# Autonomous Steering Agents

Generated by Doxygen 1.8.17

# Chapter 1

# Intent

1- implementing ai using autonomous steering agents

2- implementing smart agents using genetics algorithms

3- implementing smart agents using neural network

## 1.1 Dependencies

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

$sudo apt-get install libboost-all-dev

## 1.2 Resources

Jan Schifmann : Nature of Code

Fernando Bevilacqua : Understanding Steering Behaviors

Jer Thorp : Living in Data

OpenGL :

https://videotutorialsrock.com/index.php

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

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

# Chapter 2

# Todo List

**Member wander::wander ()**

business logic will be changed

# Chapter 3

# Hierarchical Index

## 3.1 Class Hierarchy

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

# Chapter 4

# Class Index

## 4.1 Class List

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

# Chapter 5

# File Index

## 5.1 File List

Here is a list of all files with brief descriptions:

# Chapter 6

# Class Documentation

## 6.1 agent Class Reference

`#include <agent.h>`

### Public Member Functions

- agent ()
  *default constructor.*
- agent (float x, float y)
  *constructor.*
- ~agent ()
  *destructor*
- void updatePosition (bool arrive)
  *position update calculations*
- void setFeatures (float s, float f, float r, float m)
  *initialize the agent attributes*
- string getName ()
  *name attribute getter*
- void setName (string n)
  *name attribute setter*
- float getMass ()
  *mass attribute getter*
- void setMass (float m)
  *mass attribute setter*

### Public Attributes

- 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*
- int id
  *id of the agent*
- bool arrive = false
  *has arriving behavior or not*

### 6.1.1 Detailed Description

Definition at line 20 of file agent.h.

### 6.1.2 Constructor & Destructor Documentation

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

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

default constructor.

**See also**

agent(float x, float y)

Definition at line 16 of file agent.cpp.

```
17 {
18
19 }
```

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

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

constructor.

**Parameters**

| | |
|---|---|
| *x* | position x of the agent |
| *y* | position y of the agent |

**See also**

agent()

Definition at line 37 of file agent.cpp.

```
38 {
39     position        = point(x, y);
40     velocity        = pvector(0.6, 0.0);
41     acceleration    = pvector(0.0, 0.0);
42     steering        = pvector(0.0, 0.0);
43     desiredVelocity = pvector(0.0, 0.0);
44     force           = pvector(0.0, 0.0);
45     targetPoint     = point(0.0, 0.0);
46     fillColor       = color(1.0, 0.0, 0.0);
47 }
```

#### 6.1.2.3 ~agent()

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

destructor

Definition at line 78 of file agent.cpp.

```
79 {
80
81 }
```

### 6.1.3 Member Function Documentation

#### 6.1.3.1 getMass()

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

mass attribute getter

Definition at line 29 of file agent.cpp.

```
29                  {
30     return mass;
31 }
```

#### 6.1.3.2 getName()

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

name attribute getter

Definition at line 21 of file agent.cpp.

```
21                          {
22     return name;
23 }
```

#### 6.1.3.3 setFeatures()

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

initialize the agent attributes

**Parameters**

| | |
|---|---|
| *s* | maximum velocity |
| *f* | maximum force |
| *r* | radius for arriving behavior |
| *m* | mass |

Definition at line 70 of file agent.cpp.

```
71 {
72     this->maxSpeed = s;
73     this->maxForce = f;
74     this->r = r;
75     this->mass = m;
76 }
```

#### 6.1.3.4 setMass()

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

mass attribute setter

**Parameters**

| | |
|---|---|
| *m* | set value |

Definition at line 33 of file agent.cpp.

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

```
35 }
```

#### 6.1.3.5 setName()

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

name attribute setter

**Parameters**

| | |
|---|---|
| *n* | set value |

Definition at line 25 of file agent.cpp.

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

#### 6.1.3.6 updatePosition()

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

position update calculations

**Parameters**

| | |
|---|---|
| *arrive* | has arriving behavior or not |

**See also**

agent()

Definition at line 49 of file agent.cpp.

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

### 6.1.4 Member Data Documentation

#### 6.1.4.1 acceleration

```
pvector agent::acceleration
```

acceleration of the agent

Definition at line 122 of file agent.h.

#### 6.1.4.2 arrive

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

has arriving behavior or not

Definition at line 143 of file agent.h.

#### 6.1.4.3 desiredVelocity

```
pvector agent::desiredVelocity
```

desired velocity of the agent

Definition at line 127 of file agent.h.

#### 6.1.4.4 fillColor

```
color agent::fillColor
```

color of the agent

Definition at line 82 of file agent.h.

#### 6.1.4.5 force

```
pvector agent::force
```

force of the agent

Definition at line 117 of file agent.h.

#### 6.1.4.6 id

```
int agent::id
```

id of the agent

Definition at line 138 of file agent.h.

#### 6.1.4.7 maxForce

```
float agent::maxForce
```

maximum force of the agent

Definition at line 107 of file agent.h.

#### 6.1.4.8 maxSpeed

```
float agent::maxSpeed
```

maximum speed of the agent

Definition at line 102 of file agent.h.

#### 6.1.4.9 position

```
point agent::position
```

position of the agent

Definition at line 87 of file agent.h.

#### 6.1.4.10 r

```
float agent::r
```

radius of the agent

Definition at line 132 of file agent.h.

#### 6.1.4.11 steering

```
pvector agent::steering
```

steering force of the apply

Definition at line 112 of file agent.h.

#### 6.1.4.12 targetPoint

```
point agent::targetPoint
```

target of the agent

Definition at line 97 of file agent.h.

#### 6.1.4.13 velocity

```
pvector agent::velocity
```

velocity of the agent

Definition at line 92 of file agent.h.

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

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

## 6.2 color Class Reference

fundament list of al colors

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

### Public Member Functions

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

## Static Public Member Functions

- static color getColor (int index)

  *gets colorbar colors*

## Public Attributes

- float R

  *portion of red color*
- float G

  *portion of green color*
- float B

  *portion of blue color*

### 6.2.1 Detailed Description

fundament list of al colors

Definition at line 28 of file color.h.

### 6.2.2 Constructor & Destructor Documentation

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

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

default constructor.

**See also**

color(float r, float g, float b)

Definition at line 13 of file color.cpp.

```
14 {
15
16 }
```

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

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

constructor.

**Parameters**

| | |
|---|---|
| *r* | red (0-255) |
| *g* | green (0-255) |
| *b* | blue (0-255) |

**See also**

path()

Definition at line 19 of file color.cpp.

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

### 6.2.3 Member Function Documentation

#### 6.2.3.1 getColor()

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

gets colorbar colors

**Parameters**

| | |
|---|---|
| *index* | color id |

Definition at line 26 of file color.cpp.

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

### 6.2.4 Member Data Documentation

#### 6.2.4.1 B

`float color::B`

portion of blue color

Definition at line 58 of file color.h.

#### 6.2.4.2 G

`float color::G`

portion of green color

Definition at line 53 of file color.h.

#### 6.2.4.3 R

`float color::R`

portion of red color

Definition at line 48 of file color.h.

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

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

## 6.3 evade Class Reference

`#include <evade.h>`

### Public Member Functions

- evade ()
  *default constructor.*

### Static Public Member Functions

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

## Additional Inherited Members

### 6.3.1 Detailed Description

Definition at line 15 of file evade.h.

### 6.3.2 Constructor & Destructor Documentation

#### 6.3.2.1 evade()

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

default constructor.

Definition at line 31 of file evade.cpp.

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

### 6.3.3 Member Function Documentation

#### 6.3.3.1 loop()

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

loop function of evading scenario

**Note**

opengl callback forces that function to be static

Definition at line 15 of file evade.cpp.

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

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

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

## 6.4 flee Class Reference

`#include <flee.h>`

### Public Member Functions

- flee ()
  *default constructor.*

### Static Public Member Functions

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

### Additional Inherited Members

### 6.4.1 Detailed Description

Definition at line 14 of file flee.h.

### 6.4.2 Constructor & Destructor Documentation

#### 6.4.2.1 flee()

`flee::flee ( )`

default constructor.

Definition at line 24 of file flee.cpp.

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

### 6.4.3 Member Function Documentation

#### 6.4.3.1 loop()

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

evading scenario loop function

**Note**

opengl callback forces that function to be static

Definition at line 15 of file flee.cpp.

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

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

- include/flee.h
- src/flee.cpp

## 6.5 flock Class Reference

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

### Public Member Functions

- flock ()
  *default constructor.*

### Static Public Member Functions

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

### Additional Inherited Members

### 6.5.1 Detailed Description

Definition at line 15 of file flock.h.

### 6.5.2 Constructor & Destructor Documentation

#### 6.5.2.1 flock()

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

default constructor.

Definition at line 36 of file flock.cpp.

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

### 6.5.3 Member Function Documentation

#### 6.5.3.1 loop()

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

flocking scenario loop function

**Note**

opengl callback forces that function to be static

Definition at line 15 of file flock.cpp.

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

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

- include/flock.h
- src/flock.cpp

## 6.6 flowField Class Reference

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

## Public Member Functions

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

### 6.6.1 Detailed Description

Definition at line 18 of file flowField.h.

### 6.6.2 Constructor & Destructor Documentation

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

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

default constructor.

**See also**

flowField(pvector p)

Definition at line 15 of file flowField.cpp.

```
16 {
17
18 }
```

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

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

constructor.

**Parameters**

| | |
|---|---|
| *p* | force vector |

**See also**

flowField()

Definition at line 10 of file flowField.cpp.

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

### 6.6.3 Member Function Documentation

#### 6.6.3.1 getField()

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

get force at individual pixel

**Parameters**

| | |
|---|---|
| *x* | coordinate |
| *y* | coordinate |

**Returns**

force at specified position

Definition at line 39 of file flowField.cpp.

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

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

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

## 6.7 graphics Class Reference

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

### Public Member Functions

- void drawAgent (agent &agent)

  *drawing with corresponding angle*
- void drawLine (point p1, point p2, color cl)

  *drawing line*
- void drawPath (path &path)

  *draws path*

- void drawPoint (point p)
  *draws point*
- void drawCircle (point p, float radius)
  *draws circle*
- void drawText (string text, point p)
  *draws text on screen*
- void forceInScreen (agent &agent)
  *changes agent position so that it stays in screen*
- void refreshScene ()
  *update agent position*
- point getMousePosition ()
  *gets mouse position*
- void initGraphics (int *argv, char **argc, void(*callback)())
  *initialization of graphics*

## Static Public Member Functions

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

## Static Public Attributes

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

### 6.7.1 Detailed Description

Definition at line 22 of file graphics.h.

