

Autonomous Steering Agents

Generated by Doxygen 1.8.17

1 Intent	1
1.1 Dependencies	1
1.2 Resources	1
2 Hierarchical Index	3
2.1 Class Hierarchy	3
3 Class Index	5
3.1 Class List	5
4 File Index	7
4.1 File List	7
5 Class Documentation	9
5.1 agent Class Reference	9
5.1.1 Detailed Description	10
5.1.2 Constructor & Destructor Documentation	10
5.1.2.1 agent() [1/2]	10
5.1.2.2 agent() [2/2]	10
5.1.2.3 ~agent()	11
5.1.3 Member Function Documentation	11
5.1.3.1 setFeatures()	11
5.1.3.2 updatePosition()	12
5.1.4 Member Data Documentation	12
5.1.4.1 acceleration	12
5.1.4.2 arrive	13
5.1.4.3 desiredVelocity	13
5.1.4.4 fillColor	13
5.1.4.5 force	13
5.1.4.6 id	13
5.1.4.7 mass	14
5.1.4.8 maxForce	14
5.1.4.9 maxSpeed	14
5.1.4.10 name	14
5.1.4.11 position	14
5.1.4.12 r	15
5.1.4.13 steering	15
5.1.4.14 targetPoint	15
5.1.4.15 velocity	15
5.2 color Class Reference	15
5.2.1 Detailed Description	16
5.2.2 Constructor & Destructor Documentation	16
5.2.2.1 color() [1/2]	16
5.2.2.2 color() [2/2]	16

5.2.3 Member Function Documentation	17
5.2.3.1 createColors()	17
5.2.3.2 getColor()	17
5.2.4 Member Data Documentation	18
5.2.4.1 B	18
5.2.4.2 colors	18
5.2.4.3 G	18
5.2.4.4 R	19
5.3 evade Class Reference	19
5.3.1 Detailed Description	19
5.3.2 Constructor & Destructor Documentation	19
5.3.2.1 evade()	20
5.3.3 Member Function Documentation	20
5.3.3.1 loop()	20
5.4 flee Class Reference	20
5.4.1 Detailed Description	21
5.4.2 Constructor & Destructor Documentation	21
5.4.2.1 flee()	21
5.4.3 Member Function Documentation	21
5.4.3.1 loop()	21
5.5 flock Class Reference	22
5.5.1 Detailed Description	22
5.5.2 Constructor & Destructor Documentation	22
5.5.2.1 flock()	22
5.5.3 Member Function Documentation	22
5.5.3.1 loop()	23
5.6 flowField Class Reference	23
5.6.1 Detailed Description	23
5.6.2 Constructor & Destructor Documentation	24
5.6.2.1 flowField() [1/2]	24
5.6.2.2 flowField() [2/2]	24
5.6.3 Member Function Documentation	24
5.6.3.1 getField()	24
5.7 graphics Class Reference	25
5.7.1 Detailed Description	26
5.7.2 Member Function Documentation	26
5.7.2.1 drawAgent()	26
5.7.2.2 drawCircle()	27
5.7.2.3 drawLine()	27
5.7.2.4 drawPath()	28
5.7.2.5 drawPoint()	28
5.7.2.6 drawText()	29

5.7.2.7 forceInScreen()	29
5.7.2.8 getMousePosition()	29
5.7.2.9 handleKeypress()	30
5.7.2.10 handleResize()	30
5.7.2.11 initGraphics()	30
5.7.2.12 mouseButton()	32
5.7.2.13 mouseMove()	32
5.7.2.14 refreshScene()	33
5.7.2.15 timerEvent()	33
5.7.3 Member Data Documentation	33
5.7.3.1 target_x	34
5.7.3.2 target_y	34
5.8 mouseFollower Class Reference	34
5.8.1 Detailed Description	34
5.8.2 Constructor & Destructor Documentation	35
5.8.2.1 mouseFollower()	35
5.8.3 Member Function Documentation	35
5.8.3.1 loop()	35
5.9 obstacle Class Reference	35
5.9.1 Detailed Description	36
5.9.2 Constructor & Destructor Documentation	36
5.9.2.1 obstacle() [1/2]	36
5.9.2.2 obstacle() [2/2]	36
5.9.3 Member Data Documentation	37
5.9.3.1 p	37
5.9.3.2 r	37
5.10 obstacleAvoidance Class Reference	37
5.10.1 Detailed Description	38
5.10.2 Constructor & Destructor Documentation	38
5.10.2.1 obstacleAvoidance()	38
5.10.3 Member Function Documentation	38
5.10.3.1 createObstacle()	38
5.10.3.2 loop()	39
5.10.4 Member Data Documentation	39
5.10.4.1 obstacles	40
5.11 path Class Reference	40
5.11.1 Detailed Description	40
5.11.2 Constructor & Destructor Documentation	40
5.11.2.1 path() [1/2]	41
5.11.2.2 path() [2/2]	41
5.11.3 Member Function Documentation	41
5.11.3.1 addPoint()	41

5.11.4 Member Data Documentation	42
5.11.4.1 points	42
5.11.4.2 width	42
5.12 pathFollower Class Reference	42
5.12.1 Detailed Description	43
5.12.2 Constructor & Destructor Documentation	43
5.12.2.1 pathFollower()	43
5.12.3 Member Function Documentation	43
5.12.3.1 createPath()	43
5.12.3.2 loop()	44
5.12.4 Member Data Documentation	44
5.12.4.1 myPath	44
5.13 point Class Reference	45
5.13.1 Detailed Description	45
5.13.2 Constructor & Destructor Documentation	45
5.13.2.1 point() [1/2]	46
5.13.2.2 point() [2/2]	46
5.13.3 Member Function Documentation	46
5.13.3.1 div()	46
5.13.3.2 getNormalPoint()	47
5.13.3.3 mul()	47
5.13.3.4 operator+() [1/2]	48
5.13.3.5 operator+() [2/2]	48
5.13.3.6 operator-()	49
5.13.3.7 operator==()	49
5.13.3.8 print()	49
5.13.4 Member Data Documentation	50
5.13.4.1 x	50
5.13.4.2 y	50
5.14 prison Class Reference	50
5.14.1 Detailed Description	51
5.14.2 Constructor & Destructor Documentation	51
5.14.2.1 prison()	51
5.14.3 Member Function Documentation	51
5.14.3.1 loop()	51
5.15 pursuit Class Reference	52
5.15.1 Detailed Description	52
5.15.2 Constructor & Destructor Documentation	52
5.15.2.1 pursuit()	52
5.15.3 Member Function Documentation	52
5.15.3.1 loop()	53
5.16 pvector Class Reference	53

5.16.1 Detailed Description	54
5.16.2 Constructor & Destructor Documentation	54
5.16.2.1 pvector() [1/2]	54
5.16.2.2 pvector() [2/2]	55
5.16.3 Member Function Documentation	55
5.16.3.1 add()	55
5.16.3.2 angleBetween()	55
5.16.3.3 div()	56
5.16.3.4 dotProduct()	56
5.16.3.5 getAngle()	57
5.16.3.6 limit()	57
5.16.3.7 magnitude()	57
5.16.3.8 mul()	58
5.16.3.9 normalize()	58
5.16.3.10 operator+() [1/2]	58
5.16.3.11 operator+() [2/2]	59
5.16.3.12 operator+=()	59
5.16.3.13 operator-() [1/2]	60
5.16.3.14 operator-() [2/2]	60
5.16.3.15 operator==()	61
5.16.3.16 print()	61
5.16.4 Member Data Documentation	61
5.16.4.1 x	61
5.16.4.2 y	62
5.17 random Class Reference	62
5.17.1 Detailed Description	62
5.17.2 Member Function Documentation	62
5.17.2.1 createRandomArray()	62
5.18 scenario Class Reference	63
5.18.1 Detailed Description	64
5.18.2 Constructor & Destructor Documentation	64
5.18.2.1 scenario()	64
5.18.3 Member Function Documentation	64
5.18.3.1 createAgent()	64
5.18.3.2 initGL()	65
5.18.3.3 refresh()	65
5.18.4 Member Data Documentation	65
5.18.4.1 agents	66
5.18.4.2 behavior	66
5.18.4.3 callback	66
5.18.4.4 myColor	66
5.18.4.5 name	67

5.18.4.6 view	67
5.19 steeringBehavior Class Reference	67
5.19.1 Detailed Description	68
5.19.2 Member Function Documentation	68
5.19.2.1 align()	68
5.19.2.2 avoid()	69
5.19.2.3 cohesion()	70
5.19.2.4 evade()	70
5.19.2.5 flee()	71
5.19.2.6 inFlowField()	72
5.19.2.7 pursuit()	72
5.19.2.8 seek()	73
5.19.2.9 separation()	73
5.19.2.10 setAngle()	74
5.19.2.11 stayInArea()	74
5.19.2.12 stayInPath()	75
5.19.2.13 wander()	76
5.20 wander Class Reference	76
5.20.1 Detailed Description	77
5.20.2 Constructor & Destructor Documentation	77
5.20.2.1 wander()	77
5.20.3 Member Function Documentation	77
5.20.3.1 loop()	78
5.21 windy Class Reference	78
5.21.1 Detailed Description	79
5.21.2 Constructor & Destructor Documentation	79
5.21.2.1 windy()	79
5.21.3 Member Function Documentation	79
5.21.3.1 loop()	79
5.21.4 Member Data Documentation	80
5.21.4.1 flow	80
6 File Documentation	81
6.1 include/agent.h File Reference	81
6.2 include/color.h File Reference	81
6.2.1 Detailed Description	82
6.2.2 Enumeration Type Documentation	82
6.2.2.1 num	82
6.3 include/evade.h File Reference	82
6.3.1 Detailed Description	83
6.4 include/flee.h File Reference	83
6.4.1 Detailed Description	83

6.5 include/flock.h File Reference	83
6.5.1 Detailed Description	84
6.6 include/flowField.h File Reference	84
6.6.1 Detailed Description	84
6.6.2 Macro Definition Documentation	84
6.6.2.1 FIELD_HEIGHT	85
6.6.2.2 FIELD_WIDTH	85
6.6.2.3 GRAVITY	85
6.6.2.4 WIND_WEST	85
6.7 include/graphics.h File Reference	85
6.7.1 Detailed Description	86
6.7.2 Macro Definition Documentation	86
6.7.2.1 ESC	86
6.7.2.2 HEIGHT	86
6.7.2.3 PI	86
6.7.2.4 WIDTH	86
6.8 include/mouseFollower.h File Reference	87
6.8.1 Detailed Description	87
6.9 include/obstacle.h File Reference	87
6.9.1 Detailed Description	87
6.10 include/obstacleAvoidance.h File Reference	88
6.10.1 Detailed Description	88
6.11 include/path.h File Reference	88
6.11.1 Detailed Description	88
6.12 include/pathFollower.h File Reference	89
6.12.1 Detailed Description	89
6.13 include/point.h File Reference	89
6.13.1 Detailed Description	89
6.14 include/prison.h File Reference	90
6.14.1 Detailed Description	90
6.15 include/pursuit.h File Reference	90
6.15.1 Detailed Description	90
6.16 include/pvector.h File Reference	91
6.16.1 Detailed Description	91
6.16.2 Macro Definition Documentation	91
6.16.2.1 PI	91
6.17 include/random.h File Reference	91
6.17.1 Detailed Description	92
6.18 include/scenario.h File Reference	92
6.18.1 Detailed Description	92
6.18.2 Enumeration Type Documentation	92
6.18.2.1 types	92

