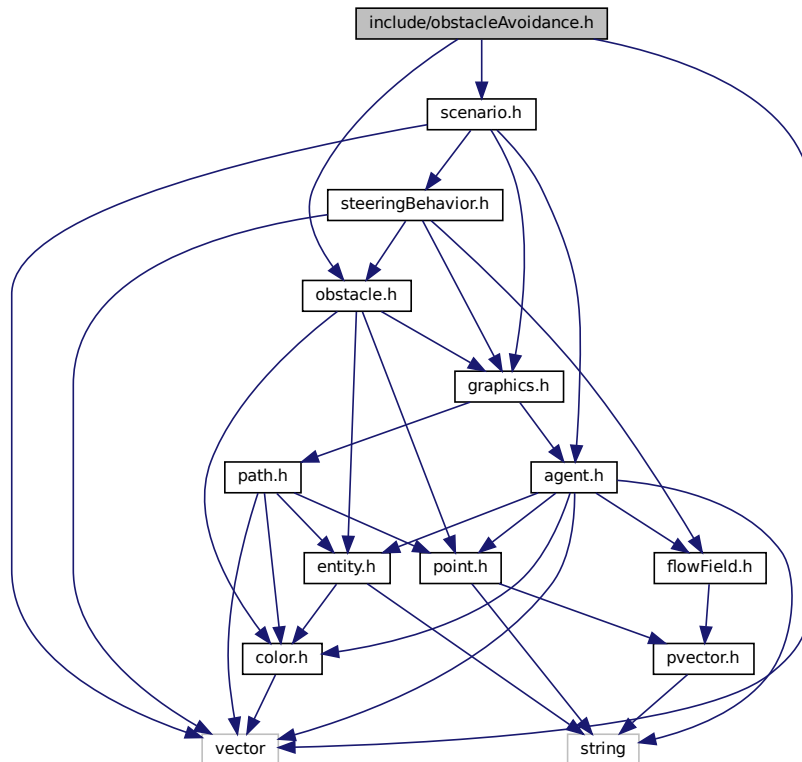
12.05.2021

## 7.12 include/obstacleAvoidance.h File Reference

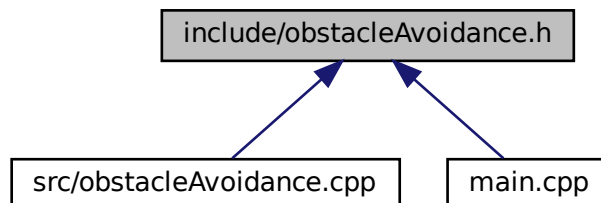
agents avoid from obstacles scenario

```
#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](#)

### 7.12.1 Detailed Description

agents avoid from obstacles scenario

#### Author

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

#### Date

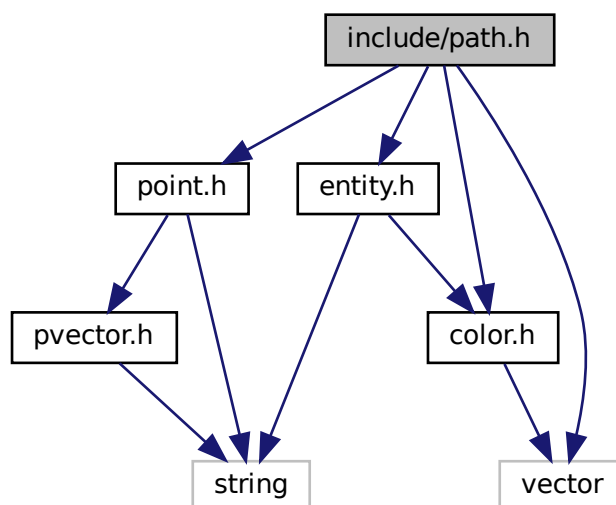
15.05.2021

## 7.13 include/path.h File Reference

path class used for path following steering behaviors.

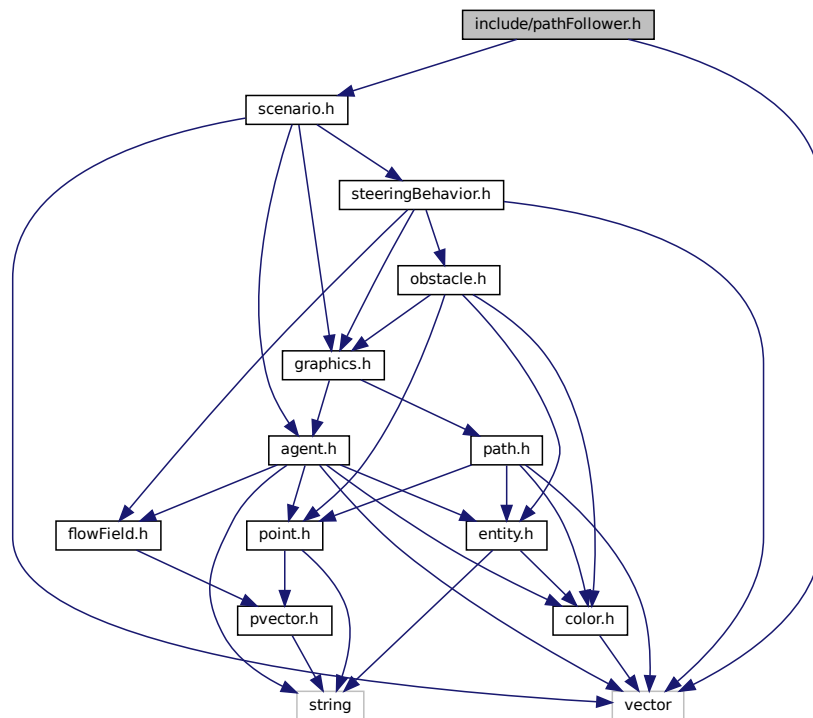
```
#include "point.h"  
#include "entity.h"  
#include "color.h"  
#include <vector>
```

Include dependency graph for path.h:

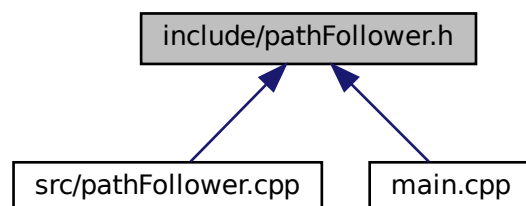




Include dependency graph for pathFollower.h:



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



## Classes

- class [pathFollower](#)

### 7.14.1 Detailed Description

path following scenario

## Author

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

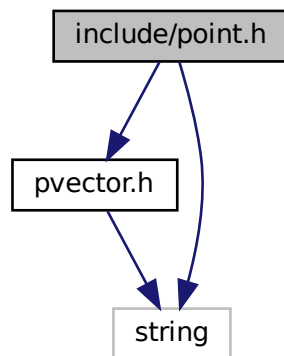
## Date

15.05.2021

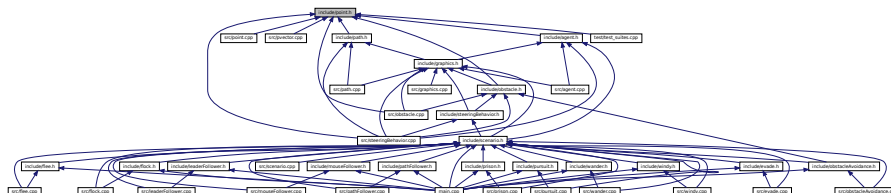
## 7.15 include/point.h File Reference

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](#)

### 7.15.1 Detailed Description

point class used for point operations

#### Author

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

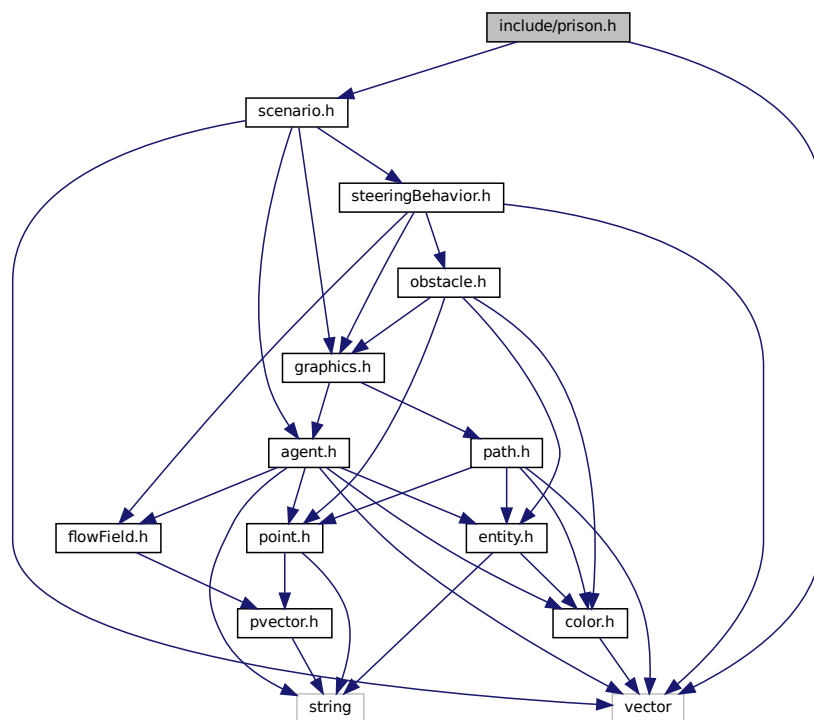
#### Date

15.05.2021

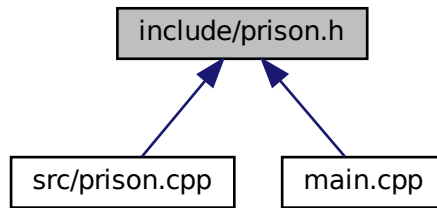
## 7.16 include/prison.h File Reference

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](#)

### 7.16.1 Detailed Description

agents cant escape from field scenario

#### Author

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

#### Date

15.05.2021

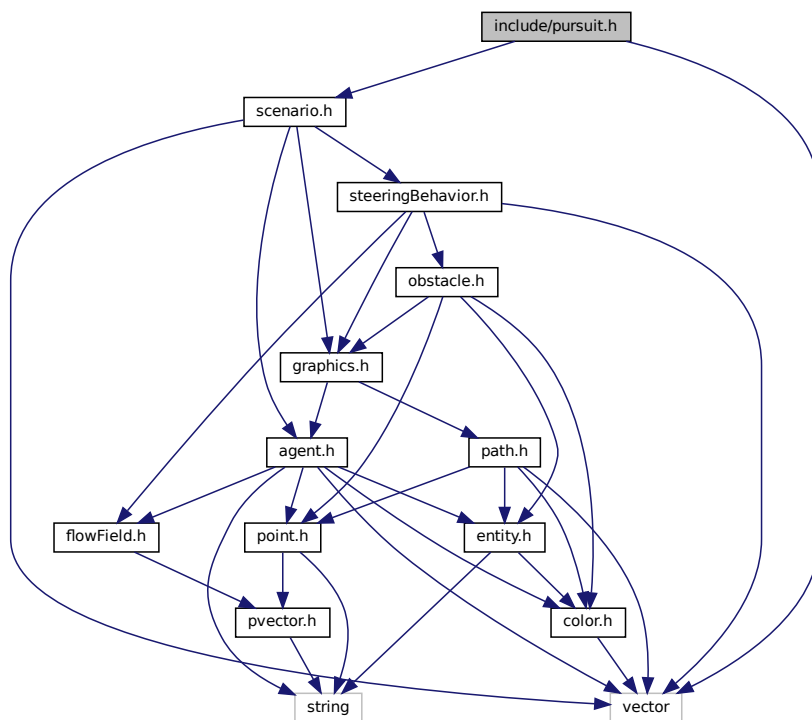
## 7.17 include/pursuit.h File Reference

one agent pursue other one scenario

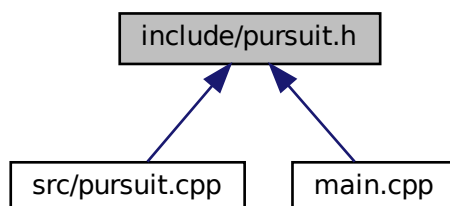
```
#include "scenario.h"  
#include <vector>
```



Include dependency graph for pursuit.h:



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



## Classes

- class [pursuit](#)

### 7.17.1 Detailed Description

one agent pursue other one scenario

**Author**

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

**Date**

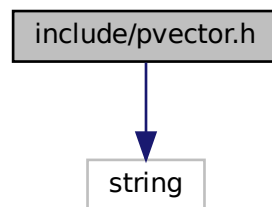
15.05.2021

## 7.18 include/pvector.h File Reference

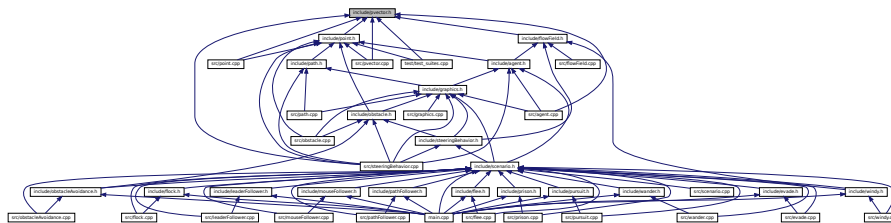
pvector class used for 2D vector operations