### 6.7.2 Member Function Documentation

#### 6.7.2.1 drawAgent()

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

drawing with corresponding angle

**Parameters**

| | |
|---|---|
| *agent* | instance to change |

Definition at line 158 of file graphics.cpp.

```
159 {
160     glPushMatrix();
161     glTranslatef(agent.position.x, agent.position.y, 0.0f);
162     glRotatef(agent.velocity.getAngle(), 0.0f, 0.0f, 1.0f);
163     glBegin(GL_TRIANGLES);
164     glColor3f( agent.fillColor.R, agent.fillColor.G, agent.fillColor.B);
165     glVertex3f( 1.0f,  0.0f, 0.0f);
166     glVertex3f(-1.0f,  0.5f, 0.0f);
167     glVertex3f(-1.0f, -0.5f, 0.0f);
168     glEnd();
169     glPopMatrix();
170 }
```

#### 6.7.2.2 drawCircle()

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

draws circle

**Parameters**

| | |
|---|---|
| *p* | center of the circle |
| *radius* | radius of the circle |

Definition at line 136 of file graphics.cpp.

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

#### 6.7.2.3 drawLine()

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

drawing line

**Parameters**

| | |
|---|---|
| *p1* | start point of the line |
| *p2* | end point of the line |
| *color* | of the line |

Definition at line 126 of file graphics.cpp.

```
127 {
128     glColor3f( cl.R, cl.G, cl.B);
129     glLineWidth(2);
130     glBegin(GL_LINES);
131     glVertex2f(p1.x, p1.y);
132     glVertex2f(p2.x, p2.y);
133     glEnd();
134 }
```

#### 6.7.2.4 drawPath()

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

draws path

**Parameters**

| | |
|---|---|
| *path* | to draw |

Definition at line 112 of file graphics.cpp.

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

#### 6.7.2.5 drawPoint()

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

draws point

**Parameters**

| | |
|---|---|
| *p* | point to draw |

Definition at line 149 of file graphics.cpp.

```
150 {
151     glColor3f(1,1,1);
152     glPointSize(4.0);
153     glBegin(GL_POINTS);
154     glVertex2f(p.x, p.y);
155     glEnd();
156 }
```

#### 6.7.2.6 drawText()

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

draws text on screen

**Parameters**

| | |
|---|---|
| *p* | position of the text |
| *text* | to display |

Definition at line 21 of file graphics.cpp.

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

#### 6.7.2.7 forceInScreen()

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

changes agent position so that it stays in screen

**Parameters**

| | |
|---|---|
| *agent* | instance |

Definition at line 61 of file graphics.cpp.

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

#### 6.7.2.8 getMousePosition()

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

gets mouse position

**Returns**

mouse point

Definition at line 56 of file graphics.cpp.

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

#### 6.7.2.9 handleKeypress()

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

key press event

**Parameters**

| | |
|---|---|
| *key* | pressed |
| *x* | unused but required for openGL |
| *y* | unused but required for openGL |

Definition at line 105 of file graphics.cpp.

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

#### 6.7.2.10 handleResize()

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

event triggered with screen resizing

**Parameters**

| | |
|---|---|
| *w* | width of the screen |
| *h* | height of the screen |

Definition at line 81 of file graphics.cpp.

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

#### 6.7.2.11 initGraphics()

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

initialization of graphics

**Parameters**

| | |
|---|---|
| *argv* | user parameters |
| *argc* | count of user parameters |
| *callback* | loop function for openGL periodic callback |

Definition at line 39 of file graphics.cpp.

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

#### 6.7.2.12 mouseButton()

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

mouse press event

**Parameters**

| | |
|---|---|
| *button* | mouse key pressed |
| *state* | down/up etc. |
| *x* | unused but required for openGL |
| *y* | unused but required for openGL |

Definition at line 99 of file graphics.cpp.

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

#### 6.7.2.13 mouseMove()

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

event triggered with mouse movements

**Parameters**

| | |
|---|---|
| *x* | osition of the mouse |
| *y* | position of the mouse |

Definition at line 73 of file graphics.cpp.

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

#### 6.7.2.14 refreshScene()

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

update agent position

Definition at line 30 of file graphics.cpp.

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

#### 6.7.2.15 timerEvent()

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

periodic timer event

**Parameters**

| | |
|---|---|
| *value* | period as ms |

Definition at line 93 of file graphics.cpp.

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

### 6.7.3 Member Data Documentation

#### 6.7.3.1 target_x

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

mouse position x

Definition at line 129 of file graphics.h.

#### 6.7.3.2 target_y

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

mouse position y

Definition at line 134 of file graphics.h.

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

- include/graphics.h
- src/graphics.cpp

## 6.8 mouseFollower Class Reference

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

### Public Member Functions

- mouseFollower ()

  *default constructor.*

## Static Public Member Functions

- static void loop ()

  *mouse following scenario loop function*

## Additional Inherited Members

### 6.8.1 Detailed Description

Definition at line 14 of file mouseFollower.h.

### 6.8.2 Constructor & Destructor Documentation

#### 6.8.2.1 mouseFollower()

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

default constructor.

Definition at line 25 of file mouseFollower.cpp.

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

### 6.8.3 Member Function Documentation

#### 6.8.3.1 loop()

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

mouse following scenario loop function

**Note**

opengl callback forces that function to be static

Definition at line 15 of file mouseFollower.cpp.

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

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

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

## 6.9 obstacle Class Reference

`#include <obstacle.h>`

### 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*
- color perimeterColor
  *obstacle color*

### 6.9.1 Detailed Description

Definition at line 13 of file obstacle.h.

### 6.9.2 Constructor & Destructor Documentation

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

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

default constructor.

**See also**

obstacle(point p, float r

Definition at line 15 of file obstacle.cpp.

```
16 {
17     p = point(0,0);
18     r = 5;
19     perimeterColor = RED;
20 }
```

#### 6.9.2.2 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 22 of file obstacle.cpp.

```
23 {
24     this->p = p;
25     this->r = r;
26     perimeterColor = RED;
27 }
```

### 6.9.3 Member Data Documentation

#### 6.9.3.1 p

```
point obstacle::p
```

center point of the obstacle

Definition at line 32 of file obstacle.h.

#### 6.9.3.2 perimeterColor

```
color obstacle::perimeterColor
```

obstacle color

Definition at line 42 of file obstacle.h.

#### 6.9.3.3 r

```
float obstacle::r
```

radius of the obstacle

Definition at line 37 of file obstacle.h.

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

- include/obstacle.h
- src/obstacle.cpp

# 6.10 obstacleAvoidance Class Reference

`#include <obstacleAvoidance.h>`

## Public Member Functions

- obstacleAvoidance ()
  *default constructor.*

## Static Public Member Functions

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

## Static Public Attributes

- static vector< obstacle > obstacles
  *list of obstacles*

## Additional Inherited Members

### 6.10.1 Detailed Description

Definition at line 15 of file obstacleAvoidance.h.

### 6.10.2 Constructor & Destructor Documentation

#### 6.10.2.1 obstacleAvoidance()

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

default constructor.

Definition at line 45 of file obstacleAvoidance.cpp.

```
46 {
47     name = "avoid obstacles";
48     createAgent(STATIC, nullptr, nullptr, nullptr);
49     createObstacle(obstacles);
50     callback = reinterpret_cast <void(*)()> ( (void *)(&loop) );
51 }
```

### 6.10.3 Member Function Documentation

#### 6.10.3.1 createObstacle()

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

creation of list of obstacles

**Parameters**

| | |
|---|---|
| *obstacles* | list to be created |

**Note**

opengl callback forces that function to be static

Definition at line 38 of file obstacleAvoidance.cpp.

```
39 {
40     obstacles.push_back(obstacle(point(0,0), 8));
41     obstacles.push_back(obstacle(point(-20,0), 3));
42     obstacles.push_back(obstacle(point(20,-10), 4));
43 }
```

#### 6.10.3.2 loop()

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

obstacle avoidance scenario loop function

**Note**

opengl callback forces that function to be static

Definition at line 17 of file obstacleAvoidance.cpp.

```
18 {
19     for(auto it = agents.begin(); it < agents.end(); it++){
20         (*it).targetPoint = view.getMousePosition();
21         pvector seek  = behavior.seek(*it);
22         seek.mul(0.5);
23 
24         pvector avoid = behavior.avoid(obstacles, *it);
25         (*it).force = avoid + seek;
26         (*it).arrive = true;
27 
28         //
29         for(auto it = obstacles.begin(); it < obstacles.end(); it++){
30             point p = (*it).p;
31             view.drawCircle(p, (*it).r);
32         }
33 
34     }
35     refresh();
36 }
```

### 6.10.4 Member Data Documentation

#### 6.10.4.1 obstacles

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

list of obstacles

**Note**

opengl callback forces that function to be static

Definition at line 32 of file obstacleAvoidance.h.

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

- include/obstacleAvoidance.h
- src/obstacleAvoidance.cpp

# 6.11 path Class Reference

`#include <path.h>`

## 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*
- color borderColor
  *path color*

### 6.11.1 Detailed Description

Definition at line 16 of file path.h.

### 6.11.2 Constructor & Destructor Documentation

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

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

default constructor.

**See also**

path(float width)

Definition at line 16 of file path.cpp.

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

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

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

donstructor.

**Parameters**

| | |
|---|---|
| *width* | The width of the path. |

**See also**

path()

Definition at line 22 of file path.cpp.

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

## 6.11.3 Member Function Documentation

### 6.11.3.1 addPoint()

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

adds a new point to the path

**Parameters**

| | |
|---|---|
| *point* | to add to the path |

Definition at line 11 of file path.cpp.

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

## 6.11.4 Member Data Documentation

### 6.11.4.1 borderColor

```
color path::borderColor
```

path color

Definition at line 50 of file path.h.

#### 6.11.4.2 points

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

list of points added to the path

Definition at line 40 of file path.h.

#### 6.11.4.3 width

```
int path::width
```

width of the path

Definition at line 45 of file path.h.

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

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

## 6.12 pathFollower Class Reference

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

### Public Member Functions

- pathFollower ()
  *default constructor.*

### Static Public Member Functions

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

### Static Public Attributes

- static path myPath
  *path that will be followed*

## Additional Inherited Members

### 6.12.1 Detailed Description

Definition at line 14 of file pathFollower.h.