6.19 include/steeringBehavior.h File Reference	93
6.19.1 Detailed Description	93
6.19.2 Macro Definition Documentation	94
6.19.2.1 AVOID_OBSTACLE	94
6.19.2.2 CIRCLE_DISTANCE	94
6.19.2.3 CIRCLE_RADIUS	94
6.19.2.4 EVADE	94
6.19.2.5 FLEE	94
6.19.2.6 FLOCK	95
6.19.2.7 FOLLOW_MOUSE	95
6.19.2.8 IN_FLOW_FIELD	95
6.19.2.9 PURSUIT	95
6.19.2.10 STAY_IN_FIELD	95
6.19.2.11 STAY_IN_PATH	95
6.19.2.12 WANDER	96
6.20 include/wander.h File Reference	96
6.20.1 Detailed Description	96
6.21 include/windy.h File Reference	96
6.21.1 Detailed Description	97
6.22 main.cpp File Reference	97
6.22.1 Detailed Description	97
6.22.2 Function Documentation	98
6.22.2.1 main()	98
6.22.2.2 menu()	98
6.22.3 Variable Documentation	99
6.22.3.1 mode	99
6.23 README.md File Reference	99
6.24 src/agent.cpp File Reference	99
6.24.1 Detailed Description	99
6.25 src/color.cpp File Reference	99
6.25.1 Detailed Description	100
6.26 src/evade.cpp File Reference	100
6.26.1 Detailed Description	100
6.27 src/flee.cpp File Reference	100
6.27.1 Detailed Description	101
6.28 src/flock.cpp File Reference	101
6.28.1 Detailed Description	101
6.29 src/flowField.cpp File Reference	101
6.29.1 Detailed Description	101
6.30 src/graphics.cpp File Reference	102
6.30.1 Detailed Description	102
6.31 src/mouseFollower.cpp File Reference	102

6.31.1 Detailed Description	102
6.32 src/obstacle.cpp File Reference	103
6.32.1 Detailed Description	103
6.33 src/obstacleAvoidance.cpp File Reference	103
6.33.1 Detailed Description	103
6.34 src/path.cpp File Reference	104
6.34.1 Detailed Description	104
6.35 src/pathFollower.cpp File Reference	104
6.35.1 Detailed Description	104
6.36 src/point.cpp File Reference	104
6.36.1 Detailed Description	105
6.37 src/prison.cpp File Reference	105
6.37.1 Detailed Description	105
6.37.2 Macro Definition Documentation	105
6.37.2.1 DISTANCE	105
6.37.2.2 WALL	106
6.38 src/pursuit.cpp File Reference	106
6.38.1 Detailed Description	106
6.39 src/pvector.cpp File Reference	106
6.39.1 Detailed Description	106
6.40 src/random.cpp File Reference	107
6.40.1 Detailed Description	107
6.41 src/scenario.cpp File Reference	107
6.41.1 Detailed Description	107
6.41.2 Macro Definition Documentation	108
6.41.2.1 MAX_NUMBER_OF_AGENTS	108
6.42 src/steeringBehavior.cpp File Reference	108
6.42.1 Detailed Description	108
6.43 src/wander.cpp File Reference	108
6.43.1 Detailed Description	109
6.44 src/windy.cpp File Reference	109
6.44.1 Detailed Description	109
6.45 test/test_suites.cpp File Reference	109
6.45.1 Detailed Description	110
6.45.2 Macro Definition Documentation	110
6.45.2.1 BOOST_TEST_MODULE	110
6.45.3 Function Documentation	111
6.45.3.1 BOOST_AUTO_TEST_CASE() [1/12]	111
6.45.3.2 BOOST_AUTO_TEST_CASE() [2/12]	111
6.45.3.3 BOOST_AUTO_TEST_CASE() [3/12]	111
6.45.3.4 BOOST_AUTO_TEST_CASE() [4/12]	112
6.45.3.5 BOOST_AUTO_TEST_CASE() [5/12]	112

6.45.3.6 BOOST_AUTO_TEST_CASE() [6/12]	112
6.45.3.7 BOOST_AUTO_TEST_CASE() [7/12]	112
6.45.3.8 BOOST_AUTO_TEST_CASE() [8/12]	113
6.45.3.9 BOOST_AUTO_TEST_CASE() [9/12]	113
6.45.3.10 BOOST_AUTO_TEST_CASE() [10/12]	113
6.45.3.11 BOOST_AUTO_TEST_CASE() [11/12]	114
6.45.3.12 BOOST_AUTO_TEST_CASE() [12/12]	114

Index	115
--------------	------------

Chapter 1

Intent

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

1.1 Dependencies

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

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

1.2 Resources

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

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

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

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

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

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

agent	9
color	15
flowField	23
graphics	25
obstacle	35
path	40
point	45
pvector	53
random	62
scenario	63
evade	19
flee	20
flock	22
mouseFollower	34
obstacleAvoidance	37
pathFollower	42
prison	50
pursuit	52
wander	76
windy	78
steeringBehavior	67

Chapter 3

Class Index

3.1 Class List

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

agent	9
color	15
evade	19
flee	20
flock	22
flowField	23
graphics	25
mouseFollower	34
obstacle	35
obstacleAvoidance	37
path	40
pathFollower	42
point	45
prison	50
pursuit	52
pvector	53
random	62
scenario	63
steeringBehavior	67
wander	76
windy	78

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

main.cpp	Client code	97
include/agent.h	Agent class defines all agent specifications	81
include/color.h	Color class used for agent, path, wall etc. color	81
include/evade.h	Evade class inherited from scenario class	82
include/flee.h	Agents flee from mouse scenario	83
include/flock.h	Flocking agents scenario	83
include/flowField.h	FlowField class, screen can be filled with a force for each pixel	84
include/graphics.h	Graphics class, drives openGL	85
include/mouseFollower.h	Agents follow mouse scenario	87
include/obstacle.h	Circular obstacles for agent avoidance behaviors	87
include/obstacleAvoidance.h	Agents avoid from obstacles scenario	88
include/path.h	Path class used for path following steering behaviors	88
include/pathFollower.h	Path following scenario	89
include/point.h	Point class used for point operations	89
include/prison.h	Agents cant escape from field scenario	90
include/pursuit.h	One agent pursue other one scenario	90
include/pvector.h	Pvector class used for 2D vector operations	91
include/random.h	Utility class for random operations	91

include/ scenario.h	
Base class for all scenarios	92
include/ steeringBehavior.h	
Functions for autonomous steering behaviors	93
include/ wander.h	
Random wandering agents scenario	96
include/ windy.h	
Windy air scenario	96
src/ agent.cpp	
Implementation of the agent class	99
src/ color.cpp	
Color class implementation	99
src/ evade.cpp	
Evade class implementation	100
src/ flee.cpp	
Flee class implementation	100
src/ flock.cpp	
Flock class implementation	101
src/ flowField.cpp	
FlowField class implementation	101
src/ graphics.cpp	
Graphics class implementation	102
src/ mouseFollower.cpp	
MouseFollower class implementation	102
src/ obstacle.cpp	
Obstacle class implementation	103
src/ obstacleAvoidance.cpp	
ObstacleAvoidance class implementation	103
src/ path.cpp	
Path class implementation	104
src/ pathFollower.cpp	
PathFollower class implementation	104
src/ point.cpp	
Point class implementation file	104
src/ prison.cpp	
Prison class implementation	105
src/ pursuit.cpp	
Prison class implementation	106
src/ pvector.cpp	
Pvector class implementation	106
src/ random.cpp	
Utility class for random operations	107
src/ scenario.cpp	
Scenario base class implementation	107
src/ steeringBehavior.cpp	
Implementation of autonomous steering behaviors	108
src/ wander.cpp	
Wander class implementation	108
src/ windy.cpp	
Windy class implementation	109
test/ test_suites.cpp	
Unit test suites	109

Chapter 5

Class Documentation

5.1 agent Class Reference

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

Collaboration diagram for agent:

Public Member Functions

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

Public Attributes

- string `name`
name of the agent
- color `fillColor`
color of the agent
- point `position`
position of the agent
- pvector `velocity`
velocity of the agent
- point `targetPoint`
target of the agent
- float `maxSpeed`
maximum speed of the agent

- float [maxForce](#)
maximum force of the agent
- [pvector steering](#)
steering force of the apply
- [pvector force](#)
force of the agent
- [pvector acceleration](#)
acceleration of the agent
- [pvector desiredVelocity](#)
desired velocity of the agent
- float [r](#)
radius of the agent
- float [mass](#)
mass of the agent
- int [id](#)
id of the agent
- bool [arrive](#) = false
has arriving behavior or not

5.1.1 Detailed Description

Definition at line 20 of file agent.h.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 [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 }
```

5.1.2.2 [agent\(\)](#) [2/2]

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

constructor.

Parameters

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

See also

[agent\(\)](#)

Definition at line 21 of file agent.cpp.

```

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

5.1.2.3 ~agent()

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

destructor

Definition at line 62 of file agent.cpp.

```

63 {
64
65 }
```

5.1.3 Member Function Documentation

5.1.3.1 setFeatures()

```

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

initialize the agent attributes

Parameters

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

Definition at line 54 of file agent.cpp.

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

5.1.3.2 updatePosition()

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

position update calculations

Parameters

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

See also

[agent\(\)](#)

Definition at line 33 of file agent.cpp.

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

Here is the call graph for this function:

5.1.4 Member Data Documentation

5.1.4.1 acceleration

`pvector agent::acceleration`

acceleration of the agent

Definition at line 105 of file agent.h.

5.1.4.2 arrive

```
bool agent::arrive = false
```

has arriving behavior or not

Definition at line 130 of file agent.h.

5.1.4.3 desiredVelocity

```
pvector agent::desiredVelocity
```

desired velocity of the agent

Definition at line 110 of file agent.h.

5.1.4.4 fillColor

```
color agent::fillColor
```

color of the agent

Definition at line 65 of file agent.h.

5.1.4.5 force

```
pvector agent::force
```

force of the agent

Definition at line 100 of file agent.h.

5.1.4.6 id

```
int agent::id
```

id of the agent

Definition at line 125 of file agent.h.

5.1.4.7 mass

```
float agent::mass
```

mass of the agent

Definition at line 120 of file agent.h.

5.1.4.8 maxForce

```
float agent::maxForce
```

maximum force of the agent

Definition at line 90 of file agent.h.

5.1.4.9 maxSpeed

```
float agent::maxSpeed
```

maximum speed of the agent

Definition at line 85 of file agent.h.

5.1.4.10 name

```
string agent::name
```

name of the agent

Definition at line 60 of file agent.h.

5.1.4.11 position

```
point agent::position
```

position of the agent

Definition at line 70 of file agent.h.

5.1.4.12 r

```
float agent::r
```

radius of the agent

Definition at line 115 of file agent.h.

5.1.4.13 steering

```
pvector agent::steering
```

steering force of the apply

Definition at line 95 of file agent.h.

5.1.4.14 targetPoint

```
point agent::targetPoint
```

target of the agent

Definition at line 80 of file agent.h.

5.1.4.15 velocity

```
pvector agent::velocity
```

velocity of the agent

Definition at line 75 of file agent.h.

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

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

5.2 color Class Reference

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

Collaboration diagram for color:

Public Member Functions

- [color](#) ()
default constructor.
- [color](#) (float r, float g, float b)
constructor.
- void [createColors](#) ()
creation of fundamental 8 colors
- [color](#) [getColor](#) (int i)
gets requested color

Public Attributes

- float [R](#)
portion of red color
- float [G](#)
portion of green color
- float [B](#)
portion of blue color
- vector< [color](#) > [colors](#)
storage structure of created fundamental colors

5.2.1 Detailed Description

Definition at line 19 of file color.h.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 [color\(\)](#) [1/2]

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

default constructor.

See also

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

Definition at line 25 of file color.cpp.