```
#include <string>
```

Include dependency graph for pvector.h:



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



### Classes

- class [pvector](#)

### Macros

- `#define` [PI](#) 3.14159265

### 7.18.1 Detailed Description

pvector class used for 2D vector operations

#### Author

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

#### Date

15.05.2021

### 7.18.2 Macro Definition Documentation

#### 7.18.2.1 PI

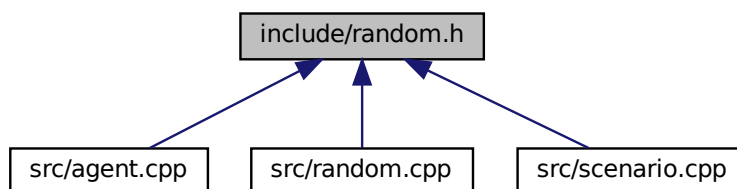
```
#define PI 3.14159265
```

Definition at line 11 of file pvector.h.

## 7.19 include/random.h File Reference

utility class for random operations

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



### Classes

- class [random](#)

### 7.19.1 Detailed Description

utility class for random operations

Author

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

Date

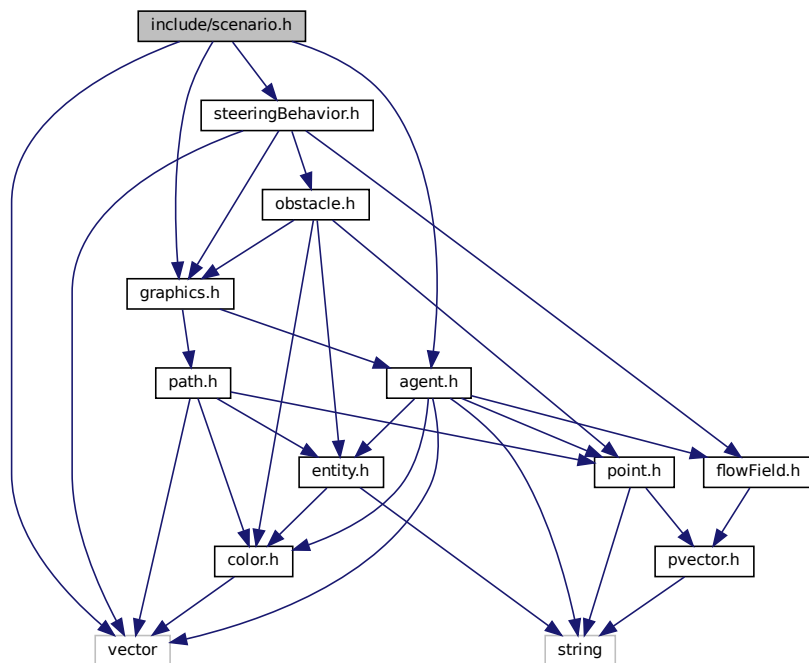
15.05.2021

## 7.20 include/scenario.h File Reference

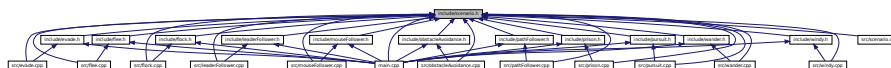
base class for all scenarios

```
#include "agent.h"
#include "graphics.h"
#include "steeringBehavior.h"
#include <vector>
```

Include dependency graph for scenario.h:



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



## Classes

- class [scenario](#)

## Enumerations

- enum [types](#) { [RANDOM](#) =0, [STATIC](#), [TROOP](#) }

### 7.20.1 Detailed Description

base class for all scenarios

#### Author

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

#### Date

15.05.2021

### 7.20.2 Enumeration Type Documentation

#### 7.20.2.1 types

enum [types](#)

##### Enumerator

RANDOM	
STATIC	
TROOP	

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

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

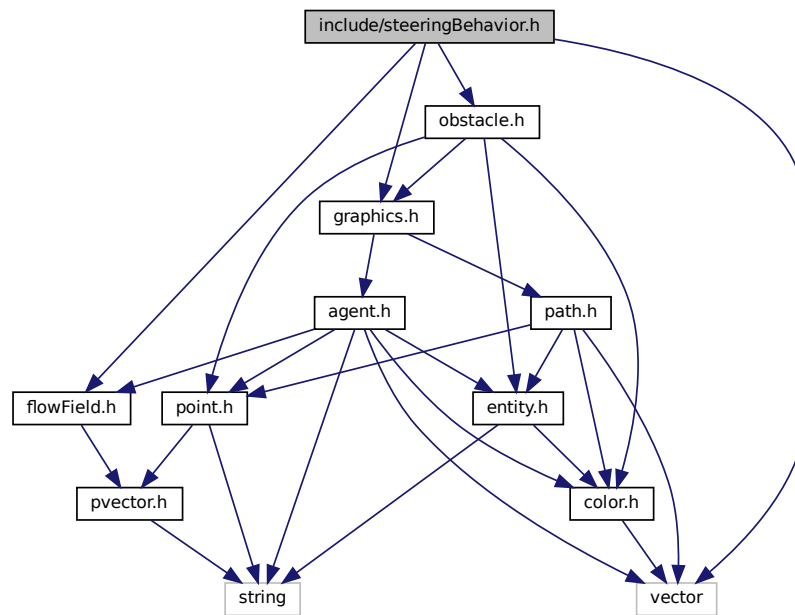
## 7.21 include/steeringBehavior.h File Reference

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
- #define [LEADER\\_FOLLOWER](#) 11

### 7.21.1 Detailed Description

functions for autonomous steering behaviors

#### Author

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

#### Date

15.05.2021

### 7.21.2 Macro Definition Documentation

#### 7.21.2.1 AVOID\_OBSTACLE

```
#define AVOID_OBSTACLE 4
```

Definition at line 21 of file steeringBehavior.h.

#### 7.21.2.2 CIRCLE\_DISTANCE

```
#define CIRCLE_DISTANCE 0.1
```

Definition at line 15 of file steeringBehavior.h.

#### 7.21.2.3 CIRCLE\_RADIUS

```
#define CIRCLE_RADIUS 0.4
```

Definition at line 16 of file steeringBehavior.h.

#### 7.21.2.4 EVADE

```
#define EVADE 10
```

Definition at line 27 of file steeringBehavior.h.

#### 7.21.2.5 FLEE

```
#define FLEE 8
```

Definition at line 25 of file steeringBehavior.h.

#### 7.21.2.6 FLOCK

```
#define FLOCK 6
```

Definition at line 23 of file steeringBehavior.h.

#### 7.21.2.7 FOLLOW\_MOUSE

```
#define FOLLOW_MOUSE 1
```

Definition at line 18 of file steeringBehavior.h.

#### 7.21.2.8 IN\_FLOW\_FIELD

```
#define IN_FLOW_FIELD 3
```

Definition at line 20 of file steeringBehavior.h.

#### 7.21.2.9 LEADER\_FOLLOWER

```
#define LEADER_FOLLOWER 11
```

Definition at line 28 of file steeringBehavior.h.

#### 7.21.2.10 PURSUIT

```
#define PURSUIT 9
```

Definition at line 26 of file steeringBehavior.h.



### 7.21.2.11 STAY\_IN\_FIELD

```
#define STAY_IN_FIELD 2
```

Definition at line 19 of file steeringBehavior.h.

### 7.21.2.12 STAY\_IN\_PATH

```
#define STAY_IN_PATH 5
```

Definition at line 22 of file steeringBehavior.h.

### 7.21.2.13 WANDER