### 6.12.2 Constructor & Destructor Documentation

#### 6.12.2.1 pathFollower()

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

default constructor.

Definition at line 39 of file pathFollower.cpp.

```
40 {
41     int agentCount = 40;
42     float maxForce = 0.2;
43     float maxSpeed = 0.4;
44     myPath = path(8);
45     createPath(myPath);
46     name = "path following";
47     createAgent(RANDOM, &agentCount, &maxForce, &maxSpeed);
48     callback = reinterpret_cast <void(*)()> ( (void *)(&loop) );
49 }
```

### 6.12.3 Member Function Documentation

#### 6.12.3.1 createPath()

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

creates path

**Parameters**

| | |
|---|---|
| *path* | to create |

**Note**

opengl callback forces that function to be static

Definition at line 31 of file pathFollower.cpp.

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

```
35    p.addPoint(point( 10,  7));
36    p.addPoint(point( 40, 12));
37 }
```

#### 6.12.3.2 loop()

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

path follower scenario loop function

**Note**

opengl callback forces that function to be static

Definition at line 17 of file pathFollower.cpp.

```
18 {
19     for(auto it = agents.begin(); it < agents.end(); it++){
20         pvector flwpth = behavior.stayInPath(*it, myPath, view);
21         pvector sep  = behavior.separation(agents, *it);
22         sep.mul(5);
23         (*it).force = sep + flwpth;
24 
25         //
26         view.drawPath(myPath);
27     }
28     refresh();
29 }
```

### 6.12.4 Member Data Documentation

#### 6.12.4.1 myPath

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

path that will be followed

**Note**

opengl callback forces that function to be static

Definition at line 38 of file pathFollower.h.

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

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

## 6.13 point Class Reference

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

## Public Member Functions

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

## Public Attributes

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

### 6.13.1 Detailed Description

Definition at line 15 of file point.h.

### 6.13.2 Constructor & Destructor Documentation

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

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

default constructor

**See also**

point(float x, float y)

Definition at line 21 of file point.cpp.

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

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

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

constructor

**Parameters**

| | |
|---|---|
| *x* | position x of the point |
| *y* | position y of the point |

**See also**

point()

Definition at line 15 of file point.cpp.

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

### 6.13.3 Member Function Documentation

#### 6.13.3.1 div()

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

divide point

**Parameters**

| | |
|---|---|
| *d* | scalar to divide position of the point |

Definition at line 42 of file point.cpp.

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

#### 6.13.3.2 getNormalPoint()

```
void point::getNormalPoint (
            point predicted,
```

```
            point start,
            point end )
```

provides normal point on a vector of a point

**Parameters**

| | |
|---|---|
| *predicted* | point that caller require normal on the vector |
| *start* | point of the vector |
| *end* | point of the vector |

Definition at line 71 of file point.cpp.

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

#### 6.13.3.3 mul()

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

multiply point

**Parameters**

| | |
|---|---|
| *d* | scalar to multiply position of the point |

Definition at line 48 of file point.cpp.

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

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

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

overloaded + operator

**Parameters**

| | |
|---|---|
| *obj* | point to add |

**Returns**

sum

Definition at line 55 of file point.cpp.

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

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

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

overloaded + operator

**Parameters**

| | |
|---|---|
| *obj* | vector to add |

**Returns**

sum

Definition at line 27 of file point.cpp.

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

#### 6.13.3.6 operator-()

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

overloaded - operator

**Parameters**

| | |
|---|---|
| *obj* | point to substract |

**Returns**

difference

Definition at line 63 of file point.cpp.

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

#### 6.13.3.7 operator==()

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

overloaded == operator

**Parameters**

| | |
|---|---|
| *obj* | point to compare |

**Returns**

comparison result

Definition at line 35 of file point.cpp.

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

#### 6.13.3.8 print()

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

debug function

**Parameters**

| | |
|---|---|
| *s* | explanation string of the log |

Definition at line 83 of file point.cpp.

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

### 6.13.4 Member Data Documentation

#### 6.13.4.1 x

```
float point::x
```

x position

Definition at line 88 of file point.h.

#### 6.13.4.2 y

```
float point::y
```

y position

Definition at line 93 of file point.h.

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

- include/point.h
- src/point.cpp

## 6.14 prison Class Reference

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

### Public Member Functions

- prison ()
  *default constructor.*

### Static Public Member Functions

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

### Additional Inherited Members

### 6.14.1 Detailed Description

Definition at line 15 of file prison.h.

### 6.14.2 Constructor & Destructor Documentation

#### 6.14.2.1 prison()

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

default constructor.

Definition at line 31 of file prison.cpp.

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

### 6.14.3 Member Function Documentation

#### 6.14.3.1 loop()

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

prisoning scenario loop function

prison loop function

**Note**

opengl callback forces that function to be static

Definition at line 18 of file prison.cpp.

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

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

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

## 6.15 pursuit Class Reference

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

## Public Member Functions

- pursuit ()

  *default constructor.*

## Static Public Member Functions

- static void loop ()

  *pursuing scenario loop function*

## Additional Inherited Members

### 6.15.1 Detailed Description

Definition at line 14 of file pursuit.h.

### 6.15.2 Constructor & Destructor Documentation

#### 6.15.2.1 pursuit()

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

default constructor.

Definition at line 31 of file pursuit.cpp.

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

### 6.15.3 Member Function Documentation

#### 6.15.3.1 loop()

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

pursuing scenario loop function

**Note**

opengl callback forces that function to be static

Definition at line 15 of file pursuit.cpp.

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

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

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

## 6.16 pvector Class Reference

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

### Public Member Functions

- pvector ()
  *default constructor*
- pvector (float x, float y)
  *constructor*
- float magnitude ()
  *calculates magnitude of the vector*
- pvector & normalize ()
  *normalize*
- void div (float i)
  *vector division*
- void mul (float i)
  *vector multiplication*
- void add (pvector p)
  *addition of vectors*
- void limit (float limit)
  *vector limitation*
- float getAngle ()

*calculates vector angle*

- float dotProduct (pvector v)

  *dot product of two vectors*
- float angleBetween (pvector v)

  *angle calculation between two vectors*
- void print (const string &s)

  *debug function*
- pvector operator+= (pvector const &obj)

  *overloaded += operator*
- pvector operator+ (pvector const &obj)

  *overloaded + operator*
- pvector operator- (pvector const &obj)

  *overloaded - operator*
- pvector operator- (point const &obj)

  *overloaded - operator*
- pvector operator+ (point const &obj)

  *overloaded + operator*
- bool operator== (pvector const &obj)

  *overloaded == operator*

## Public Attributes

- float x

  *x magnitude of the vector*
- float y

  *y magnitude of the vector*

### 6.16.1 Detailed Description

Definition at line 17 of file pvector.h.

### 6.16.2 Constructor & Destructor Documentation

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

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

default constructor

**See also**

pvector(float x, float y)

Definition at line 35 of file pvector.cpp.

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

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

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

constructor

**Parameters**

| | |
|---|---|
| *x* | magnitude of the vector |
| *y* | magnitude of the vector |

**See also**

pvector()

Definition at line 41 of file pvector.cpp.

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

### 6.16.3 Member Function Documentation

#### 6.16.3.1 add()

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

addition of vectors

**Parameters**

| | |
|---|---|
| *p* | vector to add |

Definition at line 59 of file pvector.cpp.

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

#### 6.16.3.2 angleBetween()

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

angle calculation between two vectors

**Parameters**

| | |
|---|---|
| *v* | vector to calculate angle |

**Returns**

angle

Definition at line 23 of file pvector.cpp.

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

#### 6.16.3.3 div()

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

vector division

**Parameters**

| | |
|---|---|
| *i* | scalar value to divide |

Definition at line 47 of file pvector.cpp.

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

#### 6.16.3.4 dotProduct()

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

dot product of two vectors

**Parameters**

| | |
|---|---|
| *v* | vector to calculate dot product |

**Returns**

returns scalar dot product

Definition at line 30 of file pvector.cpp.

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

#### 6.16.3.5 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 }
```

#### 6.16.3.6 limit()

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

vector limitation

**Parameters**

| | |
|---|---|
| *limit* | value to restrict vector magnitude |

Definition at line 84 of file pvector.cpp.

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

#### 6.16.3.7 magnitude()

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

calculates magnitude of the vector

**Returns**

magnitude of the vector

Definition at line 65 of file pvector.cpp.

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

#### 6.16.3.8 mul()

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

vector multiplication

**Parameters**

| | |
|---|---|
| *i* | scalar value to multiply |

Definition at line 53 of file pvector.cpp.

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

#### 6.16.3.9 normalize()

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

normalize

**Returns**

normalized vector

Definition at line 70 of file pvector.cpp.

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

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

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

overloaded + operator

**Parameters**

| | |
|---|---|
| *obj* | point to add |

**Returns**

sum

Definition at line 112 of file pvector.cpp.

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

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

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

overloaded + operator

**Parameters**

| | |
|---|---|
| *obj* | vector to add |

**Returns**

sum

Definition at line 90 of file pvector.cpp.

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

#### 6.16.3.12 operator+=()

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

overloaded += operator

**Parameters**

| | |
|---|---|
| *obj* | vector to add |

**Returns**

sum

Definition at line 98 of file pvector.cpp.

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

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

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

overloaded - operator

**Parameters**

| | |
|---|---|
| *obj* | point to substract |

**Returns**

difference

Definition at line 120 of file pvector.cpp.

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

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

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

overloaded - operator

**Parameters**

| | |
|---|---|
| *obj* | vector to substract |

**Returns**

difference

Definition at line 133 of file pvector.cpp.

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

#### 6.16.3.15 operator==()

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

overloaded == operator

**Parameters**

| | |
|---|---|
| *obj* | vector to check if equal |

**Returns**

comparison result

Definition at line 105 of file pvector.cpp.

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

#### 6.16.3.16 print()

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

debug function

**Parameters**

| | |
|---|---|
| *s* | identification text |

Definition at line 128 of file pvector.cpp.

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

### 6.16.4 Member Data Documentation

#### 6.16.4.1 x

```
float pvector::x
```

x magnitude of the vector

Definition at line 140 of file pvector.h.

#### 6.16.4.2 y

```
float pvector::y
```

y magnitude of the vector

Definition at line 145 of file pvector.h.

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

- include/pvector.h
- src/pvector.cpp

## 6.17 random Class Reference

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

### Static Public Member Functions

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

### 6.17.1 Detailed Description

Definition at line 9 of file random.h.

### 6.17.2 Member Function Documentation

#### 6.17.2.1 createRandomArray()

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

random array generation

**Parameters**

| | |
|---|---|
| *arr* | struct that includes random values |
| *size* | of the array |

Definition at line 14 of file random.cpp.

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

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

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

## 6.18 scenario Class Reference

`#include <scenario.h>`

### Public Member Functions

- scenario ()

  *default constructor.*
- void createAgent (int type, int *count, float *force, float *speed)

  *agent creation*
- void initGL (int *argv, char **argc)

  *graphics initialization*

### Static Public Member Functions

- static void refresh ()

  *refreshes all items*

### Public Attributes

- void(* callback )()

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

### Static Public Attributes

- static vector< agent > agents

  *structure stores agents*
- static graphics view

  *graphics instance used*
- static steeringBehavior behavior

  *behavior instance used*
- static string name

  *scenario name*

### 6.18.1 Detailed Description

Definition at line 19 of file scenario.h.

### 6.18.2 Constructor & Destructor Documentation

#### 6.18.2.1 scenario()

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

default constructor.

Definition at line 27 of file scenario.cpp.