```
26 {  
27  
28 }
```

5.2.2.2 [color\(\)](#) [2/2]

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

constructor.

Parameters

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

See also

[path\(\)](#)

Definition at line 13 of file color.cpp.

```

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

5.2.3 Member Function Documentation

5.2.3.1 createColors()

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

creation of fundamental 8 colors

Definition at line 30 of file color.cpp.

```

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

5.2.3.2 getColor()

```
color color::getColor (
    int i )
```

gets requested color

Parameters

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

Returns

requested color

Definition at line 20 of file color.cpp.

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

Here is the caller graph for this function:

5.2.4 Member Data Documentation

5.2.4.1 B

```
float color::B
```

portion of blue color

Definition at line 61 of file color.h.

5.2.4.2 colors

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

storage structure of created fundamental colors

Definition at line 66 of file color.h.

5.2.4.3 G

```
float color::G
```

portion of green color

Definition at line 56 of file color.h.

5.2.4.4 R

```
float color::R
```

portion of red color

Definition at line 51 of file color.h.

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

- include/color.h
- src/color.cpp

5.3 evade Class Reference

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

Inheritance diagram for evade:

Collaboration diagram for evade:

Public Member Functions

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

Static Public Member Functions

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

Additional Inherited Members

5.3.1 Detailed Description

Definition at line 15 of file evade.h.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 evade()

```
evade::evade ( )
```

default constructor.

Definition at line 31 of file evade.cpp.

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

5.3.3 Member Function Documentation

5.3.3.1 loop()

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

loop function of evading scenario

Note

opengl callback forces that function to be static

Definition at line 15 of file evade.cpp.

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

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

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

5.4 flee Class Reference

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

Inheritance diagram for flee:

Collaboration diagram for flee:

Public Member Functions

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

Static Public Member Functions

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

Additional Inherited Members

5.4.1 Detailed Description

Definition at line 14 of file flee.h.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 flee()

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

default constructor.

Definition at line 24 of file flee.cpp.

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

5.4.3 Member Function Documentation

5.4.3.1 loop()

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

evading scenario loop function

Note

opengl callback forces that function to be static

Definition at line 15 of file flee.cpp.

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

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

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

5.5 flock Class Reference

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

Inheritance diagram for flock:

Collaboration diagram for flock:

Public Member Functions

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

Static Public Member Functions

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

Additional Inherited Members

5.5.1 Detailed Description

Definition at line 15 of file flock.h.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 flock()

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

default constructor.

Definition at line 36 of file flock.cpp.

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

5.5.3 Member Function Documentation

5.5.3.1 loop()

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

flocking scenario loop function

Note

opengl callback forces that function to be static

Definition at line 15 of file flock.cpp.

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

Here is the call graph for this function:

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

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

5.6 flowField Class Reference

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

Collaboration diagram for flowField:

Public Member Functions

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

5.6.1 Detailed Description

Definition at line 18 of file flowField.h.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 flowField() [1/2]

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

default constructor.

See also

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

Definition at line 15 of file flowField.cpp.

```
16 {  
17  
18 }
```

5.6.2.2 flowField() [2/2]

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

constructor.

Parameters

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

See also

[flowField\(\)](#)

Definition at line 10 of file flowField.cpp.

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

5.6.3 Member Function Documentation

5.6.3.1 getField()

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

get force at individual pixel

Parameters

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

Returns

force at specified position

Definition at line 39 of file flowField.cpp.

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

Here is the caller graph for this function:

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

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

5.7 graphics Class Reference

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

Collaboration diagram for graphics:

Public Member Functions

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

Static Public Member Functions

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

Static Public Attributes

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

5.7.1 Detailed Description

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

5.7.2 Member Function Documentation

5.7.2.1 `drawAgent()`

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

drawing with corresponding angle

Parameters

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

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

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

```

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

```

Here is the call graph for this function:

5.7.2.2 drawCircle()

```

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

```

draws circle

Parameters

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

Definition at line 138 of file graphics.cpp.

```

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

```

5.7.2.3 drawLine()

```

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

```

drawing line

Parameters

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

Definition at line 128 of file graphics.cpp.

```

129 {
130     glColor3f( c1.R, c1.G, c1.B);
131     glLineWidth(2);
132     glBegin(GL_LINES);
133     glVertex2f(p1.x, p1.y);

```

```

134     glVertex2f(p2.x, p2.y);
135     glEnd();
136 }

```

5.7.2.4 drawPath()

```

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

```

draws path

Parameters

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

Definition at line 114 of file graphics.cpp.

```

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

```

Here is the call graph for this function:

5.7.2.5 drawPoint()

```

void graphics::drawPoint (
    point p )

```

draws point

Parameters

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

Definition at line 151 of file graphics.cpp.

```

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

```

Here is the caller graph for this function:

5.7.2.6 drawText()

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

draws text on screen

Parameters

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

Definition at line 21 of file graphics.cpp.

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

Here is the caller graph for this function:

5.7.2.7 forceInScreen()

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

changes agent position so that it stays in screen

Parameters

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

Definition at line 63 of file graphics.cpp.

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

5.7.2.8 getMousePosition()

```
point graphics::getMousePosition ( )
```

gets mouse position

Definition at line 58 of file graphics.cpp.

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

Here is the call graph for this function:

5.7.2.9 handleKeypress()

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

key press event

Parameters

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

Definition at line 107 of file graphics.cpp.

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

Here is the caller graph for this function:

5.7.2.10 handleResize()

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

event triggered with screen resizing

Parameters

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

Definition at line 83 of file graphics.cpp.

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

Here is the caller graph for this function:

5.7.2.11 initGraphics()

```
void graphics::initGraphics (
    int * argv,
```

```
char ** argc,  
void(*)() callback )
```

initialization of graphics

Parameters

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

Definition at line 41 of file graphics.cpp.

```

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

Here is the call graph for this function:

5.7.2.12 mouseButton()

```

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

mouse press event

Parameters

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

Definition at line 101 of file graphics.cpp.

```

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

Here is the caller graph for this function:

5.7.2.13 mouseMove()

```

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

event triggered with mouse movements

Parameters

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

Definition at line 75 of file graphics.cpp.

```

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

Here is the caller graph for this function:

5.7.2.14 refreshScene()

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

update agent position

Definition at line 32 of file graphics.cpp.

```

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

5.7.2.15 timerEvent()

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

periodic timer event

Parameters

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

Definition at line 95 of file graphics.cpp.

```

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

Here is the caller graph for this function:

5.7.3 Member Data Documentation

5.7.3.1 target_x

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

mouse position x

Definition at line 129 of file graphics.h.

5.7.3.2 target_y

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

mouse position y

Definition at line 134 of file graphics.h.

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

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

5.8 mouseFollower Class Reference

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

Inheritance diagram for mouseFollower:

Collaboration diagram for mouseFollower:

Public Member Functions

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

Static Public Member Functions

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

Additional Inherited Members

5.8.1 Detailed Description

Definition at line 14 of file mouseFollower.h.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 mouseFollower()

```
mouseFollower::mouseFollower ( )
```

default constructor.

Definition at line 25 of file mouseFollower.cpp.

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

5.8.3 Member Function Documentation

5.8.3.1 loop()

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

mouse following scenario loop function

Note

opengl callback forces that function to be static

Definition at line 15 of file mouseFollower.cpp.

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

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

- include/mouseFollower.h
- src/mouseFollower.cpp

5.9 obstacle Class Reference

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

Collaboration diagram for obstacle:

Public Member Functions

- [obstacle](#) ()
default constructor.
- [obstacle](#) ([point](#) p, float r)
constructor

Public Attributes

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

5.9.1 Detailed Description

Definition at line 12 of file obstacle.h.

5.9.2 Constructor & Destructor Documentation

5.9.2.1 [obstacle\(\)](#) [1/2]

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

default constructor.

See also

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

Definition at line 15 of file obstacle.cpp.

```
16 {  
17  
18 }
```

5.9.2.2 [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 20 of file obstacle.cpp.

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

5.9.3 Member Data Documentation

5.9.3.1 p

`point obstacle::p`

center point of the obstacle

Definition at line 31 of file obstacle.h.

5.9.3.2 r

`float obstacle::r`

radius of the obstacle

Definition at line 36 of file obstacle.h.

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

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

5.10 obstacleAvoidance Class Reference

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

Inheritance diagram for obstacleAvoidance:

Collaboration diagram for obstacleAvoidance:

Public Member Functions

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

Static Public Member Functions

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

Static Public Attributes

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

Additional Inherited Members

5.10.1 Detailed Description

Definition at line 15 of file obstacleAvoidance.h.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 obstacleAvoidance()

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

default constructor.

Definition at line 43 of file obstacleAvoidance.cpp.

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

5.10.3 Member Function Documentation

5.10.3.1 createObstacle()

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

creation of list of obstacles

Parameters

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

Note

opengl callback forces that function to be static

Definition at line 36 of file obstacleAvoidance.cpp.

```

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

Here is the call graph for this function:

5.10.3.2 loop()

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

obstacle avoidance scenario loop function

Note

opengl callback forces that function to be static

Definition at line 17 of file obstacleAvoidance.cpp.

```

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

Here is the call graph for this function:

5.10.4 Member Data Documentation

5.10.4.1 obstacles

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

list of obstacles

Note

opengl callback forces that function to be static

Definition at line 32 of file obstacleAvoidance.h.

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

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

5.11 path Class Reference

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

Collaboration diagram for path:

Public Member Functions

- [path](#) ()
default constructor.
- [path](#) (float [width](#))
donstructor.
- void [addPoint](#) ([point](#) p)
adds a new point to the path

Public Attributes

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

5.11.1 Detailed Description

Definition at line 15 of file path.h.

5.11.2 Constructor & Destructor Documentation

5.11.2.1 path() [1/2]

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

default constructor.