```
#define WANDER 7
```

Definition at line 24 of file steeringBehavior.h.

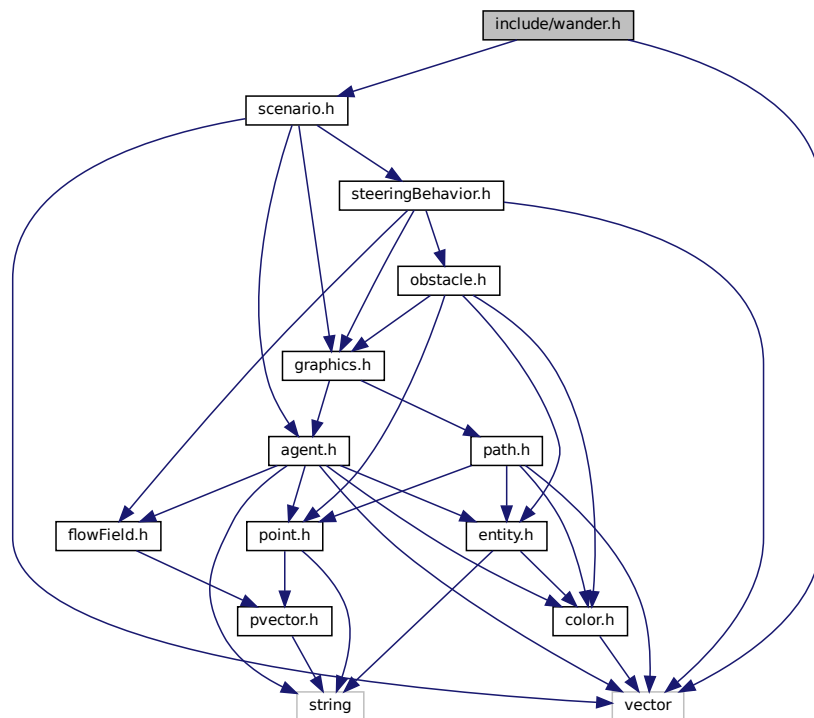
## 7.22 include/wander.h File Reference

random wandering agents scenario

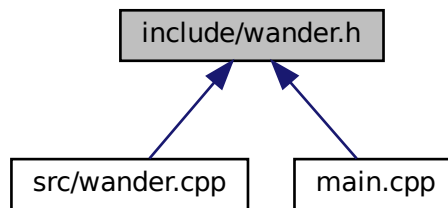
```
#include "scenario.h"
```

```
#include <vector>
```

Include dependency graph for wander.h:



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



## Classes

- class [wander](#)

### 7.22.1 Detailed Description

random wandering agents scenario

#### Author

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

#### Date

15.05.2021

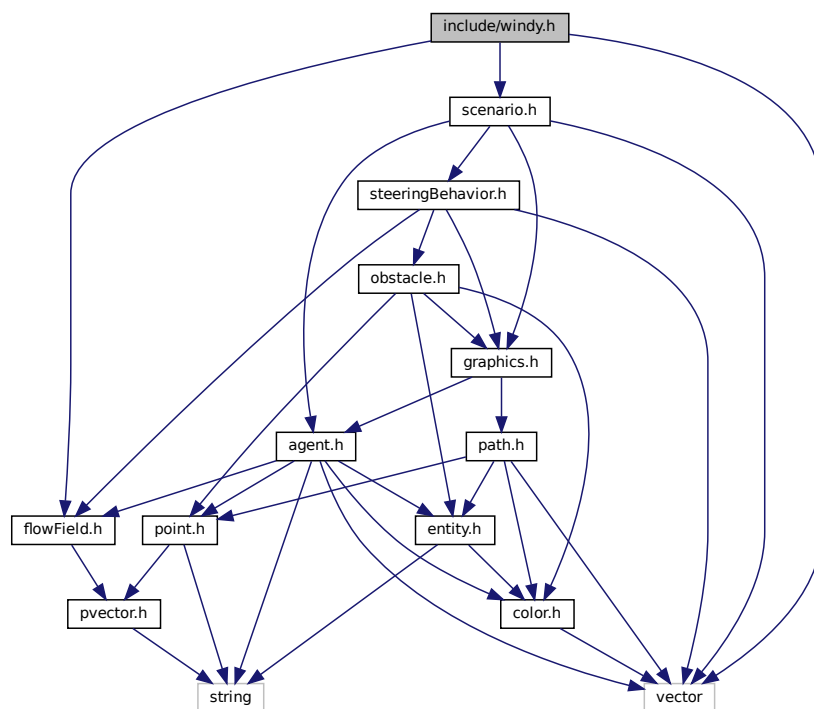
## 7.23 include/windy.h File Reference

windy air scenario

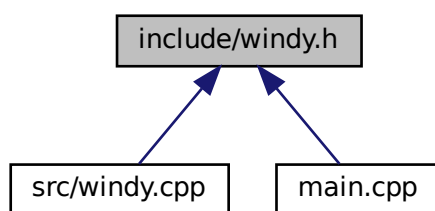
```
#include "scenario.h"  
#include "flowField.h"
```

```
#include <vector>
```

Include dependency graph for windy.h:



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



## Classes

- class [windy](#)

### 7.23.1 Detailed Description

windy air scenario



## 7.24.1 Detailed Description

client code

### Author

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

### Date

15.05.2021

## 7.24.2 Function Documentation

### 7.24.2.1 main()

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

main routine

Definition at line 50 of file main.cpp.

```
50     {
51         menu();
52
53         scenario* sc;
54
55         if(mode == FOLLOW_MOUSE) {
56             *sc = mouseFollower();
57         }
58         else if(mode == STAY_IN_FIELD) {
59             *sc = prison();
60         }
61         else if(mode == IN_FLOW_FIELD) {
62             *sc = windy();
63         }
64         else if(mode == WANDER) {
65             *sc = wander();
66         }
67         else if(mode == PURSUIT) {
68             *sc = pursuit();
69         }
70         else if(mode == FLEE) {
71             *sc = flee();
72         }
73         else if(mode == EVADE) {
74             *sc = evade();
75         }
76         else if(mode == FLOCK) {
77             *sc = flock();
78         }
79         else if(mode == STAY_IN_PATH) {
80             *sc = pathFollower();
81         }
82         else if(mode == AVOID_OBSTACLE) {
83             *sc = obstacleAvoidance();
84         }
85         else if(mode == LEADER_FOLLOWER) {
86             *sc = leaderFollower();
87         }
88
89         sc->initGL(&argc, argv);
90
91         return 0;
92     }
```

### 7.24.2.2 menu()

```
void menu ( )
```

displays menu

Definition at line 32 of file main.cpp.

```
32     {
33         cout << "Follow Mouse       : 1" << endl;
34         cout << "Stay in Field       : 2" << endl;
35         cout << "In Flow Field       : 3" << endl;
36         cout << "Avoid Obstacles    : 4" << endl;
37         cout << "Stay in Path       : 5" << endl;
38         cout << "FLOCK             : 6" << endl;
39         cout << "WANDER            : 7" << endl;
40         cout << "FLEE              : 8" << endl;
41         cout << "PURSUIT           : 9" << endl;
42         cout << "EVADE             : 10" << endl;
43         cout << "Follow Leader     : 11" << endl;
44         cin >> mode;
45     }
```

## 7.24.3 Variable Documentation

### 7.24.3.1 mode

```
int mode
```

specifies user selected scenario

Definition at line 27 of file main.cpp.

## 7.25 README.md File Reference

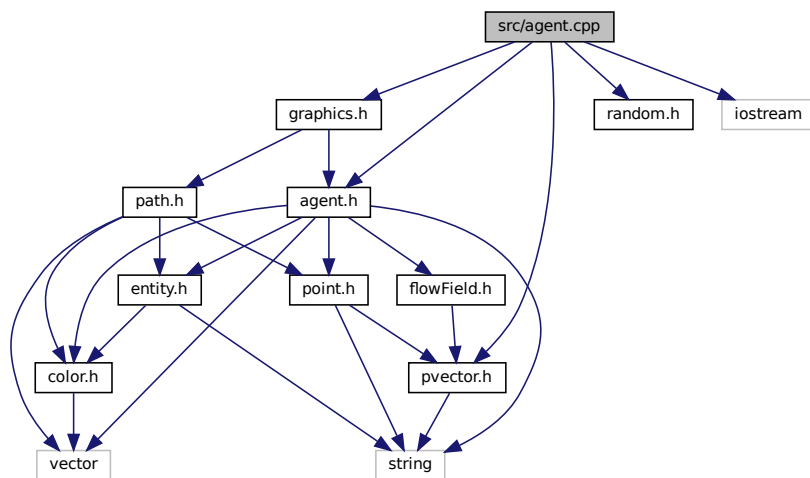
## 7.26 src/agent.cpp File Reference

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:



### 7.26.1 Detailed Description

implementation of the agent class

Author

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

Date

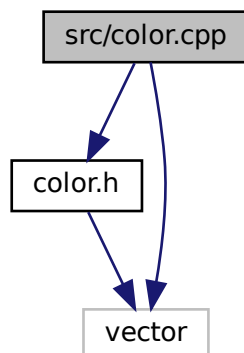
14.05.2021

## 7.27 src/color.cpp File Reference

color class implementation

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

Include dependency graph for color.cpp:



### 7.27.1 Detailed Description

color class implementation

Author

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

Date

13.05.2021

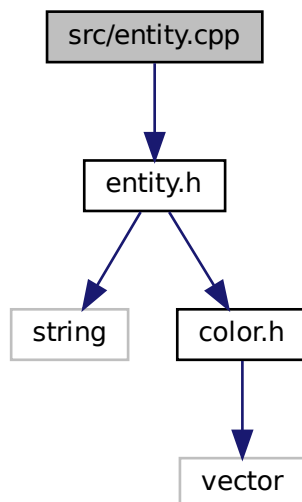
## 7.28 src/entity.cpp File Reference

entity class implementation