```
28 {
29     srand(time(NULL));
30     view = graphics();
31 }
```

### 6.18.3 Member Function Documentation

#### 6.18.3.1 createAgent()

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

agent creation

**Parameters**

| | |
|---|---|
| *type* | type of creation method |
| *count* | number of agents to be created |
| *force* | max force of agents to be created |
| *speed* | max speed of agents to be created |

Definition at line 108 of file scenario.cpp.

```
109 {
110     if(type == TROOP){
111         createTroop(*count);
112     }
113     else if(type == RANDOM){
114         createRandomAgents(*count, *force, *speed);
115     }
116     else if(type == STATIC){
117         createStaticAgents();
118     }
```

```
119     else{
120         //error message
121     }
122 }
```

#### 6.18.3.2 initGL()

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

graphics initialization

**Parameters**

| | |
|---|---|
| *argv* | list of user arguments |
| *argc* | number of user arguments |

Definition at line 21 of file scenario.cpp.

```
22 {
23     view.initGraphics(argc, argv, callback);
24 }
```

#### 6.18.3.3 refresh()

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

refreshes all items

**Note**

opengl callback forces that function to be static

Definition at line 33 of file scenario.cpp.

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

### 6.18.4 Member Data Documentation

#### 6.18.4.1 agents

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

structure stores agents

**Note**

opengl callback forces that function to be static

Definition at line 52 of file scenario.h.

#### 6.18.4.2 behavior

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

behavior instance used

**Note**

opengl callback forces that function to be static

Definition at line 64 of file scenario.h.

#### 6.18.4.3 callback

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

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

Definition at line 75 of file scenario.h.

#### 6.18.4.4 name

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

scenario name

**Note**

opengl callback forces that function to be static

Definition at line 70 of file scenario.h.

#### 6.18.4.5 view

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

graphics instance used

**Note**

opengl callback forces that function to be static

Definition at line 58 of file scenario.h.

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

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

## 6.19 steeringBehavior Class Reference

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

### Public Member Functions

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

### 6.19.1 Detailed Description

Definition at line 35 of file steeringBehavior.h.

### 6.19.2 Member Function Documentation

#### 6.19.2.1 align()

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

force to align

**Parameters**

| | |
|---|---|
| *agent* | to be aligned |
| *boids* | list of all the agents |

**Returns**

force to be applied

Definition at line 119 of file steeringBehavior.cpp.

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

#### 6.19.2.2 avoid()

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

force to avoid

**Parameters**

| | |
|---|---|
| *agent* | agent that will avoid from obstacles |
| *obstacles* | list of all existing objects |

**Returns**

force to be applied

Definition at line 183 of file steeringBehavior.cpp.

```
184 {
185    float dynamic_length = agent.velocity.magnitude() / agent.maxSpeed;
186    pvector vel = agent.velocity;
187    vel.normalize().mul(dynamic_length);
188    pvector ahead  = vel + agent.position;
189    vel.mul(6);
190    pvector ahead2 = vel + agent.position;
191    //view.drawPoint(point(ahead.x, ahead.y));
192    //view.drawPoint(point(ahead2.x, ahead2.y));
193
194    for(auto it = obstacles.begin(); it < obstacles.end(); it++){
195       float dist  = (ahead  - (*it).p).magnitude();
196       float dist2 = (ahead2 - (*it).p).magnitude();
197       if(dist < (*it).r + 2 || dist2 < (*it).r + 2){
198          pvector avoidance = ahead - (*it).p;
199          avoidance.normalize().mul(20);
200          /*a = point(avoidance.x, avoidance.y);
201          view.drawLine(agent.position, agent.position + a, color(0,1,0));*/
202          return avoidance;
203       }
204    }
205    return pvector(0,0);
206 }
```

### 6.19.2.3 cohesion()

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

force to cohesion

**Parameters**

| | |
|---|---|
| *agent* | to go to center of other agents, with specified distance |
| *boids* | list of all the agents |

**Returns**

force to be applied

Definition at line 140 of file steeringBehavior.cpp.

```
141 {
142    float neighborDist = 20;
143    point sum {0,0};
144    int count = 0;
145    for(auto it = boids.begin(); it < boids.end(); it++){
146       float d = (agent.position - (*it).position).magnitude();
147       if( (d >0) && (d < neighborDist) ){
```

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

#### 6.19.2.4 evade()

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

force to evade

**Parameters**

| | |
|---|---|
| *evader* | agent that will escape |
| *view* | used for debugging |
| *boids* | list of all the agents |
| *name* | other agent to evade |

**Returns**

force to be applied

Definition at line 47 of file steeringBehavior.cpp.

```
48 {
49     agent target;
50     for(auto it = boids.begin(); it < boids.end(); it++){
51        if((*it).getName() == name){
52           target = *it;
53        }
54     }
55
56     point p = point(evader.position.x + 2, evader.position.y - 2);
57     view.drawText(evader.getName(), p);
58     p = point(target.position.x + 2, target.position.y - 2);
59     view.drawText(target.getName(), p);
60
61     pvector targetVel = target.velocity;
62     targetVel.mul(5);//TODO: magic number
63
64     point futurePos = target.position + targetVel;
65     view.drawPoint(futurePos);
66
67     pvector dist = evader.position - futurePos;
68     dist.normalize().mul( 1 / dist.magnitude() );
69
70     evader.targetPoint = evader.position + dist;
71     return flee(evader, view, futurePos);
72 }
```

#### 6.19.2.5 flee()

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

force to flee

**Parameters**

| | |
|---|---|
| *agent* | agent that will flee |
| *view* | used for debugging |
| *p* | point that agent flees |

**Returns**

force to be applied

Definition at line 28 of file steeringBehavior.cpp.

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

#### 6.19.2.6 inFlowField()

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

gets flow field force

**Parameters**

| | |
|---|---|
| *agent* | unit to apply flow field |
| *flow* | field |

**Returns**

force to be applied

Definition at line 238 of file steeringBehavior.cpp.

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

#### 6.19.2.7 pursuit()

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

force to pursue

**Parameters**

| | |
|---|---|
| *pursuer* | agent that will follow specified agent |
| *view* | used for debugging |
| *boids* | list of all the agents |
| *name* | other agent to pursue |

**Returns**

force to be applied

Definition at line 74 of file steeringBehavior.cpp.

```
75 {
76    agent target;
77    for(auto it = boids.begin(); it < boids.end(); it++){
78       if((*it).getName() == name){
79          target = *it;
80       }
81    }
82
83    point p = point(target.position.x + 2, target.position.y - 2);
84    view.drawText(target.getName(), p);
85    p = point(pursuer.position.x + 2, pursuer.position.y - 2);
86    view.drawText(pursuer.getName(), p);
87
88    float dist = (target.position - pursuer.position).magnitude();
89    float t = dist / target.maxSpeed;
90
91    pvector targetVel = target.velocity;
92    targetVel.mul(t);
93    point futurePos = target.position + targetVel;
94    pursuer.targetPoint = futurePos;
95    return seek(pursuer);
96 }
```

#### 6.19.2.8 seek()

```
pvector steeringBehavior::seek (
            agent & agent )
```

force to seek

**Parameters**

| | |
|---|---|
| *agent* | that will go to specific target point |

**Returns**

force to be applied

Definition at line 208 of file steeringBehavior.cpp.

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

#### 6.19.2.9 separation()

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

force to separate

**Parameters**

| | |
|---|---|
| *agent* | agent that will be stayed away |
| *agents* | list of all the agents |

**Returns**

force to be applied

Definition at line 160 of file steeringBehavior.cpp.

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

#### 6.19.2.10 setAngle()

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

applies angle on vector

**Parameters**

| | |
|---|---|
| *angle* | that will be set |
| *p* | vector that angle will be applied |

Definition at line 22 of file steeringBehavior.cpp.

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

#### 6.19.2.11 stayInArea()

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

gets reflection force

**Parameters**

| | |
|---|---|
| *agent* | unit to check |
| *turnpoint* | defines border to apply force |

**Returns**

force to be applied

Definition at line 247 of file steeringBehavior.cpp.

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

```
269     return pvector(0,0);
270 }
```

#### 6.19.2.12 stayInPath()

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

gets force to follow path

**Parameters**

| | |
|---|---|
| *agent* | to follow the pathk |
| *path* | to follow |
| *view* | used for debugging |

**Returns**

force to be applied

Definition at line 215 of file steeringBehavior.cpp.

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

#### 6.19.2.13 wander()