See also

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

Definition at line 16 of file path.cpp.

```
17 {  
18  
19 }
```

5.11.2.2 path() [2/2]

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

donstructor.

Parameters

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

See also

[path\(\)](#)

Definition at line 21 of file path.cpp.

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

5.11.3 Member Function Documentation

5.11.3.1 addPoint()

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

adds a new point to the path

Parameters

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

Definition at line 11 of file path.cpp.

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

Here is the caller graph for this function:

5.11.4 Member Data Documentation

5.11.4.1 [points](#)

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

list of points added to the path

Definition at line 39 of file path.h.

5.11.4.2 [width](#)

```
int path::width
```

width of the path

Definition at line 44 of file path.h.

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

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

5.12 pathFollower Class Reference

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

Inheritance diagram for pathFollower:

Collaboration diagram for pathFollower:

Public Member Functions

- `pathFollower()`
default constructor.

Static Public Member Functions

- static void `loop()`
path follower scenario loop function
- static void `createPath(path &p)`
creates path

Static Public Attributes

- static `path myPath`
path that will be followed

Additional Inherited Members

5.12.1 Detailed Description

Definition at line 14 of file pathFollower.h.

5.12.2 Constructor & Destructor Documentation

5.12.2.1 pathFollower()

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

default constructor.

Definition at line 37 of file pathFollower.cpp.

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

5.12.3 Member Function Documentation

5.12.3.1 createPath()

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

creates path

Parameters

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

Note

opengl callback forces that function to be static

Definition at line 29 of file pathFollower.cpp.

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

Here is the call graph for this function:

5.12.3.2 loop()

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

path follower scenario loop function

Note

opengl callback forces that function to be static

Definition at line 17 of file pathFollower.cpp.

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

Here is the call graph for this function:

5.12.4 Member Data Documentation

5.12.4.1 myPath

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

path that will be followed

Note

opengl callback forces that function to be static

Definition at line 38 of file pathFollower.h.

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

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

5.13 point Class Reference

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

Collaboration diagram for point:

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

5.13.1 Detailed Description

Definition at line 15 of file point.h.

5.13.2 Constructor & Destructor Documentation

5.13.2.1 `point()` [1/2]

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

default constructor

See also

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

Definition at line 21 of file point.cpp.

```
21 {}
```

Here is the caller graph for this function:

5.13.2.2 `point()` [2/2]

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

constructor

Parameters

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

See also

[point\(\)](#)

Definition at line 15 of file point.cpp.

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

5.13.3 Member Function Documentation

5.13.3.1 `div()`

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

divide point

Parameters

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

Definition at line 38 of file point.cpp.

```

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

Here is the caller graph for this function:

5.13.3.2 getNormalPoint()

```

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

provides normal point on a vector of a point

Parameters

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

Definition at line 67 of file point.cpp.

```

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

Here is the call graph for this function: Here is the caller graph for this function:

5.13.3.3 mul()

```

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

multiply point

Parameters

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

Definition at line 44 of file point.cpp.

```

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

```

Here is the caller graph for this function:

5.13.3.4 operator+() [1/2]

```

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

```

overloaded + operator

Parameters

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

Returns

sum

Definition at line 51 of file point.cpp.

```

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

```

5.13.3.5 operator+() [2/2]

```

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

```

overloaded + operator

Parameters

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

Returns

sum

Definition at line 23 of file point.cpp.

```

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

```

5.13.3.6 operator-()

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

overloaded - operator

Parameters

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

Returns

difference

Definition at line 59 of file point.cpp.

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

5.13.3.7 operator==()

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

overloaded == operator

Parameters

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

Returns

comparison result

Definition at line 31 of file point.cpp.

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

5.13.3.8 print()

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

debug function

Parameters

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

Definition at line 79 of file point.cpp.

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

5.13.4 Member Data Documentation

5.13.4.1 x

```
float point::x
```

x position

Definition at line 88 of file point.h.

5.13.4.2 y

```
float point::y
```

y position

Definition at line 93 of file point.h.

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

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

5.14 prison Class Reference

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

Inheritance diagram for prison:

Collaboration diagram for prison:

Public Member Functions

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

Static Public Member Functions

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

Additional Inherited Members

5.14.1 Detailed Description

Definition at line 15 of file prison.h.

5.14.2 Constructor & Destructor Documentation

5.14.2.1 prison()

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

default constructor.

Definition at line 31 of file prison.cpp.

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

5.14.3 Member Function Documentation

5.14.3.1 loop()

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

prisoning scenario loop function

prison loop function

Note

opengl callback forces that function to be static

Definition at line 18 of file prison.cpp.

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

Here is the call graph for this function:

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

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

5.15 pursuit Class Reference

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

Inheritance diagram for pursuit:

Collaboration diagram for pursuit:

Public Member Functions

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

Static Public Member Functions

- static void [loop](#) ()
pursuing scenario loop function

Additional Inherited Members

5.15.1 Detailed Description

Definition at line 14 of file pursuit.h.

5.15.2 Constructor & Destructor Documentation

5.15.2.1 pursuit()

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

default constructor.

Definition at line 31 of file pursuit.cpp.

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

5.15.3 Member Function Documentation

5.15.3.1 loop()

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

pursuing scenario loop function

Note

opengl callback forces that function to be static

Definition at line 15 of file pursuit.cpp.

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

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

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

5.16 pvector Class Reference

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

Collaboration diagram for pvector:

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

5.16.1 Detailed Description

Definition at line 17 of file pvector.h.

5.16.2 Constructor & Destructor Documentation

5.16.2.1 pvector() [1/2]

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

default constructor

See also

`pvector(float x, float y)`

Definition at line 35 of file pvector.cpp.

```
36 {
37
38 }
```

5.16.2.2 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 40 of file pvector.cpp.

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

5.16.3 Member Function Documentation

5.16.3.1 add()

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

addition of vectors

Parameters

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

Definition at line 58 of file pvector.cpp.

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

5.16.3.2 angleBetween()

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

angle calculation between two vectors

Parameters

<i>v</i>	vector to calculate angle
----------	---------------------------

Returns

angle

Definition at line 23 of file pvector.cpp.

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

Here is the call graph for this function: Here is the caller graph for this function:

5.16.3.3 div()

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

vector division

Parameters

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

Definition at line 46 of file pvector.cpp.

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

Here is the caller graph for this function:

5.16.3.4 dotProduct()

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

dot product of two vectors

Parameters

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

Returns

returns scalar dot product

Definition at line 30 of file pvector.cpp.

```

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

```

Here is the caller graph for this function:

5.16.3.5 getAngle()

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

calculates vector angle

Returns

angle

Definition at line 16 of file pvector.cpp.

```

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

```

Here is the caller graph for this function:

5.16.3.6 limit()

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

vector limitation

Parameters

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

Definition at line 83 of file pvector.cpp.

```

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

```

Here is the call graph for this function: Here is the caller graph for this function:

5.16.3.7 magnitude()

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

calculates magnitude of the vector

Returns

magnitude of the vector

Definition at line 64 of file pvector.cpp.

```

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

```

Here is the caller graph for this function:

5.16.3.8 mul()

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

vector multiplication

Parameters

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

Definition at line 52 of file pvector.cpp.

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

Here is the caller graph for this function:

5.16.3.9 normalize()

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

normalize

Returns

normalized vector

Definition at line 69 of file pvector.cpp.

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

Here is the caller graph for this function:

5.16.3.10 operator+() [1/2]

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

overloaded + operator

Parameters

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

Returns

sum

Definition at line 111 of file pvector.cpp.

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

5.16.3.11 operator+() [2/2]

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

overloaded + operator

Parameters

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

Returns

sum

Definition at line 89 of file pvector.cpp.

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

5.16.3.12 operator+=()

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

overloaded += operator

Parameters

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

Returns

sum

Definition at line 97 of file pvector.cpp.

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

5.16.3.13 operator-() [1/2]

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

overloaded - operator

Parameters

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

Returns

difference

Definition at line 119 of file pvector.cpp.

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

5.16.3.14 operator-() [2/2]

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

overloaded - operator

Parameters

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

Returns

difference

Definition at line 132 of file pvector.cpp.

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


5.16.3.15 operator==()

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

overloaded == operator

Parameters

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

Returns

comparison result

Definition at line 104 of file pvector.cpp.

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

5.16.3.16 print()

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

debug function

Parameters

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

Definition at line 127 of file pvector.cpp.

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

5.16.4 Member Data Documentation

5.16.4.1 x

```
float pvector::x
```

x magnitude of the vector

Definition at line 140 of file pvector.h.

5.16.4.2 y

```
float pvector::y
```

y magnitude of the vector

Definition at line 145 of file pvector.h.

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

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

5.17 random Class Reference

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

Collaboration diagram for random:

Static Public Member Functions

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

5.17.1 Detailed Description

Definition at line 9 of file random.h.

5.17.2 Member Function Documentation

5.17.2.1 createRandomArray()

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

random array generation

Parameters

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

Definition at line 14 of file random.cpp.

```

14                                     {
15     srand(time(NULL));
16     for(int i=0; i<size; i++)
17         arr[i] = i+1;
18
19     for (int i=0; i < size; i++){
20         int r = rand() % size;
21         swap(arr[i], arr[r]);
22     }
23 }
```

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

- include/random.h
- src/random.cpp

5.18 scenario Class Reference

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

Inheritance diagram for scenario:

Collaboration diagram for scenario:

Public Member Functions

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

Static Public Member Functions

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

Public Attributes

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

Static Public Attributes

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

5.18.1 Detailed Description

Definition at line 19 of file scenario.h.

5.18.2 Constructor & Destructor Documentation

5.18.2.1 scenario()

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

default constructor.

Definition at line 28 of file scenario.cpp.

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

5.18.3 Member Function Documentation

5.18.3.1 createAgent()

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

agent creation

Parameters

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

Definition at line 106 of file scenario.cpp.

```
107 {
108     if(type == TROOP){
109         createTroop(*count);
110     }
111     else if(type == RANDOM){
112         createRandomAgents(*count, *force, *speed);
113     }
114     else if(type == STATIC){
115         createStaticAgents();
116     }
```

```

116     }
117     else{
118         //error message
119     }
120 }

```

5.18.3.2 initGL()

```

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

```

graphics initialization

Parameters

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

Definition at line 22 of file scenario.cpp.

```

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

```

Here is the caller graph for this function:

5.18.3.3 refresh()

```

void scenario::refresh ( ) [static]

```

refreshes all items

Note

opengl callback forces that function to be static

Definition at line 35 of file scenario.cpp.

```

36 {
37     for(auto it = agents.begin(); it < agents.end(); it++){
38         (*it).updatePosition((*it).arrive);
39         view.drawAgent(*it, (*it).fillColor);
40     }
41
42     view.drawText(name, point(-34, 32.25)); //TODO: magic numbers, define left corner
43     view.refreshScene();
44 }

```

Here is the call graph for this function:

5.18.4 Member Data Documentation

5.18.4.1 agents

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

structure stores agents

Note

opengl callback forces that function to be static

Definition at line 52 of file scenario.h.