```
#include "entity.h"
```

Include dependency graph for entity.cpp:



### 7.28.1 Detailed Description

entity class implementation

Author

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

Date

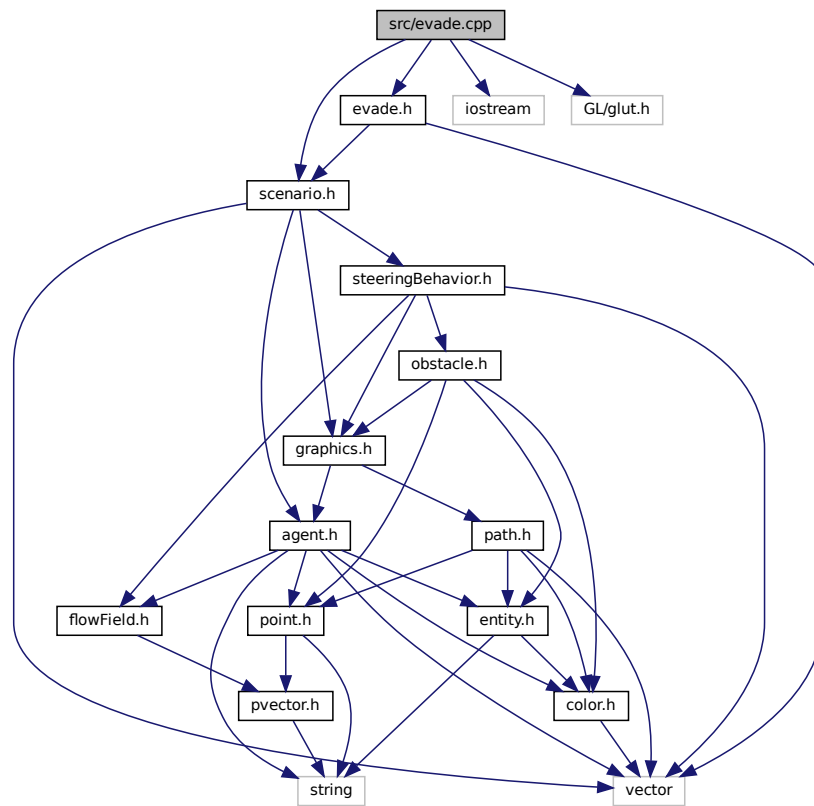
18.05.2021

## 7.29 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:
```



### 7.29.1 Detailed Description

evade class implementation

Author

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

Date

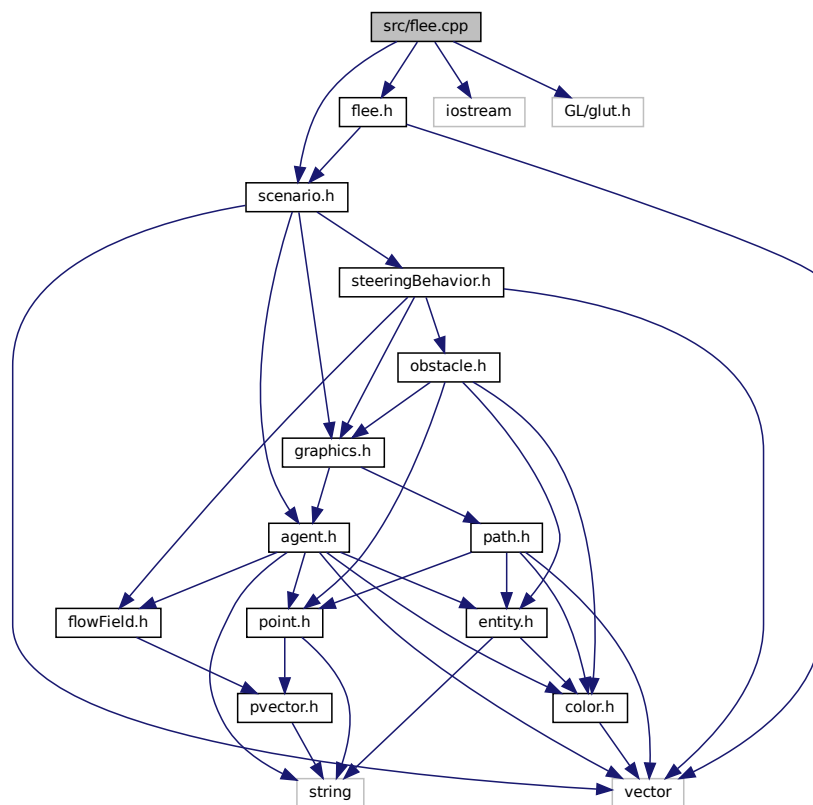
15.05.2021

## 7.30 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:
```



### 7.30.1 Detailed Description

flee class implementation

Author

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

Date

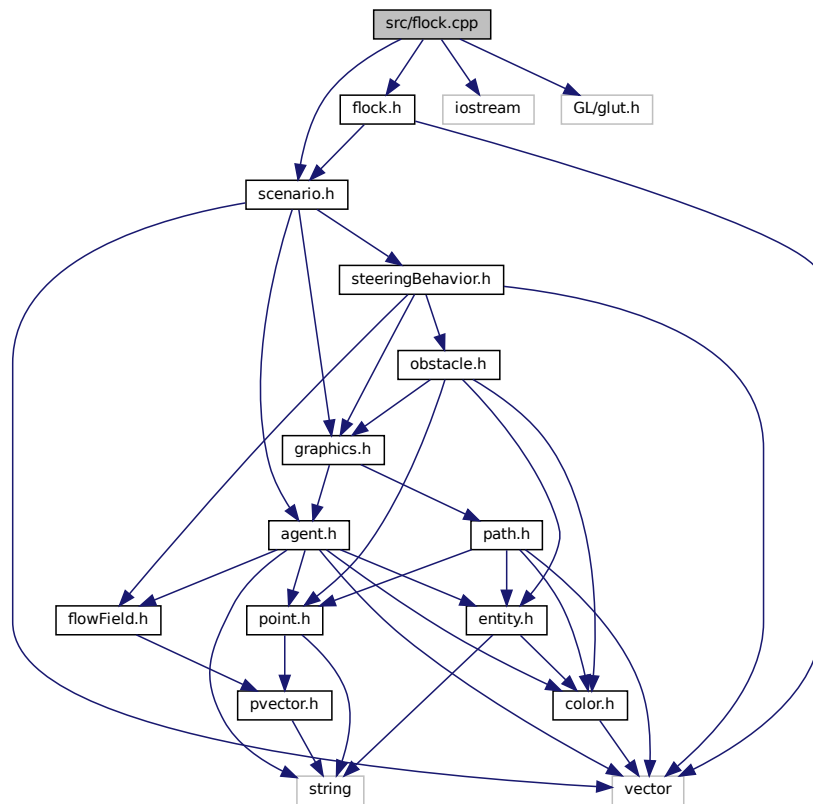
15.05.2021

## 7.31 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:
```



### 7.31.1 Detailed Description

flock class implementation

Author

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

Date

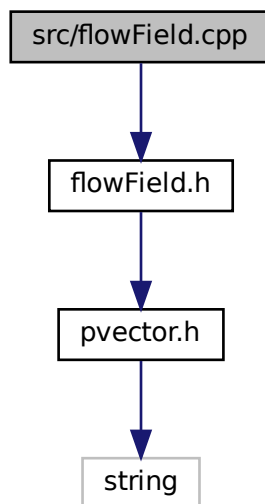
15.05.2021

## 7.32 src/flowField.cpp File Reference

[flowField](#) class implementation

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

Include dependency graph for flowField.cpp:



### 7.32.1 Detailed Description

`flowField` class implementation

#### Author

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

#### Date

13.05.2021

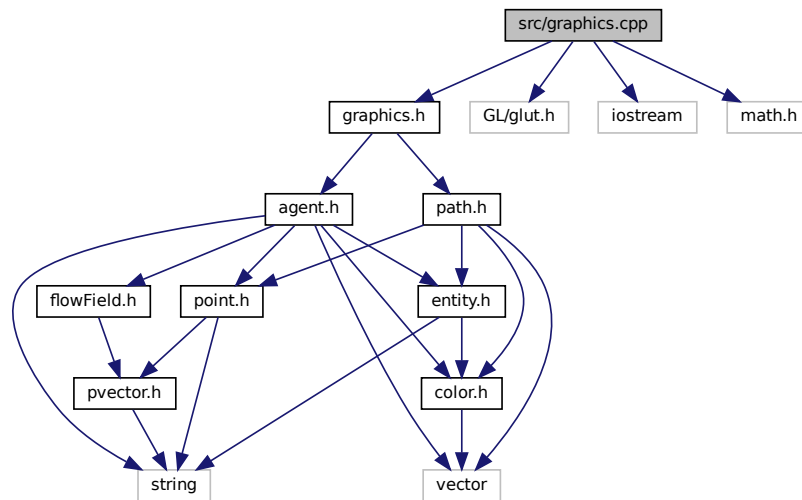
## 7.33 src/graphics.cpp File Reference

graphics class implementation

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

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

Include dependency graph for graphics.cpp:



### 7.33.1 Detailed Description

graphics class implementation

Author

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

Date

15.05.2021

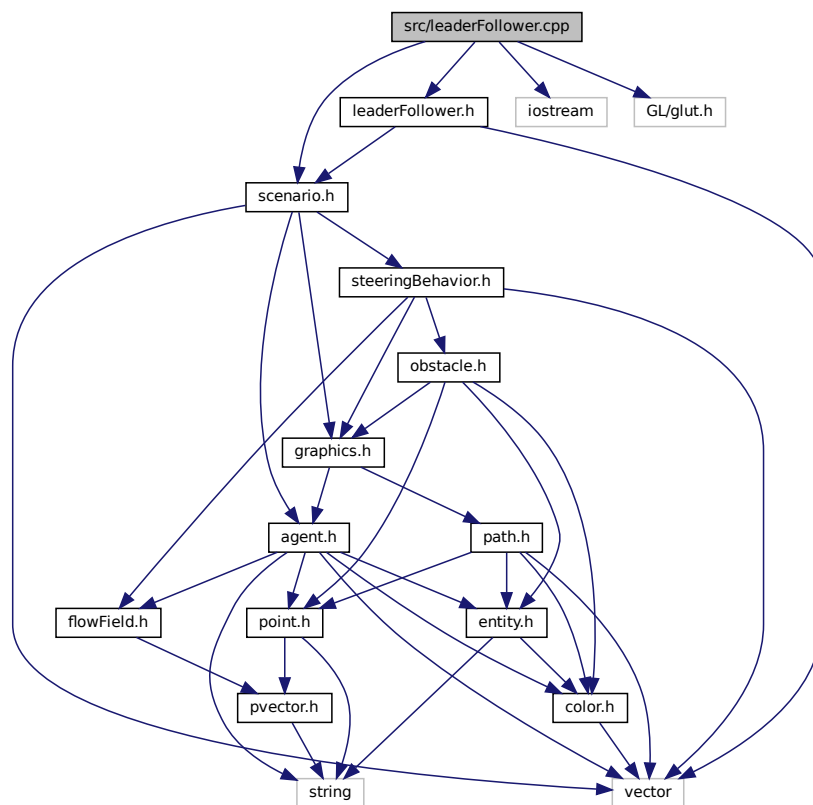
## 7.34 src/leaderFollower.cpp File Reference

`leaderFollower` class implementation

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

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

Include dependency graph for leaderFollower.cpp:



### 7.34.1 Detailed Description

[leaderFollower](#) class implementation

Author

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

Date

19.05.2021

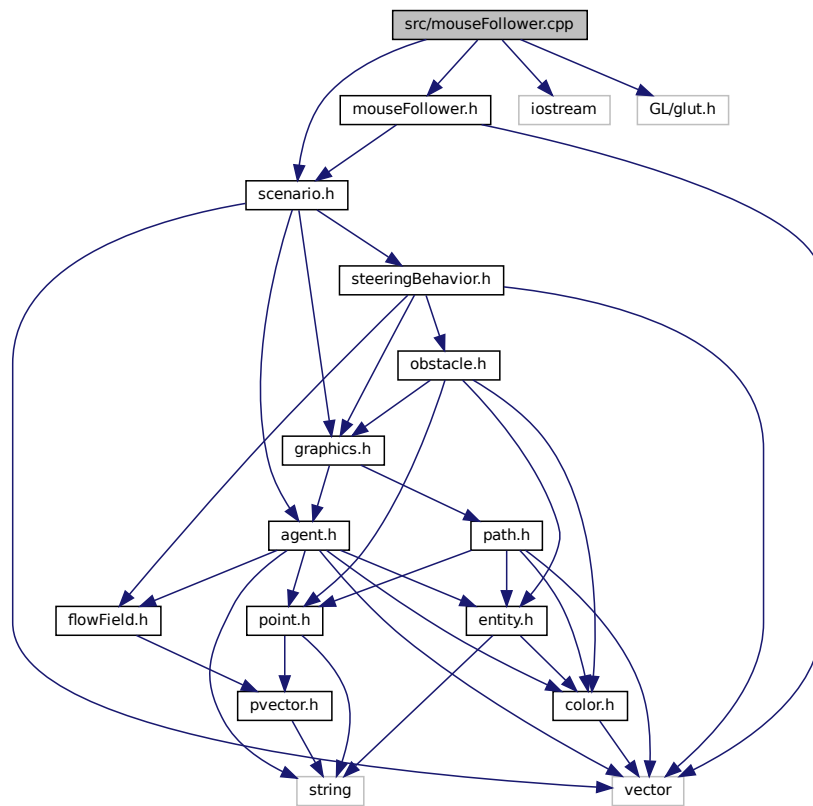
## 7.35 src/mouseFollower.cpp File Reference

[mouseFollower](#) class implementation

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

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

Include dependency graph for mouseFollower.cpp:



### 7.35.1 Detailed Description

[mouseFollower](#) class implementation

Author

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

Date

15.05.2021

## 7.36 src/obstacle.cpp File Reference

obstacle class implementation

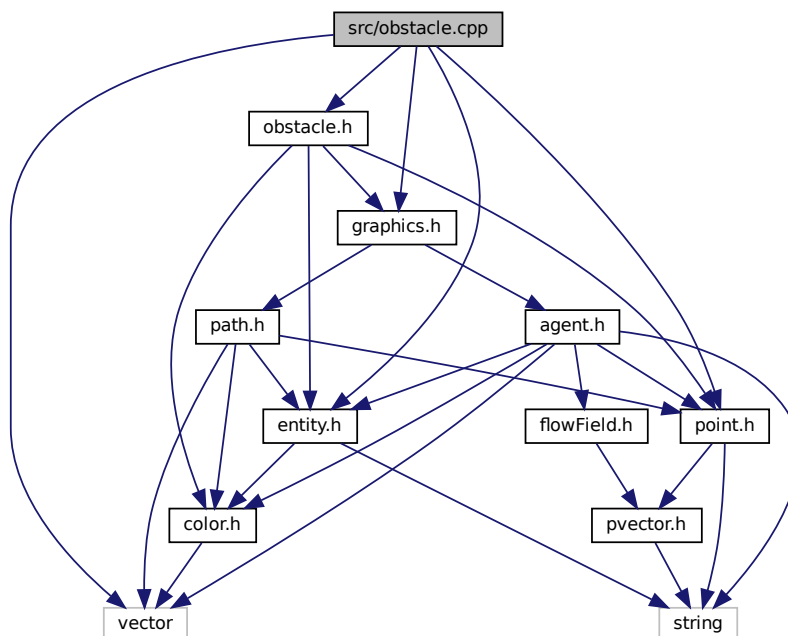
```
#include "obstacle.h"
#include "graphics.h"
#include "point.h"
```



```
#include "entity.h"
```

```
#include <vector>
```

Include dependency graph for obstacle.cpp:



### 7.36.1 Detailed Description

obstacle class implementation

Author

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

Date

12.05.2021

## 7.37 src/obstacleAvoidance.cpp File Reference

[obstacleAvoidance](#) class implementation

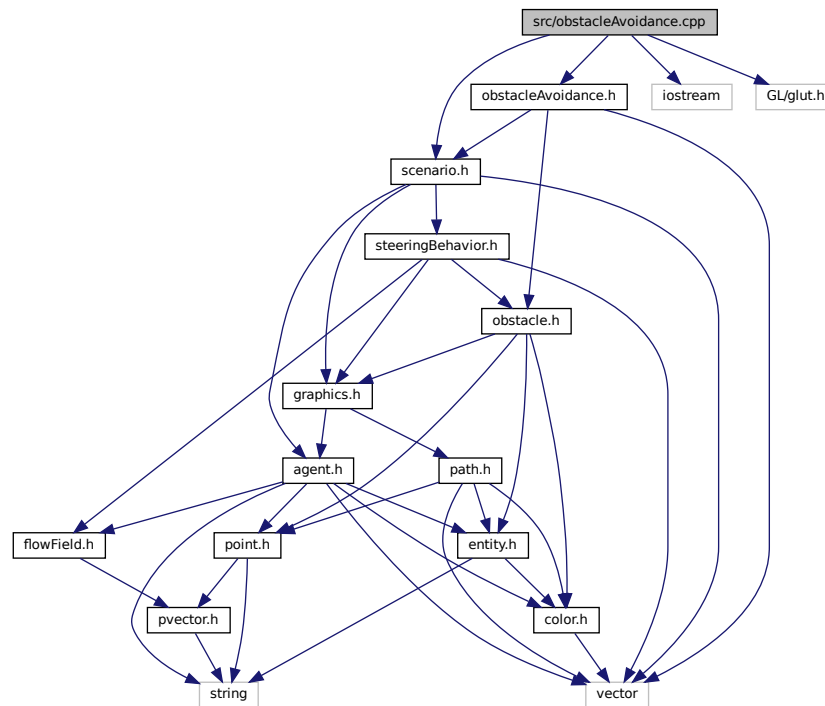
```
#include "scenario.h"
```

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

```
#include <iostream>
```

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

Include dependency graph for obstacleAvoidance.cpp:



### 7.37.1 Detailed Description

[obstacleAvoidance](#) class implementation

#### Author

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

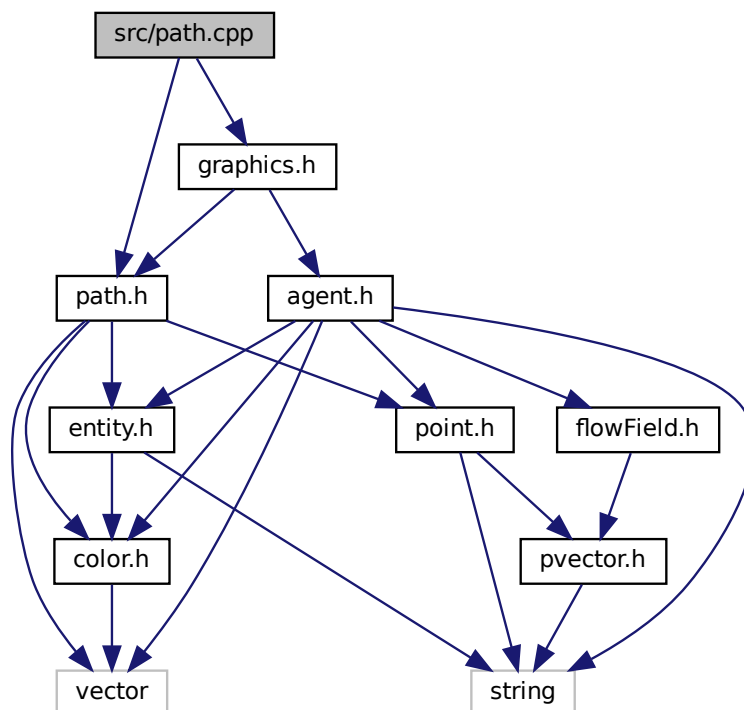
#### Date

15.05.2021

## 7.38 src/path.cpp File Reference

path class implementation

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



### 7.38.1 Detailed Description

path class implementation

Author

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

Date

12.05.2021

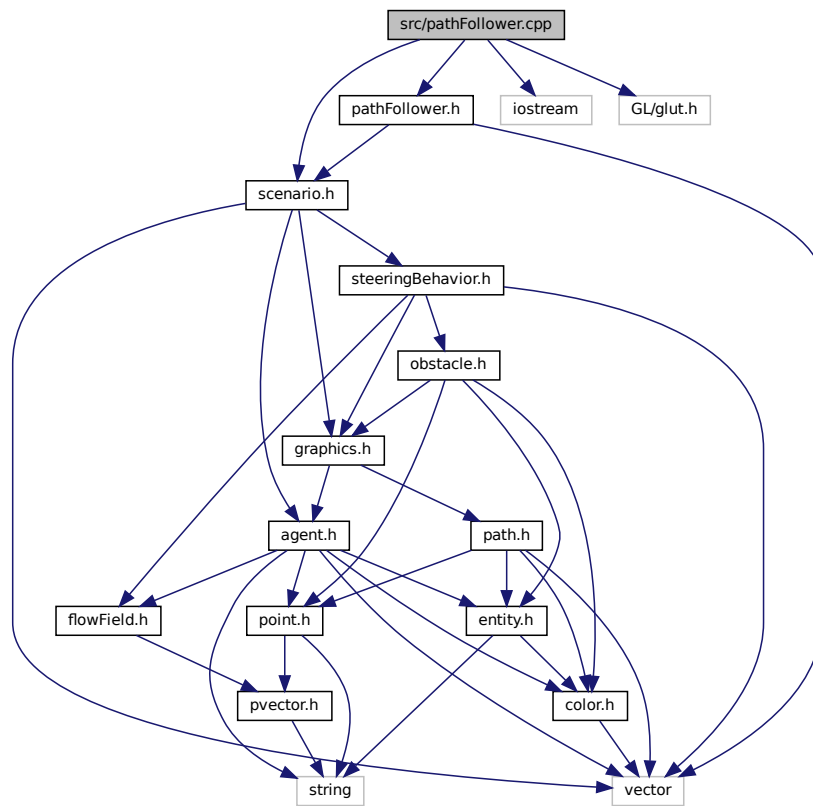
## 7.39 src/pathFollower.cpp File Reference