```
pvector steeringBehavior::wander (
             agent & agent )
```

force to wander

**Parameters**

| | |
|---|---|
| *agent* | agent that will wander |

**Returns**

force to be applied

Definition at line 98 of file steeringBehavior.cpp.

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

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

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

## 6.20 wander Class Reference

`#include <wander.h>`

### Public Member Functions

- wander ()

  *default constructor*

### Static Public Member Functions

- static void loop ()

  *wander scenario loop function*

### Additional Inherited Members

### 6.20.1 Detailed Description

Definition at line 14 of file wander.h.

### 6.20.2 Constructor & Destructor Documentation

#### 6.20.2.1 wander()

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

default constructor

**Todo** business logic will be changed

Definition at line 24 of file wander.cpp.

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

### 6.20.3 Member Function Documentation

#### 6.20.3.1 loop()

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

wander scenario loop function

**Note**

opengl callback forces that function to be static

Definition at line 15 of file wander.cpp.

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

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

- include/wander.h
- src/wander.cpp

## 6.21 windy Class Reference

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

## Public Member Functions

- windy ()

  *default constructor.*

## Static Public Member Functions

- static void loop ()

  *windy scenario loop function*

## Static Public Attributes

- static flowField flow

  *flow field used*

## Additional Inherited Members

### 6.21.1 Detailed Description

Definition at line 15 of file windy.h.

### 6.21.2 Constructor & Destructor Documentation

#### 6.21.2.1 windy()

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

default constructor.

Definition at line 29 of file windy.cpp.

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

### 6.21.3 Member Function Documentation

#### 6.21.3.1 loop()

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

windy scenario loop function

**Note**

opengl callback forces that function to be static

Definition at line 17 of file windy.cpp.

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

### 6.21.4 Member Data Documentation

#### 6.21.4.1 flow

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

flow field used

**Note**

opengl callback forces that function to be static

Definition at line 32 of file windy.h.

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

- include/windy.h
- src/windy.cpp

# Chapter 7

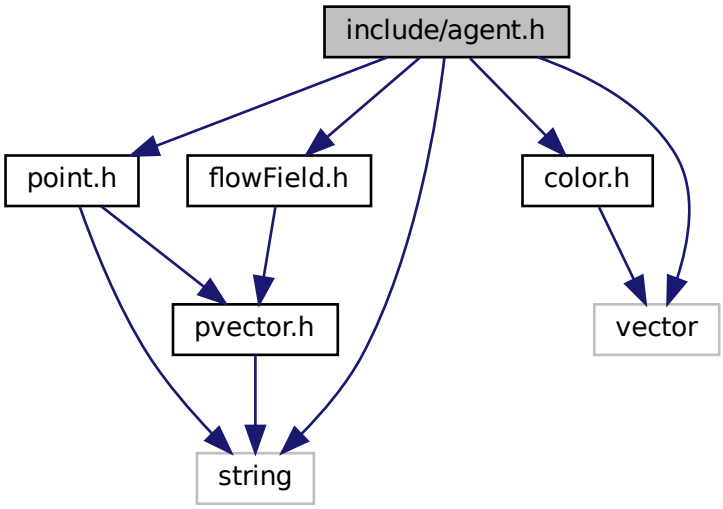
# File Documentation

## 7.1 include/agent.h File Reference

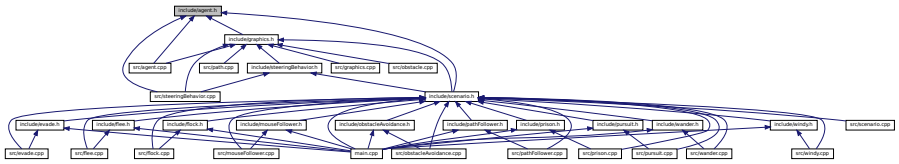
agent class defines all agent specifications

```
#include "point.h"
#include "color.h"
#include "flowField.h"
#include <vector>
#include <string>
```

Include dependency graph for agent.h:



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



### Classes

- class agent

### 7.1.1 Detailed Description

agent class defines all agent specifications

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

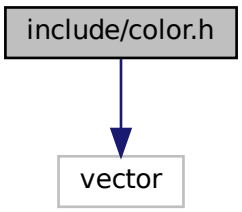
14.05.2021

## 7.2 include/color.h File Reference

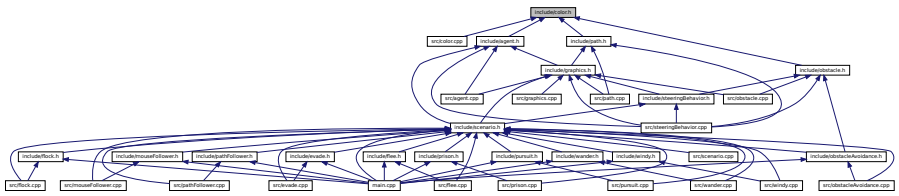
color class used for agent, path, wall etc. color

```
#include <vector>
```

Include dependency graph for color.h:




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



## Classes

- class color

  *fundament list of al colors*

## Macros

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

### 7.2.1 Detailed Description

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

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

13.05.2021

### 7.2.2 Macro Definition Documentation

#### 7.2.2.1 BLACK

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

Definition at line 10 of file color.h.

#### 7.2.2.2 BLUE

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

Definition at line 11 of file color.h.

#### 7.2.2.3 CYAN

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

Definition at line 13 of file color.h.

#### 7.2.2.4 GREEN

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

Definition at line 12 of file color.h.

#### 7.2.2.5 MAGENDA

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

Definition at line 16 of file color.h.

#### 7.2.2.6 RED

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

Definition at line 14 of file color.h.

#### 7.2.2.7 WHITE

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

Definition at line 17 of file color.h.

#### 7.2.2.8 YELLOW

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

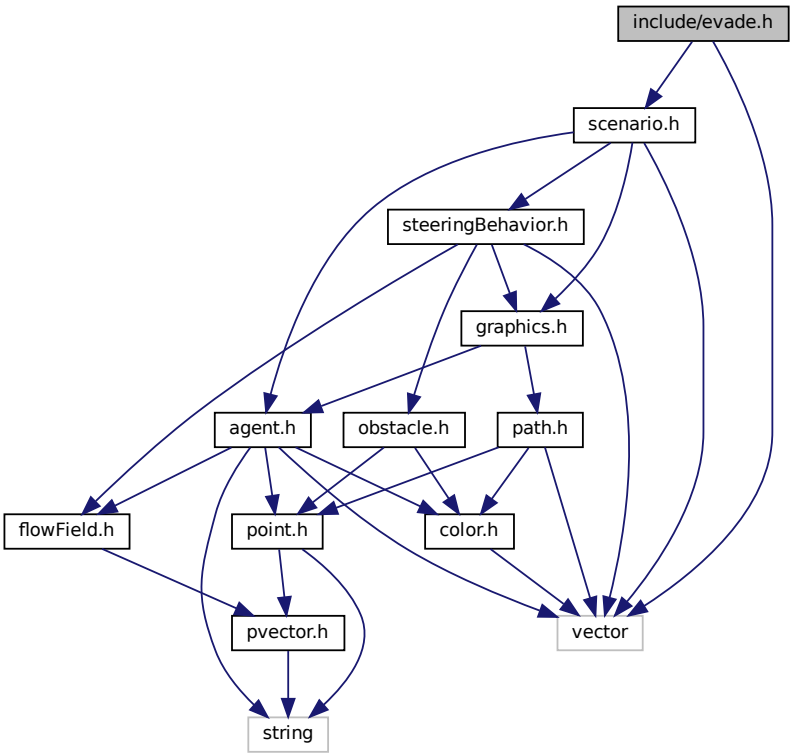
Definition at line 15 of file color.h.

## 7.3 include/evade.h File Reference

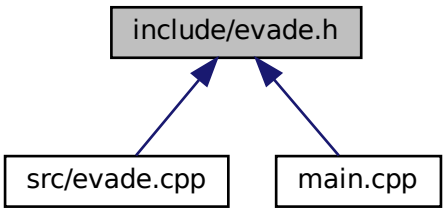
evade class inherited from scenario class

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

Include dependency graph for evade.h:




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




### Classes

- class evade

### 7.3.1 Detailed Description

evade class inherited from scenario class

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

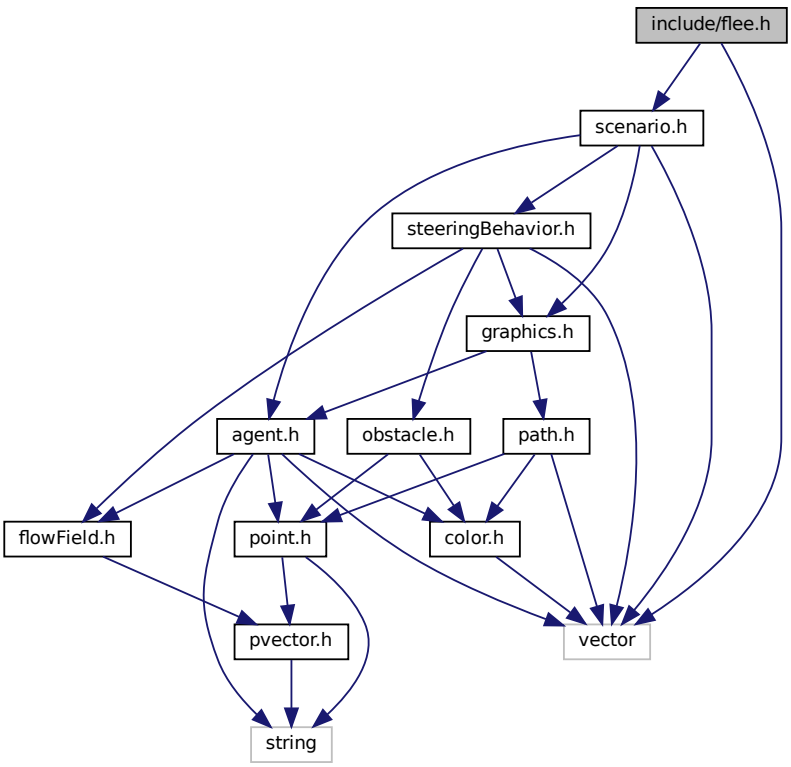
15.05.2021

## 7.4 include/flee.h File Reference

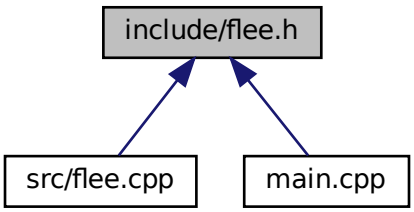
agents flee from mouse scenario

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

Include dependency graph for flee.h:



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




## Classes

- class flee

### 7.4.1 Detailed Description

agents flee from mouse scenario

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

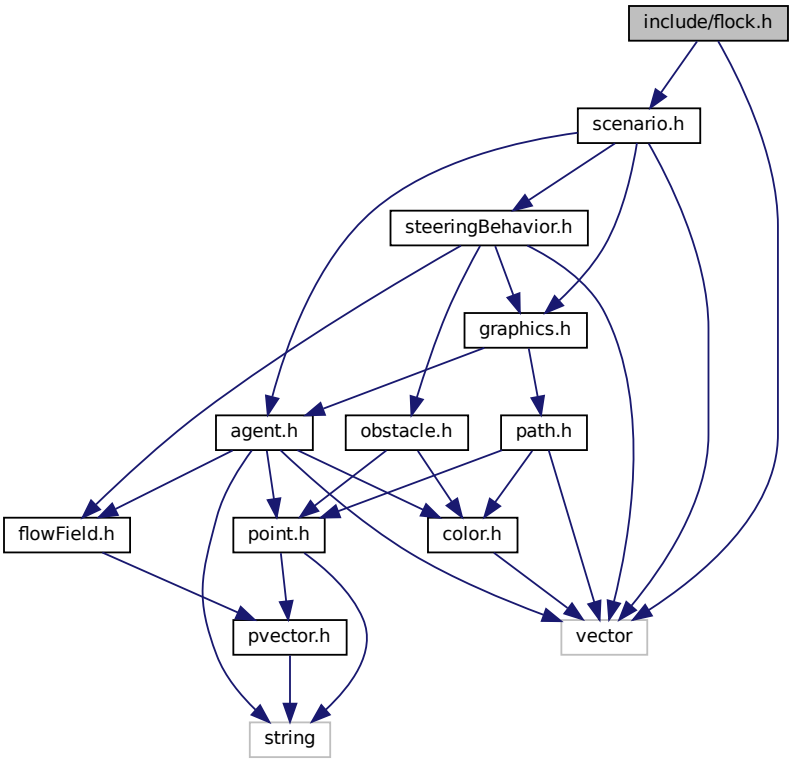
15.05.2021

## 7.5 include/flock.h File Reference

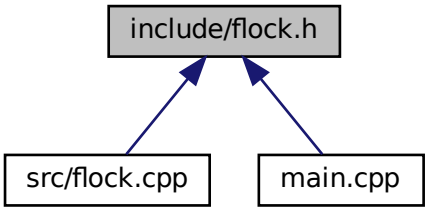
flocking agents scenario

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

Include dependency graph for flock.h:




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




## Classes

- class flock

### 7.5.1 Detailed Description

flocking agents scenario

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

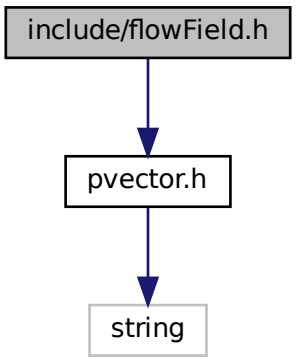
15.05.2021

## 7.6 include/flowField.h File Reference

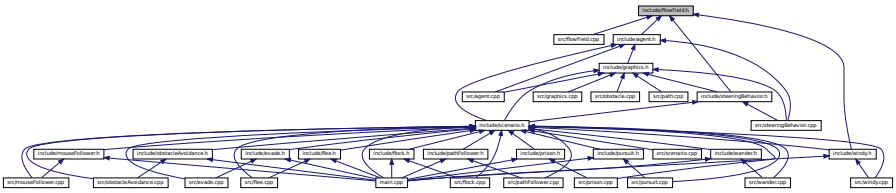
flowField class, screen can be filled with a force for each pixel

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

Include dependency graph for flowField.h:




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



## Classes

- class flowField

## Macros

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

### 7.6.1 Detailed Description

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

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

13.05.2021

### 7.6.2 Macro Definition Documentation

#### 7.6.2.1 FIELD_HEIGHT