5.18.4.2 behavior

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

behavior instance used

Note

opengl callback forces that function to be static

Definition at line 64 of file scenario.h.

5.18.4.3 callback

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

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

Definition at line 81 of file scenario.h.

5.18.4.4 myColor

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

color instance used

Note

opengl callback forces that function to be static

Definition at line 70 of file scenario.h.

5.18.4.5 name

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

scenario name

Note

opengl callback forces that function to be static

Definition at line 76 of file scenario.h.

5.18.4.6 view

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

graphics instance used

Note

opengl callback forces that function to be static

Definition at line 58 of file scenario.h.

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

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

5.19 steeringBehavior Class Reference

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

Collaboration diagram for steeringBehavior:

Public Member Functions

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

5.19.1 Detailed Description

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

5.19.2 Member Function Documentation

5.19.2.1 align()

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

force to align

Parameters

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

Returns

force to be applied

Definition at line 117 of file steeringBehavior.cpp.

```

118 {
119     float neighborDist = 30; //TODO: magic number
120     pvector sum {0,0};
121     int count = 0;
122     for(auto it = boids.begin(); it < boids.end(); it++){
123         float d = (agent.position - (*it).position).magnitude();
124         if( (d >0) && (d < neighborDist) ){
125             sum += (*it).velocity;
126             count++;
127         }
128     }
129     if(count>0){
130         sum.div(count);
131         sum.normalize().mul(agent.maxSpeed);
132         agent.steering = sum - agent.velocity;
133         return agent.steering;
134     }
135     return pvector(0,0);
136 }
```

Here is the call graph for this function:

5.19.2.2 avoid()

```

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

force to avoid

Parameters

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

Returns

force to be applied

Definition at line 181 of file steeringBehavior.cpp.

```

182 {
183     float dynamic_length = agent.velocity.magnitude() / agent.maxSpeed;
184     pvector vel = agent.velocity;
185     vel.normalize().mul(dynamic_length);
186     pvector ahead = vel + agent.position;
187     vel.mul(6);
188     pvector ahead2 = vel + agent.position;
189     //view.drawPoint(point(ahead.x, ahead.y));
190     //view.drawPoint(point(ahead2.x, ahead2.y));
191
192     for(auto it = obstacles.begin(); it < obstacles.end(); it++){
193         float dist = (ahead - (*it).p).magnitude();
194         float dist2 = (ahead2 - (*it).p).magnitude();
195         if(dist < (*it).r + 2 || dist2 < (*it).r + 2){
196             pvector avoidance = ahead - (*it).p;
197             avoidance.normalize().mul(20);
198             /*a = point(avoidance.x, avoidance.y);
199             view.drawLine(agent.position, agent.position + a, color(0,1,0));*/
200             return avoidance;
201         }
202     }
203     return pvector(0,0);
204 }
```

Here is the call graph for this function:

5.19.2.3 cohesion()

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

force to cohesion

Parameters

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

Returns

force to be applied

Definition at line 138 of file steeringBehavior.cpp.

```
139 {
140     float neighborDist = 20; //TODO: magic number
141     point sum {0,0};
142     int count = 0;
143     for(auto it = boids.begin(); it < boids.end(); it++){
144         float d = (agent.position - (*it).position).magnitude();
145         if( (d > 0) && (d < neighborDist) ){
146             sum = sum + (*it).position;
147             count++;
148         }
149     }
150     if(count > 0){
151         sum.div(count);
152         agent.targetPoint = sum;
153         return seek(agent);
154     }
155     return pvector(0,0);
156 }
```

Here is the call graph for this function:

5.19.2.4 evade()

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

force to evade

Parameters

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

Returns

force to be applied

Definition at line 45 of file steeringBehavior.cpp.

```

46 {
47     agent target;
48     for(auto it = boids.begin(); it < boids.end(); it++){
49         if((*it).name == name){
50             target = *it;
51         }
52     }
53
54     point p = point(evader.position.x + 2, evader.position.y - 2);
55     view.drawText(evader.name, p);
56     p = point(target.position.x + 2, target.position.y - 2);
57     view.drawText(target.name, p);
58
59     pvector targetVel = target.velocity;
60     targetVel.mul(5); //TODO: magic number
61
62     point futurePos = target.position + targetVel;
63     view.drawPoint(futurePos);
64
65     pvector dist = evader.position - futurePos;
66     dist.normalize().mul( 1 / dist.magnitude() );
67
68     evader.targetPoint = evader.position + dist;
69     return flee(evader, view, futurePos);
70 }
```

Here is the call graph for this function:

5.19.2.5 flee()

```

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

force to flee

Parameters

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

Returns

force to be applied

Definition at line 28 of file steeringBehavior.cpp.

```

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

Here is the call graph for this function:

5.19.2.6 inFlowField()

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

gets flow field force

Parameters

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

Returns

force to be applied

Definition at line 236 of file steeringBehavior.cpp.

```
237 {
238     //pos_x, pos_y must be non negative integer
239     int pos_x = abs((int)agent.position.x) % WIDTH;
240     int pos_y = abs((int)agent.position.y) % HEIGHT;
241     //TODO: modification required for non uniform fields
242     return flow.getField(pos_x, pos_y);
243 }
```

Here is the call graph for this function:

5.19.2.7 pursuit()

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

force to pursue

Parameters

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

Returns

force to be applied

Definition at line 72 of file steeringBehavior.cpp.

```
73 {
74     agent target;
75     for(auto it = boids.begin(); it < boids.end(); it++){
76         if((*it).name == name){
```

```

77         target = *it;
78     }
79 }
80
81 point p = point(target.position.x + 2, target.position.y - 2);
82 view.drawText(target.name, p);
83 p = point(pursuer.position.x + 2, pursuer.position.y - 2);
84 view.drawText(pursuer.name, p);
85
86 float dist = (target.position - pursuer.position).magnitude();
87 float t = dist / target.maxSpeed;
88
89 pvector targetVel = target.velocity;
90 targetVel.mul(t);
91 point futurePos = target.position + targetVel;
92 pursuer.targetPoint = futurePos;
93 return seek(pursuer);
94 }

```

Here is the call graph for this function:

5.19.2.8 seek()

```

pvector steeringBehavior::seek (
    agent & agent )

```

force to seek

Parameters

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

Returns

force to be applied

Definition at line 206 of file steeringBehavior.cpp.

```

207 {
208     agent.desiredVelocity = agent.targetPoint - agent.position;
209     agent.steering = agent.desiredVelocity - agent.velocity;
210     return agent.steering;
211 }

```

5.19.2.9 separation()

```

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

```

force to separate

Parameters

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

Returns

force to be applied

Definition at line 158 of file steeringBehavior.cpp.

```

159 {
160     float desiredSeparation = 5; //TODO: magic number
161     pvector sum = pvector(0,0);
162     int count = 0;
163     for(auto it = agents.begin(); it < agents.end(); it++){
164         float d = (agent.position - (*it).position).magnitude();
165         if( (d > 0) && (d < desiredSeparation) ){
166             pvector diff = agent.position - (*it).position;
167             diff.normalize().div(d);
168             sum = sum + diff;
169             count++;
170         }
171     }
172     if(count > 0){
173         sum.div(count);
174         sum.normalize().mul(agent.maxSpeed);
175         agent.steering = sum - agent.velocity;
176         return agent.steering;
177     }
178     return pvector(0,0);
179 }
```

Here is the call graph for this function:

5.19.2.10 setAngle()

```

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

applies angle on vector

Parameters

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

Definition at line 22 of file steeringBehavior.cpp.

```

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

5.19.2.11 stayInArea()

```

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

gets reflection force

Parameters

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

Returns

force to be applied

Definition at line 245 of file steeringBehavior.cpp.

```

246 {
247     if(agent.position.x >= turnPoint){
248         agent.desiredVelocity = pvector( -agent.maxSpeed, agent.velocity.y );
249         agent.steering = agent.desiredVelocity - agent.velocity;
250         return agent.steering;
251     }
252     else if(agent.position.x <= -turnPoint){
253         agent.desiredVelocity = pvector( agent.maxSpeed, agent.velocity.y );
254         agent.steering = agent.desiredVelocity - agent.velocity;
255         return agent.steering;
256     }
257     else if(agent.position.y >= turnPoint){
258         agent.desiredVelocity = pvector( agent.velocity.x, -agent.maxSpeed );
259         agent.steering = agent.desiredVelocity - agent.velocity;
260         return agent.steering;
261     }
262     else if(agent.position.y <= -turnPoint){
263         agent.desiredVelocity = pvector( agent.velocity.x, agent.maxSpeed );
264         agent.steering = agent.desiredVelocity - agent.velocity;
265         return agent.steering;
266     }
267     return pvector(0,0);
268 }
```

5.19.2.12 stayInPath()

```

pvector steeringBehavior::stayInPath (
    agent & agent,
    path & path,
    graphics view )
```

gets force to follow path

Parameters

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

Returns

force to be applied

Definition at line 213 of file steeringBehavior.cpp.

```

214 {
215     float worldRecord = 1000000; //TODO: magic number
216     point normalPoint, predictedPos, start, end;
217     pvector distance;
218     for(auto it = path.points.begin(); it < path.points.end()-1; it++){
219         start = point ((*it).x, (*it).y);
220         end = point ((*it+1).x, (*it+1).y);
221         predictedPos = agent.position + agent.velocity;
222         normalPoint.getNormalPoint(predictedPos, start, end);
223         if (normalPoint.x < start.x || normalPoint.x > end.x){
224             normalPoint = end;
225         }
226         distance = predictedPos - normalPoint;
227         if (distance.magnitude() < worldRecord){
228             worldRecord = distance.magnitude();
229             agent.targetPoint = end;
230         }
231     }
```

```

230     }
231     view.drawPoint(agent.targetPoint);
232 }
233 return seek(agent);
234 }

```

Here is the call graph for this function:

5.19.2.13 wander()

```

pvector steeringBehavior::wander (
    agent & agent )

```

force to wander

Parameters

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

Returns

force to be applied

Definition at line 96 of file steeringBehavior.cpp.

```

97 {
98     pvector circleCenter = agent.velocity;
99     circleCenter.normalize().mul(CIRCLE_DISTANCE + CIRCLE_RADIUS);
100
101     int wanderAngle = (rand() % 360);
102     pvector displacement {0, 1};
103     setAngle(displacement, wanderAngle);
104     displacement.mul(CIRCLE_RADIUS);
105
106     agent.desiredVelocity = displacement + circleCenter;
107     agent.steering = agent.desiredVelocity - agent.velocity;
108
109     //move it to the center when it is out of screen
110     if(agent.position.x > WIDTH || agent.position.x < -WIDTH ||
111        agent.position.y > HEIGHT || agent.position.y < -HEIGHT)
112         agent.position = point(0,0);
113
114     return agent.steering;
115 }

```

Here is the call graph for this function:

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

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

5.20 wander Class Reference

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

Inheritance diagram for wander:

Collaboration diagram for wander:

Public Member Functions

- `wander ()`

default constructor.