`pathFollower` class implementation

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

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

Include dependency graph for pathFollower.cpp:



### 7.39.1 Detailed Description

[pathFollower](#) class implementation

Author

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

Date

15.05.2021

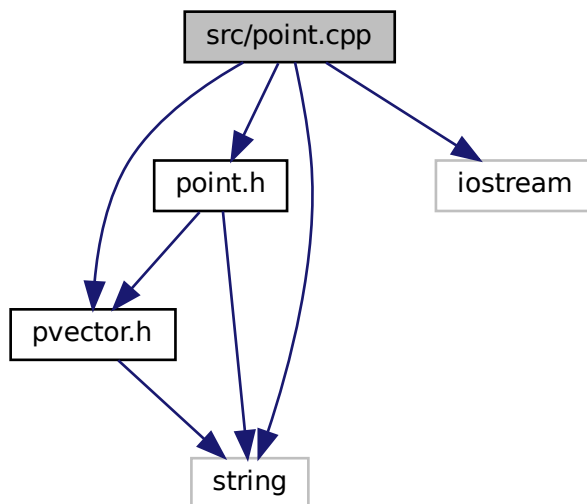
## 7.40 src/point.cpp File Reference

point class implementation file

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

```
#include <iostream>
```

Include dependency graph for point.cpp:



### 7.40.1 Detailed Description

point class implementation file

#### Author

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

#### Date

15.05.2021

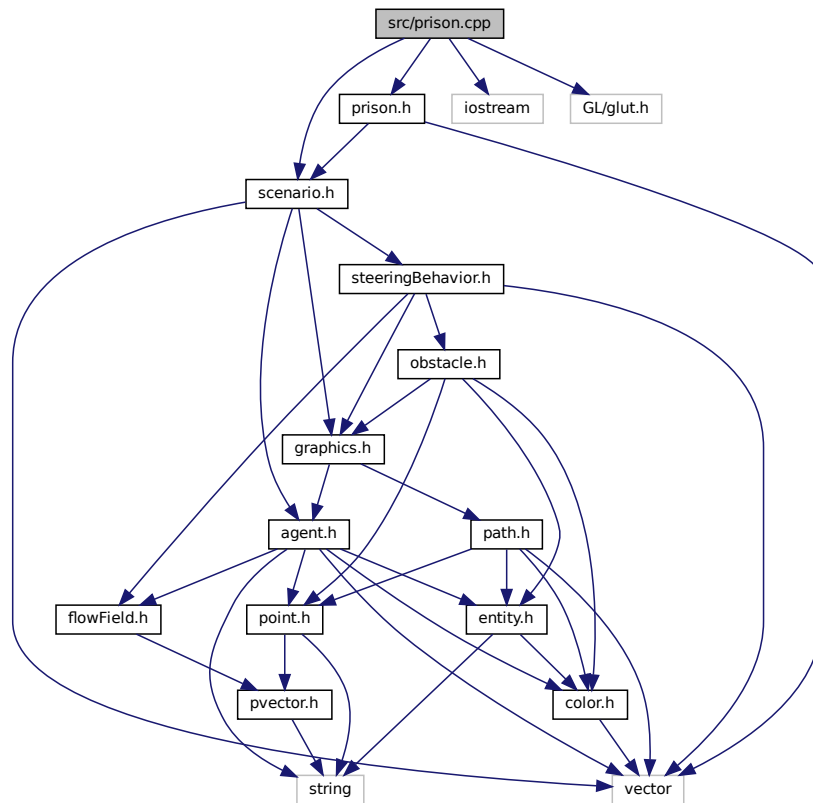
## 7.41 src/prison.cpp File Reference

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

### 7.41.1 Detailed Description

prison 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 DISTANCE

```
#define DISTANCE 2
```

Definition at line 14 of file prison.cpp.

### 7.41.2.2 WALL

```
#define WALL 30
```

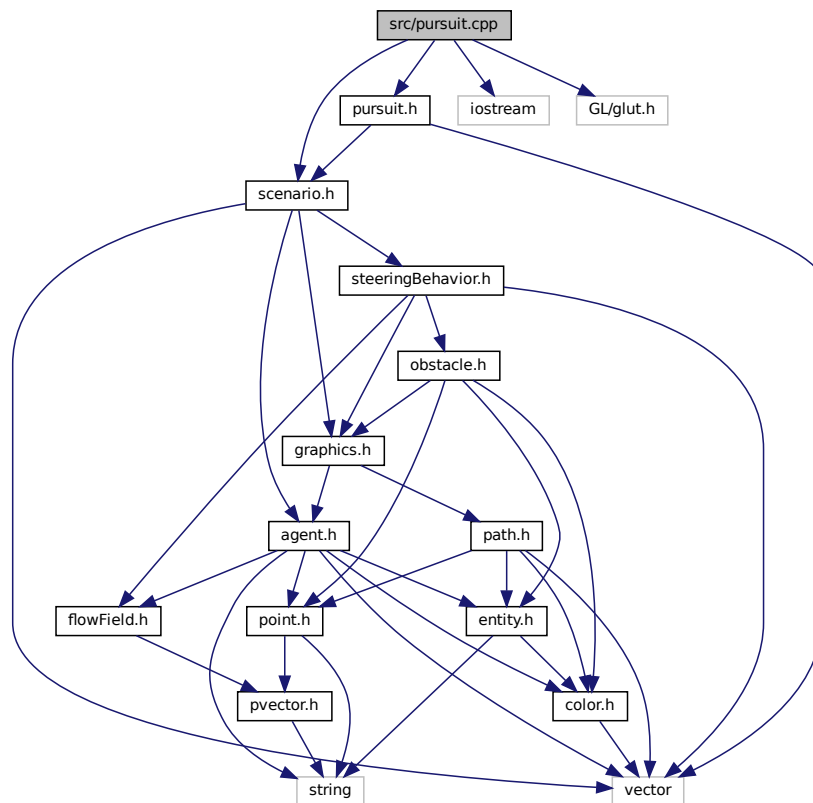
Definition at line 13 of file prison.cpp.

## 7.42 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:



### 7.42.1 Detailed Description

prison class implementation

Author

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

Date

15.05.2021

## 7.43 src/pvector.cpp File Reference

pvector class implementation

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

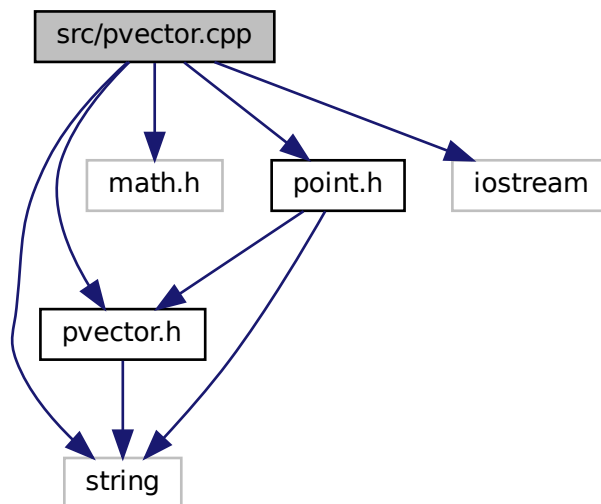
```
#include "math.h"
```

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

```
#include <iostream>
```

```
#include <string>
```

Include dependency graph for pvector.cpp:



### 7.43.1 Detailed Description

pvector class implementation

Author

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

Date

15.05.2021

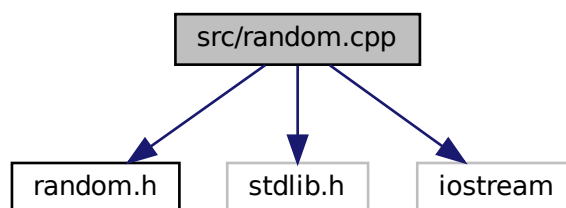


## 7.44 src/random.cpp File Reference

utility class for random operations

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

Include dependency graph for random.cpp:



### 7.44.1 Detailed Description

utility class for random operations

Author

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

Date

15.05.2021

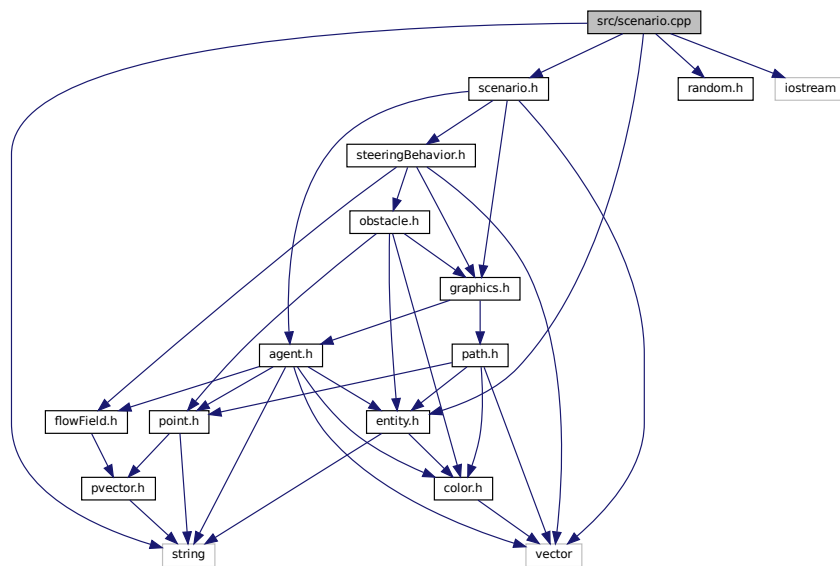
## 7.45 src/scenario.cpp File Reference

scenario base class implementation

```
#include "scenario.h"  
#include "random.h"  
#include "entity.h"  
#include <iostream>
```

```
#include <string>
```

Include dependency graph for scenario.cpp:



## Macros

- `#define MAX_NUMBER_OF_AGENTS 50`

### 7.45.1 Detailed Description

scenario base class implementation

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 MAX\_NUMBER\_OF\_AGENTS

```
#define MAX_NUMBER_OF_AGENTS 50
```

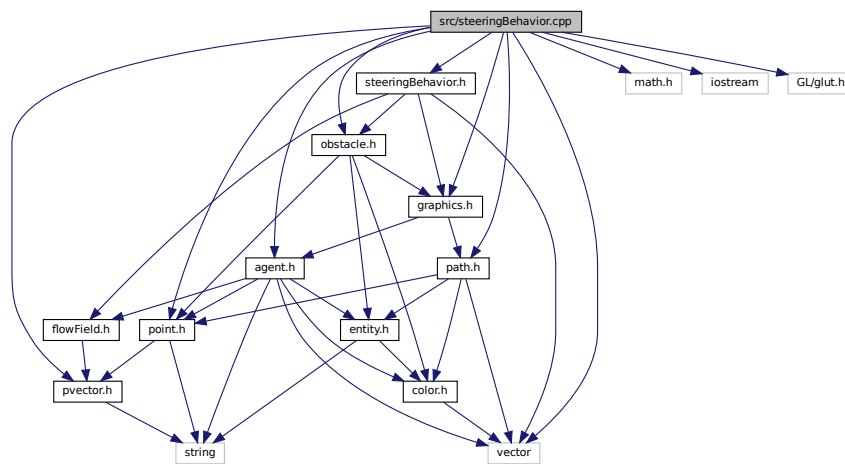
Definition at line 14 of file scenario.cpp.

## 7.46 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:



### 7.46.1 Detailed Description

implementation of autonomous steering behaviors

Author

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

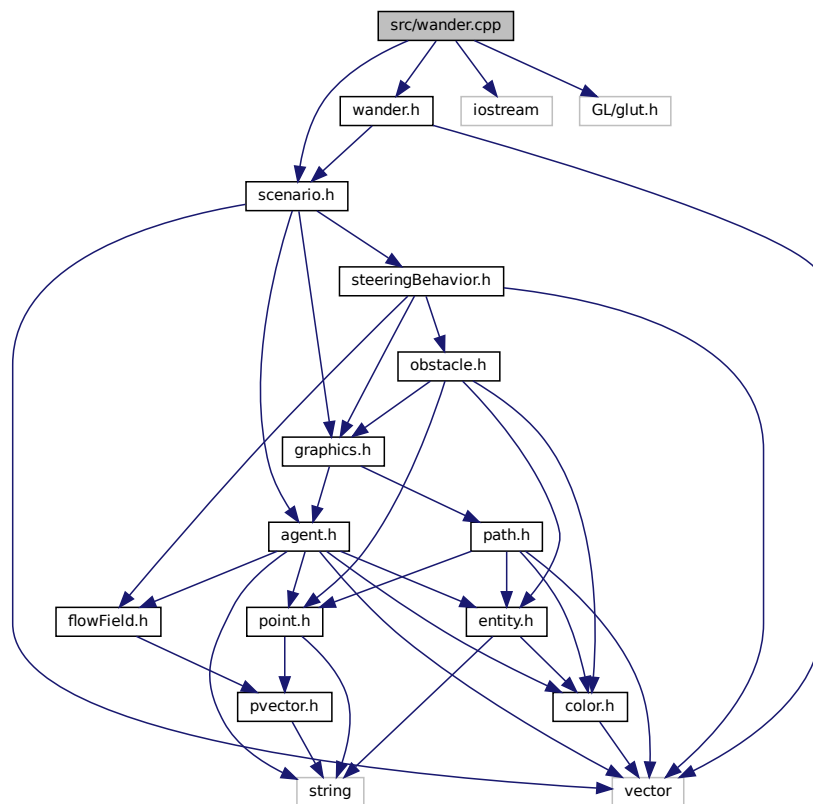
Date

15.05.2021

## 7.47 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:
```



### 7.47.1 Detailed Description

wander class implementation

Author

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

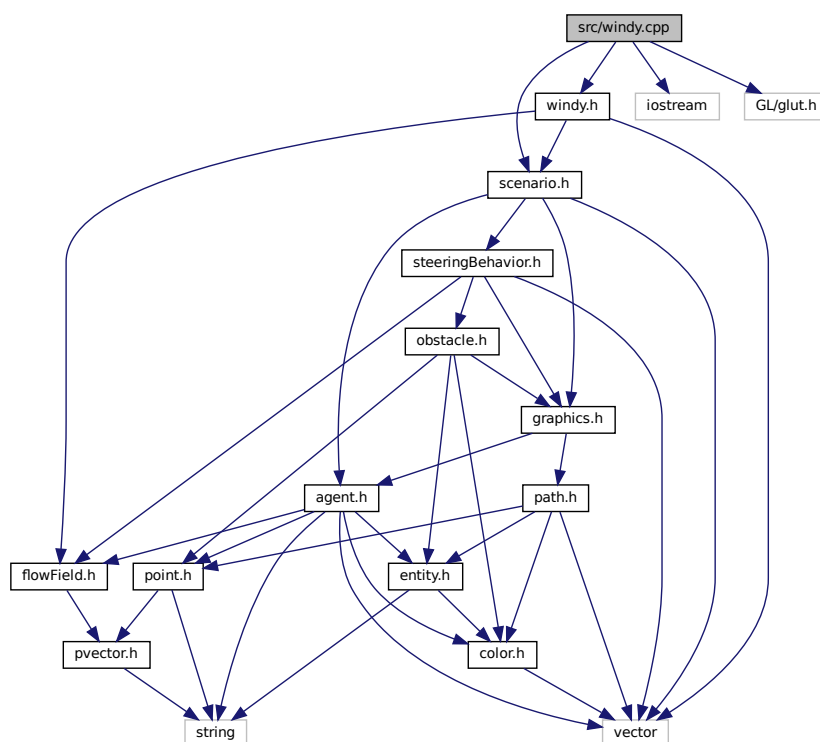
Date

15.05.2021

## 7.48 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:
```



### 7.48.1 Detailed Description

windy class implementation

Author

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

Date

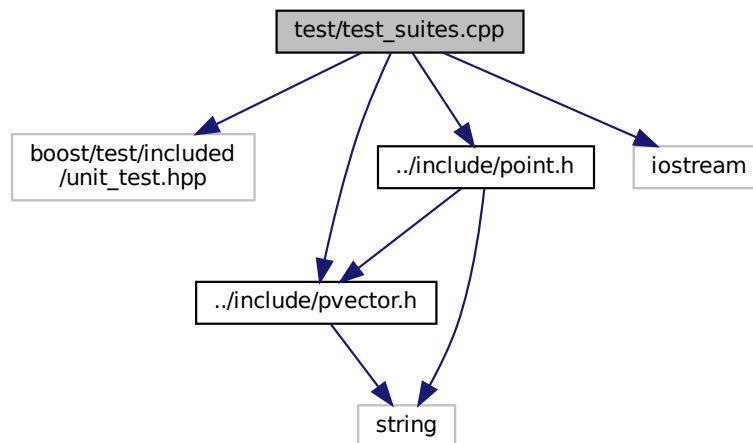
15.05.2021

## 7.49 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*

## 7.49.1 Detailed Description

unit test suites

Author

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

Date

15.05.2021

## 7.49.2 Macro Definition Documentation

### 7.49.2.1 BOOST\_TEST\_MODULE

```
#define BOOST_TEST_MODULE test_suites
```

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

## 7.49.3 Function Documentation

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

```
BOOST_AUTO_TEST_CASE (
    s1t1 )
```

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 }
```

**7.49.3.2 BOOST\_AUTO\_TEST\_CASE()** [2/12]

```
BOOST_AUTO_TEST_CASE (
    s1t2 )
```

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 }
```

**7.49.3.3 BOOST\_AUTO\_TEST\_CASE()** [3/12]

```
BOOST_AUTO_TEST_CASE (
    s1t3 )
```

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 }
```

**7.49.3.4 BOOST\_AUTO\_TEST\_CASE()** [4/12]

```
BOOST_AUTO_TEST_CASE (
    s1t4 )
```

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 }
```

**7.49.3.5 BOOST\_AUTO\_TEST\_CASE()** [5/12]

```
BOOST_AUTO_TEST_CASE (
    s1t5 )
```

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 }
```



### 7.49.3.6 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 }
```

### 7.49.3.7 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 }
```

### 7.49.3.8 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 }
```

**7.49.3.9 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 }
```

**7.49.3.10 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 }
```

**7.49.3.11 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 }
```

**7.49.3.12 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 }
```

# Index

- ~agent
  - agent, 16
- acceleration
  - agent, 19
- add
  - pvector, 66
- addPoint
  - path, 52
- agent, 13
  - ~agent, 16
  - acceleration, 19
  - agent, 14
  - arrive, 19
  - desiredVelocity, 19
  - draw, 16
  - force, 19
  - getMass, 17
  - getName, 17
  - id, 20
  - maxForce, 20
  - maxSpeed, 20
  - position, 20
  - r, 20
  - setFeatures, 17
  - setMass, 18
  - setName, 18
  - steering, 21
  - targetPoint, 21
  - updatePosition, 18
  - velocity, 21
- agents
  - scenario, 76
- align
  - steeringBehavior, 79
- angleBetween
  - pvector, 66
- arrive
  - agent, 19
- avoid
  - steeringBehavior, 79
- AVOID\_OBSTACLE
  - steeringBehavior.h, 121
- B
  - color, 23
- behavior
  - scenario, 77
- BLACK
  - color.h, 93
- BLUE
  - color.h, 93
- BOOST\_AUTO\_TEST\_CASE
  - test\_suites.cpp, 153–156
- BOOST\_TEST\_MODULE
  - test\_suites.cpp, 153
- callback
  - scenario, 77
- CIRCLE\_DISTANCE
  - steeringBehavior.h, 121
- CIRCLE\_RADIUS
  - steeringBehavior.h, 121
- cohesion
  - steeringBehavior, 80
- color, 21
  - B, 23
  - color, 22
  - G, 24
  - getColor, 23
  - R, 24
- color.h
  - BLACK, 93
  - BLUE, 93
  - CYAN, 94
  - GREEN, 94
  - MAGENDA, 94
  - RED, 94
  - WHITE, 94
  - YELLOW, 94
- createAgent
  - scenario, 75
- createObstacle
  - obstacleAvoidance, 49
- createPath
  - pathFollower, 54
- createRandomArray
  - random, 73
- CYAN
  - color.h, 94
- desiredVelocity
  - agent, 19
- DISTANCE
  - prison.cpp, 144
- div
  - point, 57
  - pvector, 67
- dotProduct
  - pvector, 67