Static Public Member Functions

- static void `loop ()`

wander scenario loop function

Additional Inherited Members

5.20.1 Detailed Description

Definition at line 14 of file wander.h.

5.20.2 Constructor & Destructor Documentation

5.20.2.1 wander()

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

default constructor.

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 }
```

5.20.3 Member Function Documentation

5.20.3.1 loop()

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

wander scenario loop function

Note

opengl callback forces that function to be static

Definition at line 15 of file wander.cpp.

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

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

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

5.21 windy Class Reference

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

Inheritance diagram for windy:

Collaboration diagram for windy:

Public Member Functions

- [windy \(\)](#)
default constructor.

Static Public Member Functions

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

Static Public Attributes

- static [flowField flow](#)
flow field used

Additional Inherited Members

5.21.1 Detailed Description

Definition at line 15 of file windy.h.

5.21.2 Constructor & Destructor Documentation

5.21.2.1 windy()

```
windy::windy ( )
```

default constructor.

Definition at line 29 of file windy.cpp.

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

5.21.3 Member Function Documentation

5.21.3.1 loop()

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

windy scenario loop function

Note

opengl callback forces that function to be static

Definition at line 17 of file windy.cpp.

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

5.21.4 Member Data Documentation

5.21.4.1 flow

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

flow field used

Note

opengl callback forces that function to be static

Definition at line 32 of file windy.h.

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

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

Chapter 6

File Documentation

6.1 include/agent.h File Reference

agent class defines all agent specifications

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

6.2 include/color.h File Reference

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

```
#include <vector>
Include dependency graph for color.h: This graph shows which files directly or indirectly include this file:
```

Classes

- class `color`

Enumerations

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

6.2.1 Detailed Description

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

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

13.05.2021

6.2.2 Enumeration Type Documentation

6.2.2.1 num

enum `num`

fundament list of al colors

Enumerator

BLACK	
BLUE	
GREEN	
CYAN	
RED	
MAGENDA	
YELLOW	
WHITE	

Definition at line 17 of file color.h.

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

6.3 include/evade.h File Reference

evade class inherited from scenario class

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

Include dependency graph for evade.h: This graph shows which files directly or indirectly include this file:

Classes

- class `evade`

6.3.1 Detailed Description

evade class inherited from scenario class

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.4 include/flee.h File Reference

agents flee from mouse scenario

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

Include dependency graph for flee.h: This graph shows which files directly or indirectly include this file:

Classes

- class [flee](#)

6.4.1 Detailed Description

agents flee from mouse scenario

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.5 include/flock.h File Reference

flocking agents scenario

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

Include dependency graph for flock.h: This graph shows which files directly or indirectly include this file:

Classes

- class [flock](#)

6.5.1 Detailed Description

flocking agents scenario

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.6 include/flowField.h File Reference

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

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

Include dependency graph for flowField.h: This graph shows which files directly or indirectly include this file:

Classes

- class [flowField](#)

Macros

- #define [FIELD_WIDTH](#) 34
- #define [FIELD_HEIGHT](#) 34
- #define [WIND_WEST](#) 0.1, 0.0
- #define [GRAVITY](#) 0.0, -0.1

6.6.1 Detailed Description

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

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

13.05.2021

6.6.2 Macro Definition Documentation

6.6.2.1 FIELD_HEIGHT

```
#define FIELD_HEIGHT 34
```

Definition at line 13 of file flowField.h.

6.6.2.2 FIELD_WIDTH

```
#define FIELD_WIDTH 34
```

Definition at line 12 of file flowField.h.

6.6.2.3 GRAVITY

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

Definition at line 16 of file flowField.h.

6.6.2.4 WIND_WEST

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

Definition at line 15 of file flowField.h.

6.7 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

6.7.1 Detailed Description

graphics class, drives openGL

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.7.2 Macro Definition Documentation

6.7.2.1 ESC

```
#define ESC 27
```

Definition at line 16 of file graphics.h.

6.7.2.2 HEIGHT

```
#define HEIGHT 34
```

Definition at line 14 of file graphics.h.

6.7.2.3 PI

```
#define PI 3.14159265
```

Definition at line 17 of file graphics.h.

6.7.2.4 WIDTH

```
#define WIDTH 34
```

Definition at line 13 of file graphics.h.

6.8 include/mouseFollower.h File Reference

agents follow mouse scenario

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

Include dependency graph for mouseFollower.h: This graph shows which files directly or indirectly include this file:

Classes

- class [mouseFollower](#)

6.8.1 Detailed Description

agents follow mouse scenario

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.9 include/obstacle.h File Reference

circular obstacles for agent avoidance behaviors

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

Include dependency graph for obstacle.h: This graph shows which files directly or indirectly include this file:

Classes

- class [obstacle](#)

6.9.1 Detailed Description

circular obstacles for agent avoidance behaviors

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

12.05.2021

6.10 include/obstacleAvoidance.h File Reference

agents avoid from obstacles scenario

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

Include dependency graph for obstacleAvoidance.h: This graph shows which files directly or indirectly include this file:

Classes

- class [obstacleAvoidance](#)

6.10.1 Detailed Description

agents avoid from obstacles scenario

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.11 include/path.h File Reference

path class used for path following steering behaviors.

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

Include dependency graph for path.h: This graph shows which files directly or indirectly include this file:

Classes

- class [path](#)

6.11.1 Detailed Description

path class used for path following steering behaviors.

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

12.05.2021

6.12 include/pathFollower.h File Reference

path following scenario

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

Include dependency graph for pathFollower.h: This graph shows which files directly or indirectly include this file:

Classes

- class [pathFollower](#)

6.12.1 Detailed Description

path following scenario

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.13 include/point.h File Reference

point class used for point operations

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

Include dependency graph for point.h: This graph shows which files directly or indirectly include this file:

Classes

- class [point](#)

6.13.1 Detailed Description

point class used for point operations

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.14 include/prison.h File Reference

agents cant escape from field scenario

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

Include dependency graph for prison.h: This graph shows which files directly or indirectly include this file:

Classes

- class [prison](#)

6.14.1 Detailed Description

agents cant escape from field scenario

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.15 include/pursuit.h File Reference

one agent pursue other one scenario

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

Include dependency graph for pursuit.h: This graph shows which files directly or indirectly include this file:

Classes

- class [pursuit](#)

6.15.1 Detailed Description

one agent pursue other one scenario

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.16 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

6.16.1 Detailed Description

pvector class used for 2D vector operations

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.16.2 Macro Definition Documentation

6.16.2.1 PI

```
#define PI 3.14159265
```

Definition at line 11 of file pvector.h.

6.17 include/random.h File Reference

utility class for random operations

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

Classes

- class [random](#)

6.17.1 Detailed Description

utility class for random operations

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.18 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](#) }

6.18.1 Detailed Description

base class for all scenarios

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.18.2 Enumeration Type Documentation

6.18.2.1 types

enum [types](#)

Enumerator

RANDOM	
STATIC	
TROOP	

Definition at line 17 of file scenario.h.

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

6.19 include/steeringBehavior.h File Reference

functions for autonomous steering behaviors

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

Include dependency graph for steeringBehavior.h: This graph shows which files directly or indirectly include this file:

Classes

- class [steeringBehavior](#)

Macros

- #define [CIRCLE_DISTANCE](#) 0.1
- #define [CIRCLE_RADIUS](#) 0.4
- #define [FOLLOW_MOUSE](#) 1
- #define [STAY_IN_FIELD](#) 2
- #define [IN_FLOW_FIELD](#) 3
- #define [AVOID_OBSTACLE](#) 4
- #define [STAY_IN_PATH](#) 5
- #define [FLOCK](#) 6
- #define [WANDER](#) 7
- #define [FLEE](#) 8
- #define [PURSUIT](#) 9
- #define [EVADE](#) 10

6.19.1 Detailed Description

functions for autonomous steering behaviors

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.19.2 Macro Definition Documentation

6.19.2.1 AVOID_OBSTACLE

```
#define AVOID_OBSTACLE 4
```

Definition at line 21 of file steeringBehavior.h.

6.19.2.2 CIRCLE_DISTANCE

```
#define CIRCLE_DISTANCE 0.1
```

Definition at line 15 of file steeringBehavior.h.

6.19.2.3 CIRCLE_RADIUS

```
#define CIRCLE_RADIUS 0.4
```

Definition at line 16 of file steeringBehavior.h.

6.19.2.4 EVADE

```
#define EVADE 10
```

Definition at line 27 of file steeringBehavior.h.

6.19.2.5 FLEE

```
#define FLEE 8
```

Definition at line 25 of file steeringBehavior.h.

6.19.2.6 FLOCK

```
#define FLOCK 6
```

Definition at line 23 of file steeringBehavior.h.

6.19.2.7 FOLLOW_MOUSE

```
#define FOLLOW_MOUSE 1
```

Definition at line 18 of file steeringBehavior.h.

6.19.2.8 IN_FLOW_FIELD

```
#define IN_FLOW_FIELD 3
```

Definition at line 20 of file steeringBehavior.h.

6.19.2.9 PURSUIT

```
#define PURSUIT 9
```

Definition at line 26 of file steeringBehavior.h.

6.19.2.10 STAY_IN_FIELD

```
#define STAY_IN_FIELD 2
```

Definition at line 19 of file steeringBehavior.h.

6.19.2.11 STAY_IN_PATH

```
#define STAY_IN_PATH 5
```

Definition at line 22 of file steeringBehavior.h.

6.19.2.12 WANDER

```
#define WANDER 7
```

Definition at line 24 of file steeringBehavior.h.

6.20 include/wander.h File Reference

random wandering agents scenario

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

Include dependency graph for wander.h: This graph shows which files directly or indirectly include this file:

Classes

- class [wander](#)

6.20.1 Detailed Description

random wandering agents scenario

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.21 include/windy.h File Reference

windy air scenario

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

Include dependency graph for windy.h: This graph shows which files directly or indirectly include this file:

Classes

- class [windy](#)

6.21.1 Detailed Description

windy air scenario

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.22 main.cpp File Reference

client code

```
#include <iostream>
#include "mouseFollower.h"
#include "prison.h"
#include "windy.h"
#include "wander.h"
#include "pursuit.h"
#include "flee.h"
#include "scenario.h"
#include "evade.h"
#include "flock.h"
#include "pathFollower.h"
#include "obstacleAvoidance.h"
Include dependency graph for main.cpp:
```

Functions

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

Variables

- int [mode](#)
specifies user selected scenario

6.22.1 Detailed Description

client code

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.22.2 Function Documentation

6.22.2.1 main()

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

main routine

Definition at line 48 of file main.cpp.