- draw
  - agent, 16
  - entity, 25
  - obstacle, 47
  - path, 52
- drawAgent
  - graphics, 34
- drawCircle
  - graphics, 35
- drawLine
  - graphics, 35
- drawPath
  - graphics, 36
- drawPoint
  - graphics, 36
- drawText
  - graphics, 36
- entity, 24
  - draw, 25
  - entity, 25
  - entityColor, 27
  - getId, 25
  - getName, 26
  - setId, 26
  - setName, 26
- entityColor
  - entity, 27
- ESC
  - graphics.h, 103
- EVADE
  - steeringBehavior.h, 121
- evade, 27
  - evade, 27
  - loop, 28
  - steeringBehavior, 81
- FIELD\_HEIGHT
  - flowField.h, 101
- FIELD\_WIDTH
  - flowField.h, 101
- FLEE
  - steeringBehavior.h, 121
- flee, 28
  - flee, 29
  - loop, 29
  - steeringBehavior, 81
- FLOCK
  - steeringBehavior.h, 122
- flock, 30
  - flock, 30
  - loop, 30
- flow
  - windy, 90
- flowField, 31
  - flowField, 32
  - getField, 32
- flowField.h
  - FIELD\_HEIGHT, 101
  - FIELD\_WIDTH, 101
  - GRAVITY, 101
  - WIND\_WEST, 101
- FOLLOW\_MOUSE
  - steeringBehavior.h, 122
- force
  - agent, 19
- forceInScreen
  - graphics, 37
- G
  - color, 24
- getAngle
  - pvector, 68
- getColor
  - color, 23
- getField
  - flowField, 32
- getId
  - entity, 25
- getMass
  - agent, 17
- getMousePosition
  - graphics, 37
- getName
  - agent, 17
  - entity, 26
- getNormalPoint
  - point, 57
- graphics, 33
  - drawAgent, 34
  - drawCircle, 35
  - drawLine, 35
  - drawPath, 36
  - drawPoint, 36
  - drawText, 36
  - forceInScreen, 37
  - getMousePosition, 37
  - handleKeypress, 37
  - handleResize, 38
  - initGraphics, 38
  - mouseButton, 40
  - mouseMove, 40
  - refreshScene, 41
  - target\_x, 41
  - target\_y, 42
  - timerEvent, 41
- graphics.h
  - ESC, 103
  - HEIGHT, 103
  - PI, 103
  - WIDTH, 103
- GRAVITY
  - flowField.h, 101
- GREEN
  - color.h, 94
- handleKeypress
  - graphics, 37

- handleResize
  - graphics, 38
- HEIGHT
  - graphics.h, 103
- id
  - agent, 20
- IN\_FLOW\_FIELD
  - steeringBehavior.h, 122
- include/agent.h, 91
- include/color.h, 92
- include/entity.h, 95
- include/evade.h, 96
- include/flee.h, 97
- include/flock.h, 98
- include/flowField.h, 100
- include/graphics.h, 102
- include/leaderFollower.h, 104
- include/mouseFollower.h, 105
- include/obstacle.h, 106
- include/obstacleAvoidance.h, 108
- include/path.h, 109
- include/pathFollower.h, 110
- include/point.h, 112
- include/prison.h, 113
- include/pursuit.h, 114
- include/pvector.h, 116
- include/random.h, 117
- include/scenario.h, 118
- include/steeringBehavior.h, 119
- include/wander.h, 123
- include/windy.h, 124
- inFlowField
  - steeringBehavior, 82
- initGL
  - scenario, 76
- initGraphics
  - graphics, 38
- LEADER\_FOLLOWER
  - steeringBehavior.h, 122
- leaderFollower, 42
  - leaderFollower, 43
  - leaderPosition, 44
  - leaderVelocity, 44
  - loop, 43
- leaderPosition
  - leaderFollower, 44
- leaderVelocity
  - leaderFollower, 44
- limit
  - pvector, 68
- loop
  - evade, 28
  - flee, 29
  - flock, 30
  - leaderFollower, 43
  - mouseFollower, 45
  - obstacleAvoidance, 50
  - pathFollower, 55
  - prison, 62
  - pursuit, 63
  - wander, 88
  - windy, 89
- MAGENDA
  - color.h, 94
- magnitude
  - pvector, 68
- main
  - main.cpp, 127
- main.cpp, 126
  - main, 127
  - menu, 127
  - mode, 128
- MAX\_NUMBER\_OF\_AGENTS
  - scenario.cpp, 148
- maxForce
  - agent, 20
- maxSpeed
  - agent, 20
- menu
  - main.cpp, 127
- mode
  - main.cpp, 128
- mouseButton
  - graphics, 40
- mouseFollower, 45
  - loop, 45
  - mouseFollower, 45
- mouseMove
  - graphics, 40
- mul
  - point, 58
  - pvector, 68
- myPath
  - pathFollower, 55
- name
  - scenario, 77
- normalize
  - pvector, 69
- obstacle, 46
  - draw, 47
  - obstacle, 47
  - p, 48
  - r, 48
- obstacleAvoidance, 48
  - createObstacle, 49
  - loop, 50
  - obstacleAvoidance, 49
  - obstacles, 50
- obstacles
  - obstacleAvoidance, 50
- operator+
  - point, 58, 59
  - pvector, 69, 70

- operator+=
  - pvector, 70
- operator-
  - point, 59
  - pvector, 71
- operator==
  - point, 60
  - pvector, 71
- p
  - obstacle, 48
- path, 51
  - addPoint, 52
  - draw, 52
  - path, 51
  - points, 53
  - width, 53
- pathFollower, 53
  - createPath, 54
  - loop, 55
  - myPath, 55
  - pathFollower, 54
- PI
  - graphics.h, 103
  - pvector.h, 117
- point, 55
  - div, 57
  - getNormalPoint, 57
  - mul, 58
  - operator+, 58, 59
  - operator-, 59
  - operator==, 60
  - point, 56
  - print, 60
  - x, 60
  - y, 61
- points
  - path, 53
- position
  - agent, 20
- print
  - point, 60
  - pvector, 72
- prison, 61
  - loop, 62
  - prison, 61
- prison.cpp
  - DISTANCE, 144
  - WALL, 145
- PURSUIT
  - steeringBehavior.h, 122
- pursuit, 62
  - loop, 63
  - pursuit, 63
  - steeringBehavior, 83
- pvector, 64
  - add, 66
  - angleBetween, 66
  - div, 67
  - dotProduct, 67
  - getAngle, 68
  - limit, 68
  - magnitude, 68
  - mul, 68
  - normalize, 69
  - operator+, 69, 70
  - operator+=, 70
  - operator-, 71
  - operator==, 71
  - print, 72
  - pvector, 65
  - x, 72
  - y, 72
- pvector.h
  - PI, 117
- R
  - color, 24
- r
  - agent, 20
  - obstacle, 48
- RANDOM
  - scenario.h, 119
- random, 73
  - createRandomArray, 73
- README.md, 128
- RED
  - color.h, 94
- refresh
  - scenario, 76
- refreshScene
  - graphics, 41
- scenario, 74
  - agents, 76
  - behavior, 77
  - callback, 77
  - createAgent, 75
  - initGL, 76
  - name, 77
  - refresh, 76
  - scenario, 75
  - view, 77
- scenario.cpp
  - MAX\_NUMBER\_OF\_AGENTS, 148
- scenario.h
  - RANDOM, 119
  - STATIC, 119
  - TROOP, 119
  - types, 119
- seek
  - steeringBehavior, 83
- separation
  - steeringBehavior, 84
- setAngle
  - steeringBehavior, 84
- setFeatures
  - agent, 17

- setId
  - entity, 26
- setMass
  - agent, 18
- setName
  - agent, 18
  - entity, 26
- src/agent.cpp, 128
- src/color.cpp, 129
- src/entity.cpp, 130
- src/evade.cpp, 131
- src/flee.cpp, 132
- src/flock.cpp, 133
- src/flowField.cpp, 134
- src/graphics.cpp, 135
- src/leaderFollower.cpp, 136
- src/mouseFollower.cpp, 137
- src/obstacle.cpp, 138
- src/obstacleAvoidance.cpp, 139
- src/path.cpp, 140
- src/pathFollower.cpp, 141
- src/point.cpp, 142
- src/prison.cpp, 143
- src/pursuit.cpp, 145
- src/pvector.cpp, 146
- src/random.cpp, 147
- src/scenario.cpp, 147
- src/steeringBehavior.cpp, 149
- src/wander.cpp, 150
- src/windy.cpp, 151
- STATIC
  - scenario.h, 119
- STAY\_IN\_FIELD
  - steeringBehavior.h, 122
- STAY\_IN\_PATH
  - steeringBehavior.h, 123
- stayInArea
  - steeringBehavior, 85
- stayInPath
  - steeringBehavior, 86
- steering
  - agent, 21
- steeringBehavior, 78
  - align, 79
  - avoid, 79
  - cohesion, 80
  - evade, 81
  - flee, 81
  - inFlowField, 82
  - pursuit, 83
  - seek, 83
  - separation, 84
  - setAngle, 84
  - stayInArea, 85
  - stayInPath, 86
  - wander, 86
- steeringBehavior.h
  - AVOID\_OBSTACLE, 121
  - CIRCLE\_DISTANCE, 121
  - CIRCLE\_RADIUS, 121
  - EVADE, 121
  - FLEE, 121
  - FLOCK, 122
  - FOLLOW\_MOUSE, 122
  - IN\_FLOW\_FIELD, 122
  - LEADER\_FOLLOWER, 122
  - PURSUIT, 122
  - STAY\_IN\_FIELD, 122
  - STAY\_IN\_PATH, 123
  - WANDER, 123
- target\_x
  - graphics, 41
- target\_y
  - graphics, 42
- targetPoint
  - agent, 21
- test/test\_suites.cpp, 152
- test\_suites.cpp
  - BOOST\_AUTO\_TEST\_CASE, 153–156
  - BOOST\_TEST\_MODULE, 153
- timerEvent
  - graphics, 41
- TROOP
  - scenario.h, 119
- types
  - scenario.h, 119
- updatePosition
  - agent, 18
- velocity
  - agent, 21
- view
  - scenario, 77
- WALL
  - prison.cpp, 145
- WANDER
  - steeringBehavior.h, 123
- wander, 87
  - loop, 88
  - steeringBehavior, 86
  - wander, 87
- WHITE
  - color.h, 94
- WIDTH
  - graphics.h, 103
- width
  - path, 53
- WIND\_WEST
  - flowField.h, 101
- windy, 88
  - flow, 90
  - loop, 89
  - windy, 89
- x

point, [60](#)  
pvector, [72](#)

y

point, [61](#)  
pvector, [72](#)

YELLOW

color.h, [94](#)