```
48 {
49     menu();
50
51     scenario* sc;
52
53     if(mode == FOLLOW_MOUSE) {
54         *sc = mouseFollower();
55     }
56     else if(mode == STAY_IN_FIELD) {
57         *sc = prison();
58     }
59     else if(mode == IN_FLOW_FIELD) {
60         *sc = windy();
61     }
62     else if(mode == WANDER) {
63         *sc = wander();
64     }
65     else if(mode == PURSUIT) {
66         *sc = pursuit();
67     }
68     else if(mode == FLEE) {
69         *sc = flee();
70     }
71     else if(mode == EVADE) {
72         *sc = evade();
73     }
74     else if(mode == FLOCK) {
75         *sc = flock();
76     }
77     else if(mode == STAY_IN_PATH) {
78         *sc = pathFollower();
79     }
80     else if(mode == AVOID_OBSTACLE) {
81         *sc = obstacleAvoidance();
82     }
83
84     sc->initGL(&argc, argv);
85
86     return 0;
87 }
```

Here is the call graph for this function:

6.22.2.2 menu()

```
void menu ( )
```

displays menu

Definition at line 31 of file main.cpp.

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

Here is the caller graph for this function:

6.22.3 Variable Documentation

6.22.3.1 mode

```
int mode
```

specifies user selected scenario

Definition at line 26 of file main.cpp.

6.23 README.md File Reference

6.24 src/agent.cpp File Reference

implementation of the agent class

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

6.24.1 Detailed Description

implementation of the agent class

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

14.05.2021

6.25 src/color.cpp File Reference

color class implementation

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

6.25.1 Detailed Description

color class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

13.05.2021

6.26 src/evade.cpp File Reference

evade class implementation

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

6.26.1 Detailed Description

evade class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.27 src/flee.cpp File Reference

flee class implementation

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


6.27.1 Detailed Description

flee class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.28 src/flock.cpp File Reference

flock class implementation

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

6.28.1 Detailed Description

flock class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.29 src/flowField.cpp File Reference

[flowField](#) class implementation

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

6.29.1 Detailed Description

[flowField](#) class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

13.05.2021

6.30 src/graphics.cpp File Reference

graphics class implementation

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

6.30.1 Detailed Description

graphics class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.31 src/mouseFollower.cpp File Reference

[mouseFollower](#) class implementation

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

6.31.1 Detailed Description

[mouseFollower](#) class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.32 src/obstacle.cpp File Reference

obstacle class implementation

```
#include "obstacle.h"
#include "graphics.h"
#include "point.h"
#include <vector>
Include dependency graph for obstacle.cpp:
```

6.32.1 Detailed Description

obstacle class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

12.05.2021

6.33 src/obstacleAvoidance.cpp File Reference

[obstacleAvoidance](#) class implementation

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

6.33.1 Detailed Description

[obstacleAvoidance](#) class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.34 src/path.cpp File Reference

path class implementation

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

6.34.1 Detailed Description

path class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

12.05.2021

6.35 src/pathFollower.cpp File Reference

[pathFollower](#) class implementation

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

6.35.1 Detailed Description

[pathFollower](#) class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.36 src/point.cpp File Reference

point class implementation file

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

6.36.1 Detailed Description

point class implementation file

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.37 src/prison.cpp File Reference

prison class implementation

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

Macros

- #define [WALL](#) 30
- #define [DISTANCE](#) 2

6.37.1 Detailed Description

prison class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.37.2 Macro Definition Documentation

6.37.2.1 DISTANCE

```
#define DISTANCE 2
```

Definition at line 14 of file prison.cpp.

6.37.2.2 WALL

```
#define WALL 30
```

Definition at line 13 of file prison.cpp.

6.38 src/pursuit.cpp File Reference

prison class implementation

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

Include dependency graph for pursuit.cpp:

6.38.1 Detailed Description

prison class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.39 src/pvector.cpp File Reference

pvector class implementation

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

Include dependency graph for pvector.cpp:

6.39.1 Detailed Description

pvector class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.40 src/random.cpp File Reference

utility class for random operations

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

6.40.1 Detailed Description

utility class for random operations

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.41 src/scenario.cpp File Reference

scenario base class implementation

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

Macros

- #define [MAX_NUMBER_OF_AGENTS](#) 50

6.41.1 Detailed Description

scenario base class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.41.2 Macro Definition Documentation

6.41.2.1 MAX_NUMBER_OF_AGENTS

```
#define MAX_NUMBER_OF_AGENTS 50
```

Definition at line 12 of file scenario.cpp.

6.42 src/steeringBehavior.cpp File Reference

implementation of autonomous steering behaviors

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

Include dependency graph for steeringBehavior.cpp:

6.42.1 Detailed Description

implementation of autonomous steering behaviors

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.43 src/wander.cpp File Reference

wander class implementation

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

Include dependency graph for wander.cpp:

6.43.1 Detailed Description

wander class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.44 src/windy.cpp File Reference

windy class implementation

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

6.44.1 Detailed Description

windy class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.45 test/test_suites.cpp File Reference

unit test suites

```
#include <boost/test/included/unit_test.hpp>
#include "../include/pvector.h"
#include "../include/point.h"
#include <iostream>
Include dependency graph for test_suites.cpp:
```

Macros

- `#define BOOST_TEST_MODULE test_suites`

Functions

- [BOOST_AUTO_TEST_CASE](#) (s1t1)
pvector magnitude test case
- [BOOST_AUTO_TEST_CASE](#) (s1t2)
pvector mul test case
- [BOOST_AUTO_TEST_CASE](#) (s1t3)
pvector div test case
- [BOOST_AUTO_TEST_CASE](#) (s1t4)
pvector dotproduct test case
- [BOOST_AUTO_TEST_CASE](#) (s1t5)
pvector angle between vectors test case
- [BOOST_AUTO_TEST_CASE](#) (s1t6)
pvector get vector angle test case
- [BOOST_AUTO_TEST_CASE](#) (s1t7)
pvector normalize test case
- [BOOST_AUTO_TEST_CASE](#) (s1t8)
pvector limit test case
- [BOOST_AUTO_TEST_CASE](#) (s1t9)
pvector overloaded operators test case
- [BOOST_AUTO_TEST_CASE](#) (s2t1)
point multiplication test case
- [BOOST_AUTO_TEST_CASE](#) (s2t2)
point division test case
- [BOOST_AUTO_TEST_CASE](#) (s2t3)
point overloaded operators test case

6.45.1 Detailed Description

unit test suites

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.45.2 Macro Definition Documentation

6.45.2.1 BOOST_TEST_MODULE

```
#define BOOST_TEST_MODULE test_suites
```

Definition at line 8 of file test_suites.cpp.

6.45.3 Function Documentation

6.45.3.1 BOOST_AUTO_TEST_CASE() [1/12]

```
BOOST_AUTO_TEST_CASE (
    slt1 )
```

pvector magnitude test case

Definition at line 22 of file test_suites.cpp.

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

Here is the call graph for this function:

6.45.3.2 BOOST_AUTO_TEST_CASE() [2/12]

```
BOOST_AUTO_TEST_CASE (
    slt2 )
```

pvector mul test case

Definition at line 33 of file test_suites.cpp.

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

Here is the call graph for this function:

6.45.3.3 BOOST_AUTO_TEST_CASE() [3/12]

```
BOOST_AUTO_TEST_CASE (
    slt3 )
```

pvector div test case

Definition at line 44 of file test_suites.cpp.

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

Here is the call graph for this function:

6.45.3.4 BOOST_AUTO_TEST_CASE() [4/12]

```
BOOST_AUTO_TEST_CASE (
    slt4 )
```

pvector dotproduct test case

Definition at line 55 of file test_suites.cpp.

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

Here is the call graph for this function:

6.45.3.5 BOOST_AUTO_TEST_CASE() [5/12]

```
BOOST_AUTO_TEST_CASE (
    slt5 )
```

pvector angle between vectors test case

Definition at line 66 of file test_suites.cpp.

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

Here is the call graph for this function:

6.45.3.6 BOOST_AUTO_TEST_CASE() [6/12]

```
BOOST_AUTO_TEST_CASE (
    slt6 )
```

pvector get vector angle test case

Definition at line 77 of file test_suites.cpp.

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

Here is the call graph for this function:

6.45.3.7 BOOST_AUTO_TEST_CASE() [7/12]

```
BOOST_AUTO_TEST_CASE (
    slt7 )
```

pvector normalize test case

Definition at line 87 of file test_suites.cpp.

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

Here is the call graph for this function:

6.45.3.8 BOOST_AUTO_TEST_CASE() [8/12]

```
BOOST_AUTO_TEST_CASE (
    slt8 )
```

pvector limit test case

Definition at line 99 of file test_suites.cpp.

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

Here is the call graph for this function:

6.45.3.9 BOOST_AUTO_TEST_CASE() [9/12]

```
BOOST_AUTO_TEST_CASE (
    slt9 )
```

pvector overloaded operators test case

Definition at line 111 of file test_suites.cpp.

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

Here is the call graph for this function:

6.45.3.10 BOOST_AUTO_TEST_CASE() [10/12]

```
BOOST_AUTO_TEST_CASE (
    s2t1 )
```

point multiplication test case

Definition at line 133 of file test_suites.cpp.

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

Here is the call graph for this function:

6.45.3.11 BOOST_AUTO_TEST_CASE() [11/12]

```
BOOST_AUTO_TEST_CASE (
    s2t2 )
```

point division test case

Definition at line 144 of file test_suites.cpp.

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

Here is the call graph for this function:

6.45.3.12 BOOST_AUTO_TEST_CASE() [12/12]

```
BOOST_AUTO_TEST_CASE (
    s2t3 )
```

point overloaded operators test case

Definition at line 155 of file test_suites.cpp.

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

Here is the call graph for this function:

Index

- ~agent
 - agent, [11](#)
- acceleration
 - agent, [12](#)
- add
 - pvector, [55](#)
- addPoint
 - path, [41](#)
- agent, [9](#)
 - ~agent, [11](#)
 - acceleration, [12](#)
 - agent, [10](#)
 - arrive, [12](#)
 - desiredVelocity, [13](#)
 - fillColor, [13](#)
 - force, [13](#)
 - id, [13](#)
 - mass, [13](#)
 - maxForce, [14](#)
 - maxSpeed, [14](#)
 - name, [14](#)
 - position, [14](#)
 - r, [14](#)
 - setFeatures, [11](#)
 - steering, [15](#)
 - targetPoint, [15](#)
 - updatePosition, [12](#)
 - velocity, [15](#)
- agents
 - scenario, [65](#)
- align
 - steeringBehavior, [68](#)
- angleBetween
 - pvector, [55](#)
- arrive
 - agent, [12](#)
- avoid
 - steeringBehavior, [69](#)
- AVOID_OBSTACLE
 - steeringBehavior.h, [94](#)
- B
 - color, [18](#)
- behavior
 - scenario, [66](#)
- BLACK
 - color.h, [82](#)
- BLUE
 - color.h, [82](#)
- BOOST_AUTO_TEST_CASE
 - test_suites.cpp, [111–114](#)
- BOOST_TEST_MODULE
 - test_suites.cpp, [110](#)
- callback
 - scenario, [66](#)
- CIRCLE_DISTANCE
 - steeringBehavior.h, [94](#)
- CIRCLE_RADIUS
 - steeringBehavior.h, [94](#)
- cohesion
 - steeringBehavior, [69](#)
- color, [15](#)
 - B, [18](#)
 - color, [16](#)
 - colors, [18](#)
 - createColors, [17](#)
 - G, [18](#)
 - getColor, [17](#)
 - R, [18](#)
- color.h
 - BLACK, [82](#)
 - BLUE, [82](#)
 - CYAN, [82](#)
 - GREEN, [82](#)
 - MAGENDA, [82](#)
 - num, [82](#)
 - RED, [82](#)
 - WHITE, [82](#)
 - YELLOW, [82](#)
- colors
 - color, [18](#)
- createAgent
 - scenario, [64](#)
- createColors
 - color, [17](#)
- createObstacle
 - obstacleAvoidance, [38](#)
- createPath
 - pathFollower, [43](#)
- createRandomArray
 - random, [62](#)
- CYAN
 - color.h, [82](#)
- desiredVelocity
 - agent, [13](#)
- DISTANCE
 - prison.cpp, [105](#)

- div
 - point, 46
 - pvector, 56
- dotProduct
 - pvector, 56
- drawAgent
 - graphics, 26
- drawCircle
 - graphics, 27
- drawLine
 - graphics, 27
- drawPath
 - graphics, 28
- drawPoint
 - graphics, 28
- drawText
 - graphics, 28
- ESC
 - graphics.h, 86
- EVADE
 - steeringBehavior.h, 94
- evade, 19
 - evade, 19
 - loop, 20
 - steeringBehavior, 70
- FIELD_HEIGHT
 - flowField.h, 84
- FIELD_WIDTH
 - flowField.h, 85
- fillColor
 - agent, 13
- FLEE
 - steeringBehavior.h, 94
- flee, 20
 - flee, 21
 - loop, 21
 - steeringBehavior, 71
- FLOCK
 - steeringBehavior.h, 94
- flock, 22
 - flock, 22
 - loop, 22
- flow
 - windy, 80
- flowField, 23
 - flowField, 24
 - getField, 24
- flowField.h
 - FIELD_HEIGHT, 84
 - FIELD_WIDTH, 85
 - GRAVITY, 85
 - WIND_WEST, 85
- FOLLOW_MOUSE
 - steeringBehavior.h, 95
- force
 - agent, 13
- forceInScreen
 - graphics, 29
- G
 - color, 18
- getAngle
 - pvector, 57
- getColor
 - color, 17
- getField
 - flowField, 24
- getMousePosition
 - graphics, 29
- getNormalPoint
 - point, 47
- graphics, 25
 - drawAgent, 26
 - drawCircle, 27
 - drawLine, 27
 - drawPath, 28
 - drawPoint, 28
 - drawText, 28
 - forceInScreen, 29
 - getMousePosition, 29
 - handleKeypress, 29
 - handleResize, 30
 - initGraphics, 30
 - mouseButton, 32
 - mouseMove, 32
 - refreshScene, 33
 - target_x, 33
 - target_y, 34
 - timerEvent, 33
- graphics.h
 - ESC, 86
 - HEIGHT, 86
 - PI, 86
 - WIDTH, 86
- GRAVITY
 - flowField.h, 85
- GREEN
 - color.h, 82
- handleKeypress
 - graphics, 29
- handleResize
 - graphics, 30
- HEIGHT
 - graphics.h, 86
- id
 - agent, 13
- IN_FLOW_FIELD
 - steeringBehavior.h, 95
- include/agent.h, 81
- include/color.h, 81
- include/evade.h, 82
- include/flee.h, 83
- include/flock.h, 83
- include/flowField.h, 84

- include/graphics.h, 85
- include/mouseFollower.h, 87
- include/obstacle.h, 87
- include/obstacleAvoidance.h, 88
- include/path.h, 88
- include/pathFollower.h, 89
- include/point.h, 89
- include/prison.h, 90
- include/pursuit.h, 90
- include/pvector.h, 91
- include/random.h, 91
- include/scenario.h, 92
- include/steeringBehavior.h, 93
- include/wander.h, 96
- include/windy.h, 96
- inFlowField
 - steeringBehavior, 71
- initGL
 - scenario, 65
- initGraphics
 - graphics, 30
- limit
 - pvector, 57
- loop
 - evade, 20
 - flee, 21
 - flock, 22
 - mouseFollower, 35
 - obstacleAvoidance, 39
 - pathFollower, 44
 - prison, 51
 - pursuit, 52
 - wander, 77
 - windy, 79
- MAGENDA
 - color.h, 82
- magnitude
 - pvector, 57
- main
 - main.cpp, 98
- main.cpp, 97
 - main, 98
 - menu, 98
 - mode, 99
- mass
 - agent, 13
- MAX_NUMBER_OF_AGENTS
 - scenario.cpp, 108
- maxForce
 - agent, 14
- maxSpeed
 - agent, 14
- menu
 - main.cpp, 98
- mode
 - main.cpp, 99
- mouseButton
 - graphics, 32
- mouseFollower, 34
 - loop, 35
 - mouseFollower, 35
- mouseMove
 - graphics, 32
- mul
 - point, 47
 - pvector, 57
- myColor
 - scenario, 66
- myPath
 - pathFollower, 44
- name
 - agent, 14
 - scenario, 66
- normalize
 - pvector, 58
- num
 - color.h, 82
- obstacle, 35
 - obstacle, 36
 - p, 37
 - r, 37
- obstacleAvoidance, 37
 - createObstacle, 38
 - loop, 39
 - obstacleAvoidance, 38
 - obstacles, 39
- obstacles
 - obstacleAvoidance, 39
- operator+
 - point, 48
 - pvector, 58, 59
- operator+=
 - pvector, 59
- operator-
 - point, 48
 - pvector, 60
- operator==
 - point, 49
 - pvector, 60
- p
 - obstacle, 37
- path, 40
 - addPoint, 41
 - path, 40, 41
 - points, 42
 - width, 42
- pathFollower, 42
 - createPath, 43
 - loop, 44
 - myPath, 44
 - pathFollower, 43
- PI
 - graphics.h, 86

- pvector.h, 91
- point, 45
 - div, 46
 - getNormalPoint, 47
 - mul, 47
 - operator+, 48
 - operator-, 48
 - operator==, 49
 - point, 45, 46
 - print, 49
 - x, 50
 - y, 50
- points
 - path, 42
- position
 - agent, 14
- print
 - point, 49
 - pvector, 61
- prison, 50
 - loop, 51
 - prison, 51
- prison.cpp
 - DISTANCE, 105
 - WALL, 105
- PURSUIT
 - steeringBehavior.h, 95
- pursuit, 52
 - loop, 52
 - pursuit, 52
 - steeringBehavior, 72
- pvector, 53
 - add, 55
 - angleBetween, 55
 - div, 56
 - dotProduct, 56
 - getAngle, 57
 - limit, 57
 - magnitude, 57
 - mul, 57
 - normalize, 58
 - operator+, 58, 59
 - operator+=, 59
 - operator-, 60
 - operator==, 60
 - print, 61
 - pvector, 54
 - x, 61
 - y, 61
- pvector.h
 - PI, 91
- R
 - color, 18
- r
 - agent, 14
 - obstacle, 37
- RANDOM
 - scenario.h, 93
- random, 62
 - createRandomArray, 62
- README.md, 99
- RED
 - color.h, 82
- refresh
 - scenario, 65
- refreshScene
 - graphics, 33
- scenario, 63
 - agents, 65
 - behavior, 66
 - callback, 66
 - createAgent, 64
 - initGL, 65
 - myColor, 66
 - name, 66
 - refresh, 65
 - scenario, 64
 - view, 67
- scenario.cpp
 - MAX_NUMBER_OF_AGENTS, 108
- scenario.h
 - RANDOM, 93
 - STATIC, 93
 - TROOP, 93
 - types, 92
- seek
 - steeringBehavior, 73
- separation
 - steeringBehavior, 73
- setAngle
 - steeringBehavior, 74
- setFeatures
 - agent, 11
- src/agent.cpp, 99
- src/color.cpp, 99
- src/evade.cpp, 100
- src/flee.cpp, 100
- src/flock.cpp, 101
- src/flowField.cpp, 101
- src/graphics.cpp, 102
- src/mouseFollower.cpp, 102
- src/obstacle.cpp, 103
- src/obstacleAvoidance.cpp, 103
- src/path.cpp, 104
- src/pathFollower.cpp, 104
- src/point.cpp, 104
- src/prison.cpp, 105
- src/pursuit.cpp, 106
- src/pvector.cpp, 106
- src/random.cpp, 107
- src/scenario.cpp, 107
- src/steeringBehavior.cpp, 108
- src/wander.cpp, 108
- src/windy.cpp, 109
- STATIC
 - scenario.h, 93

- STAY_IN_FIELD
 - steeringBehavior.h, [95](#)
- STAY_IN_PATH
 - steeringBehavior.h, [95](#)
- stayInArea
 - steeringBehavior, [74](#)
- stayInPath
 - steeringBehavior, [75](#)
- steering
 - agent, [15](#)
- steeringBehavior, [67](#)
 - align, [68](#)
 - avoid, [69](#)
 - cohesion, [69](#)
 - evade, [70](#)
 - flee, [71](#)
 - inFlowField, [71](#)
 - pursuit, [72](#)
 - seek, [73](#)
 - separation, [73](#)
 - setAngle, [74](#)
 - stayInArea, [74](#)
 - stayInPath, [75](#)
 - wander, [76](#)
- steeringBehavior.h
 - AVOID_OBSTACLE, [94](#)
 - CIRCLE_DISTANCE, [94](#)
 - CIRCLE_RADIUS, [94](#)
 - EVADE, [94](#)
 - FLEE, [94](#)
 - FLOCK, [94](#)
 - FOLLOW_MOUSE, [95](#)
 - IN_FLOW_FIELD, [95](#)
 - PURSUIT, [95](#)
 - STAY_IN_FIELD, [95](#)
 - STAY_IN_PATH, [95](#)
 - WANDER, [95](#)
- target_x
 - graphics, [33](#)
- target_y
 - graphics, [34](#)
- targetPoint
 - agent, [15](#)
- test/test_suites.cpp, [109](#)
- test_suites.cpp
 - BOOST_AUTO_TEST_CASE, [111–114](#)
 - BOOST_TEST_MODULE, [110](#)
- timerEvent
 - graphics, [33](#)
- TROOP
 - scenario.h, [93](#)
- types
 - scenario.h, [92](#)
- updatePosition
 - agent, [12](#)
- velocity
 - agent, [15](#)
- view
 - scenario, [67](#)
- WALL
 - prison.cpp, [105](#)
- WANDER
 - steeringBehavior.h, [95](#)
- wander, [76](#)
 - loop, [77](#)
 - steeringBehavior, [76](#)
 - wander, [77](#)
- WHITE
 - color.h, [82](#)
- WIDTH
 - graphics.h, [86](#)
- width
 - path, [42](#)
- WIND_WEST
 - flowField.h, [85](#)
- windy, [78](#)
 - flow, [80](#)
 - loop, [79](#)
 - windy, [79](#)
- x
 - point, [50](#)
 - pvector, [61](#)
- y
 - point, [50](#)
 - pvector, [61](#)
- YELLOW
 - color.h, [82](#)