```
#define FIELD_HEIGHT 34
```

Definition at line 13 of file flowField.h.

#### 7.6.2.2 FIELD_WIDTH

```
#define FIELD_WIDTH 34
```

Definition at line 12 of file flowField.h.

#### 7.6.2.3 GRAVITY

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

Definition at line 16 of file flowField.h.

#### 7.6.2.4 WIND_WEST

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

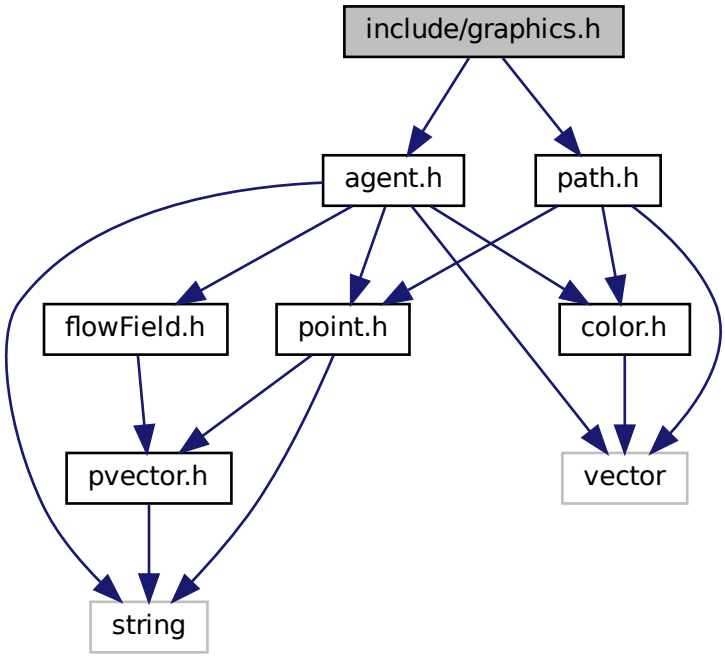
Definition at line 15 of file flowField.h.

## 7.7 include/graphics.h File Reference

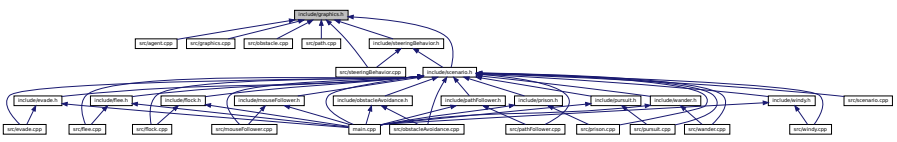
graphics class, drives openGL

```
#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

### 7.7.1 Detailed Description

graphics class, drives openGL

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

15.05.2021

### 7.7.2 Macro Definition Documentation

#### 7.7.2.1 ESC

```
#define ESC 27
```

Definition at line 16 of file graphics.h.

#### 7.7.2.2 HEIGHT

```
#define HEIGHT 34
```

Definition at line 14 of file graphics.h.

#### 7.7.2.3 PI

```
#define PI 3.14159265
```

Definition at line 17 of file graphics.h.

#### 7.7.2.4 WIDTH

```
#define WIDTH 34
```

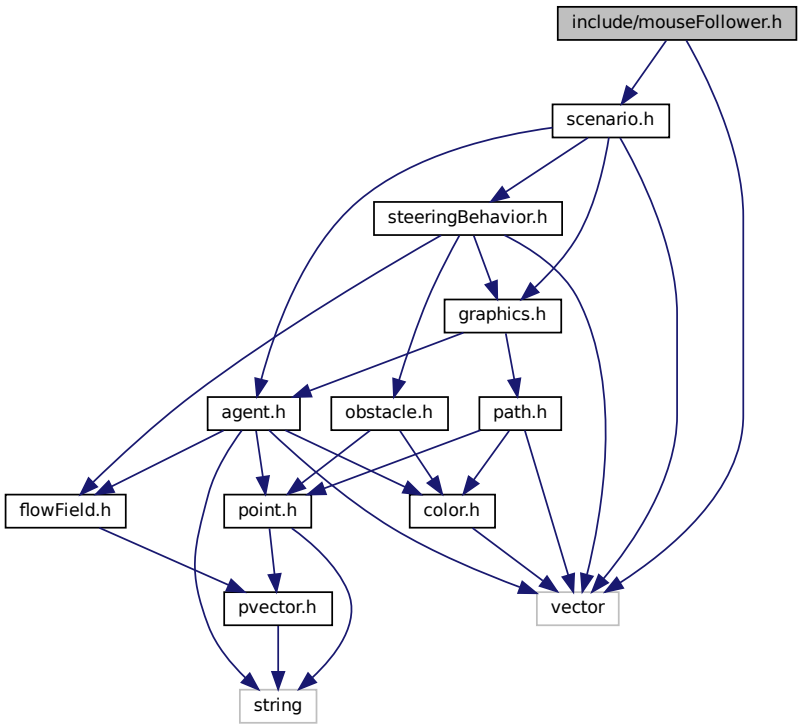
Definition at line 13 of file graphics.h.

## 7.8 include/mouseFollower.h File Reference

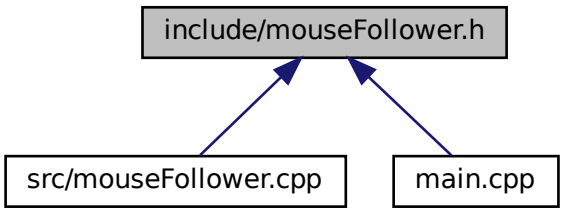
agents follow mouse scenario

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

Include dependency graph for mouseFollower.h:



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




### Classes

- class mouseFollower

### 7.8.1 Detailed Description

agents follow mouse scenario

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

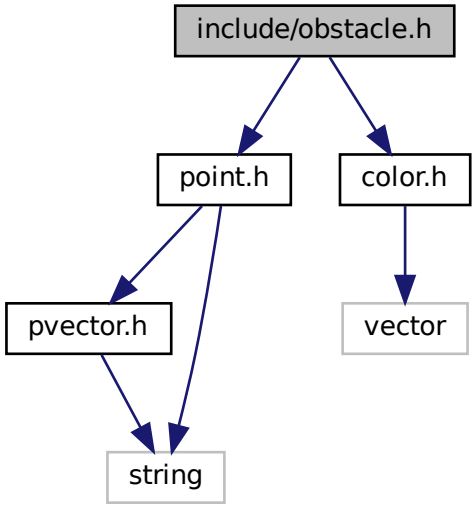
15.05.2021

## 7.9 include/obstacle.h File Reference

circular obstacles for agent avoidance behaviors

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

Include dependency graph for obstacle.h:




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

## Classes

- class obstacle

### 7.9.1 Detailed Description

circular obstacles for agent avoidance behaviors

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

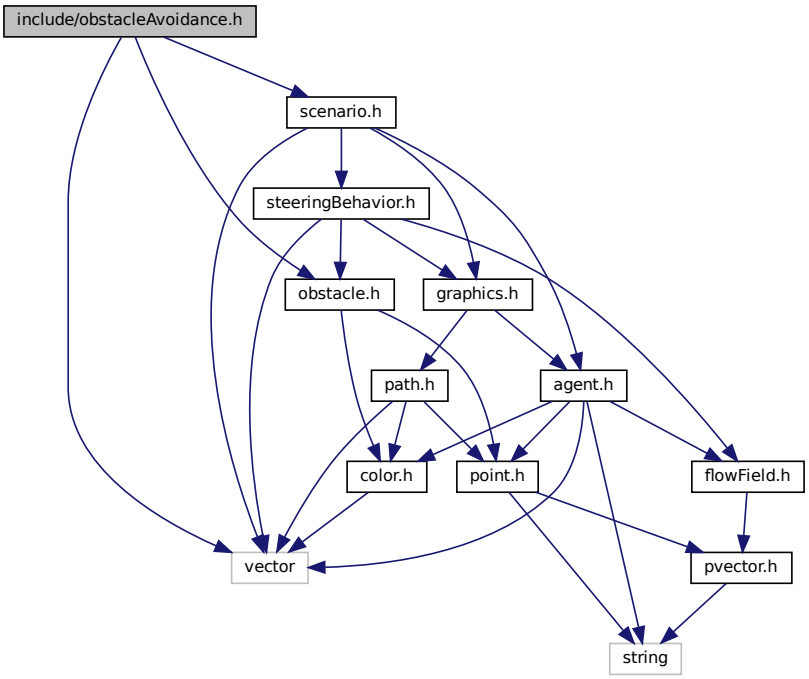
12.05.2021

## 7.10 include/obstacleAvoidance.h File Reference

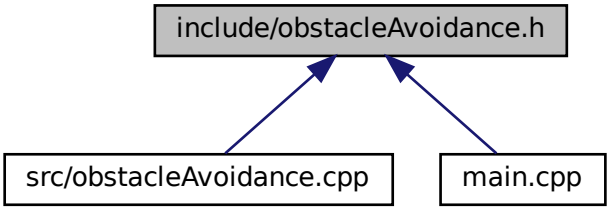
agents avoid from obstacles scenario

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

Include dependency graph for obstacleAvoidance.h:




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




### Classes

- class obstacleAvoidance

### 7.10.1 Detailed Description

agents avoid from obstacles scenario

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

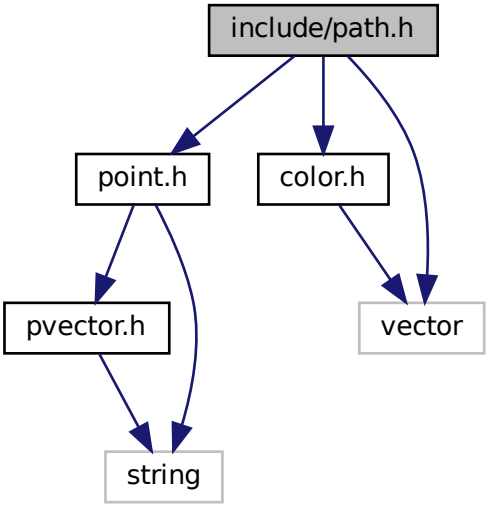
15.05.2021

## 7.11 include/path.h File Reference

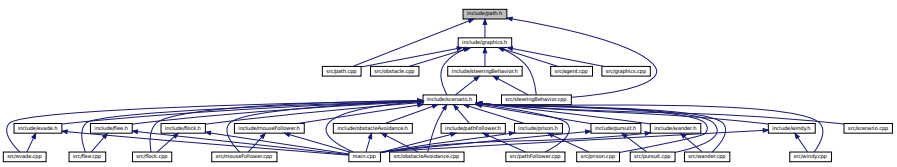
path class used for path following steering behaviors.

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

Include dependency graph for path.h:




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



### Classes

- class path

### 7.11.1 Detailed Description

path class used for path following steering behaviors.

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

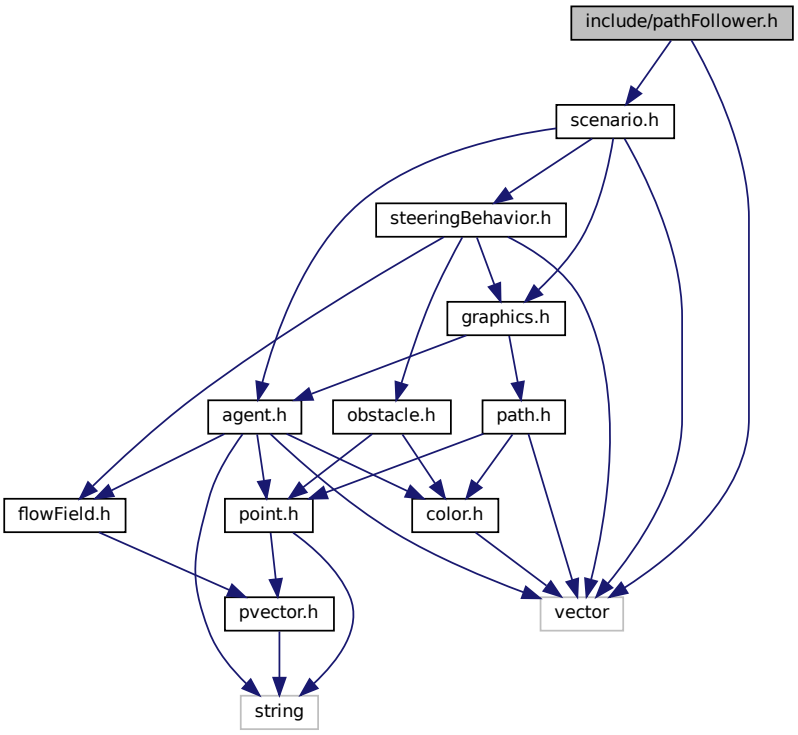
12.05.2021

## 7.12 include/pathFollower.h File Reference

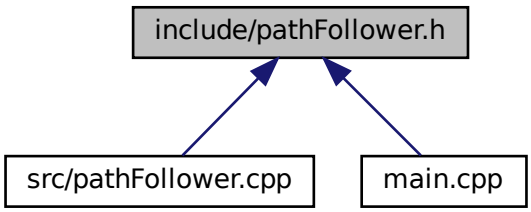
path following scenario

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

Include dependency graph for pathFollower.h:



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




### Classes

- class pathFollower

### 7.12.1 Detailed Description

path following scenario

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

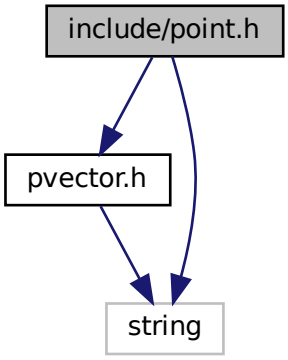
15.05.2021

## 7.13 include/point.h File Reference

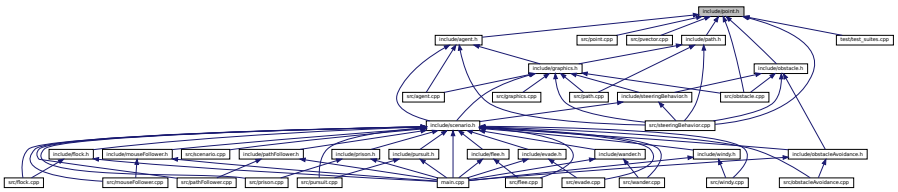
point class used for point operations

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

Include dependency graph for point.h:




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



## Classes

- class point

### 7.13.1 Detailed Description

point class used for point operations

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

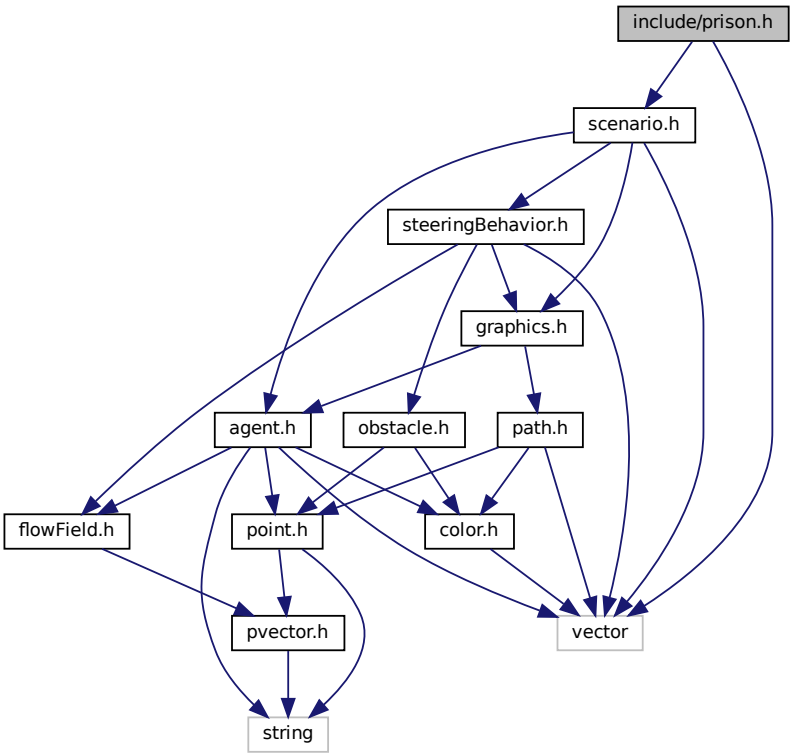
15.05.2021

## 7.14 include/prison.h File Reference

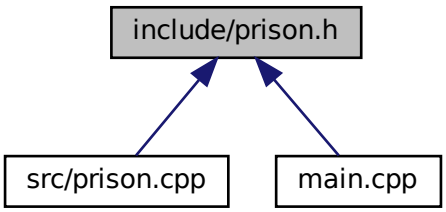
agents cant escape from field scenario

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

Include dependency graph for prison.h:




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




### Classes

- class prison

### 7.14.1 Detailed Description

agents cant escape from field scenario

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

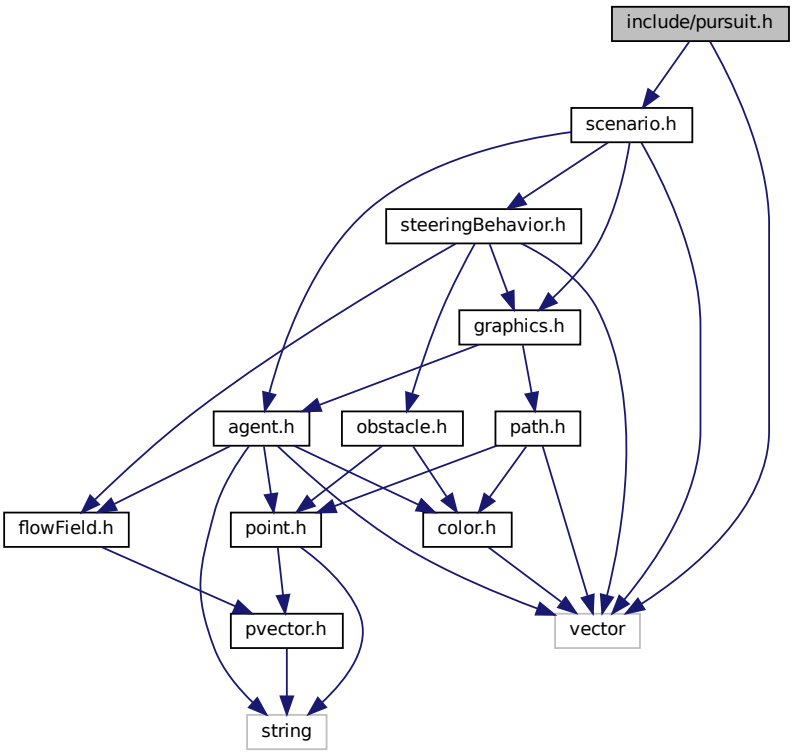
15.05.2021

## 7.15 include/pursuit.h File Reference

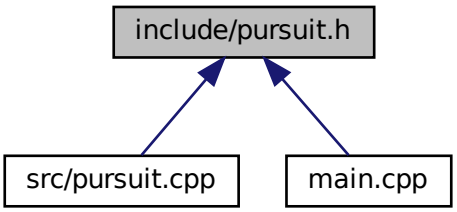
one agent pursue other one scenario

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

Include dependency graph for pursuit.h:



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




## Classes

- class pursuit

### 7.15.1 Detailed Description

one agent pursue other one scenario

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

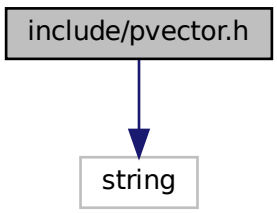
15.05.2021

## 7.16 include/pvector.h File Reference

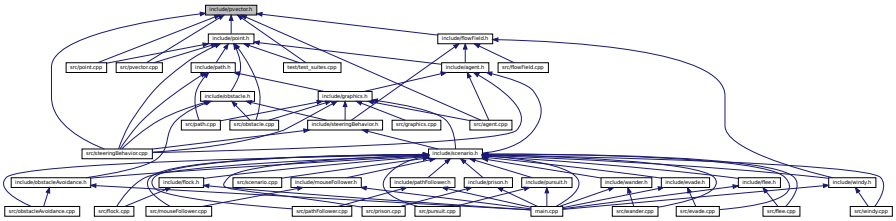
pvector class used for 2D vector operations

```
#include <string>
```

Include dependency graph for pvector.h:



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



## Classes

- class pvector

## Macros

- #define PI 3.14159265

### 7.16.1 Detailed Description

pvector class used for 2D vector operations

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

15.05.2021

### 7.16.2 Macro Definition Documentation

#### 7.16.2.1 PI

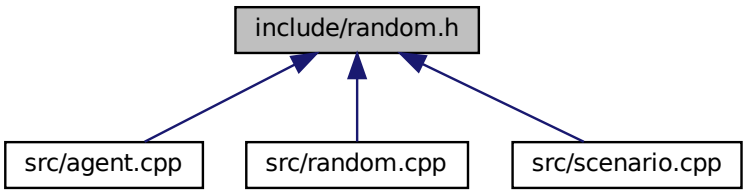
```
#define PI 3.14159265
```

Definition at line 11 of file pvector.h.

## 7.17 include/random.h File Reference

utility class for random operations

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




### Classes

- class random

### 7.17.1 Detailed Description

utility class for random operations

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

15.05.2021

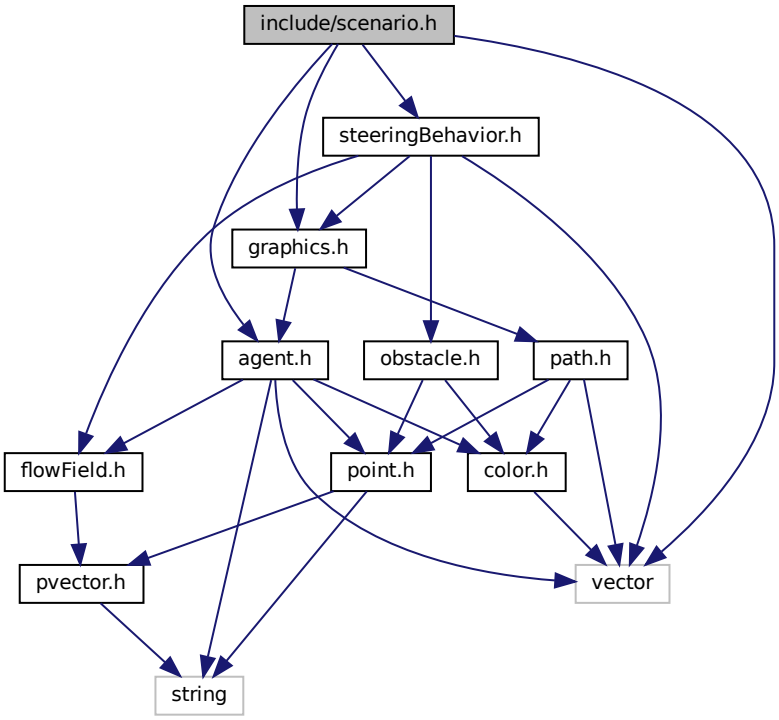
## 7.18 include/scenario.h File Reference

base class for all scenarios

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

```
#include <vector>
```

Include dependency graph for scenario.h:




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

## Classes

- class scenario

## Enumerations

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

### 7.18.1 Detailed Description

base class for all scenarios

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

15.05.2021

### 7.18.2 Enumeration Type Documentation

#### 7.18.2.1 types

```
enum types
```

**Enumerator**

| | |
|---|---|
| RANDOM | |
| STATIC | |
| TROOP | |

Definition at line 17 of file scenario.h.

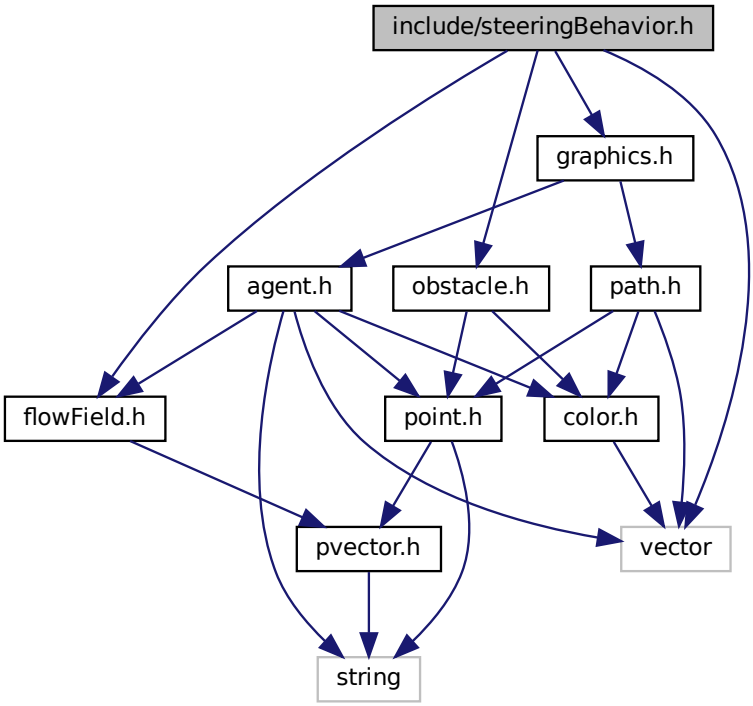
```
17 { RANDOM=0, STATIC, TROOP };
```

## 7.19 include/steeringBehavior.h File Reference

functions for autonomous steering behaviors

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

Include dependency graph for steeringBehavior.h:




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

## Classes

- class steeringBehavior

## Macros

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

### 7.19.1 Detailed Description

functions for autonomous steering behaviors

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

15.05.2021

### 7.19.2 Macro Definition Documentation

#### 7.19.2.1 AVOID_OBSTACLE

```
#define AVOID_OBSTACLE 4
```

Definition at line 21 of file steeringBehavior.h.

#### 7.19.2.2 CIRCLE_DISTANCE

```
#define CIRCLE_DISTANCE 0.1
```

Definition at line 15 of file steeringBehavior.h.

#### 7.19.2.3 CIRCLE_RADIUS

```
#define CIRCLE_RADIUS 0.4
```

Definition at line 16 of file steeringBehavior.h.

#### 7.19.2.4 EVADE

```
#define EVADE 10
```

Definition at line 27 of file steeringBehavior.h.

#### 7.19.2.5 FLEE

```
#define FLEE 8
```

Definition at line 25 of file steeringBehavior.h.

#### 7.19.2.6 FLOCK

```
#define FLOCK 6
```

Definition at line 23 of file steeringBehavior.h.

#### 7.19.2.7 FOLLOW_MOUSE

```
#define FOLLOW_MOUSE 1
```

Definition at line 18 of file steeringBehavior.h.

#### 7.19.2.8 IN_FLOW_FIELD

```
#define IN_FLOW_FIELD 3
```

Definition at line 20 of file steeringBehavior.h.

#### 7.19.2.9 PURSUIT

```
#define PURSUIT 9
```

Definition at line 26 of file steeringBehavior.h.

#### 7.19.2.10 STAY_IN_FIELD

```
#define STAY_IN_FIELD 2
```

Definition at line 19 of file steeringBehavior.h.

#### 7.19.2.11 STAY_IN_PATH

```
#define STAY_IN_PATH 5
```

Definition at line 22 of file steeringBehavior.h.

#### 7.19.2.12 WANDER

```
#define WANDER 7
```

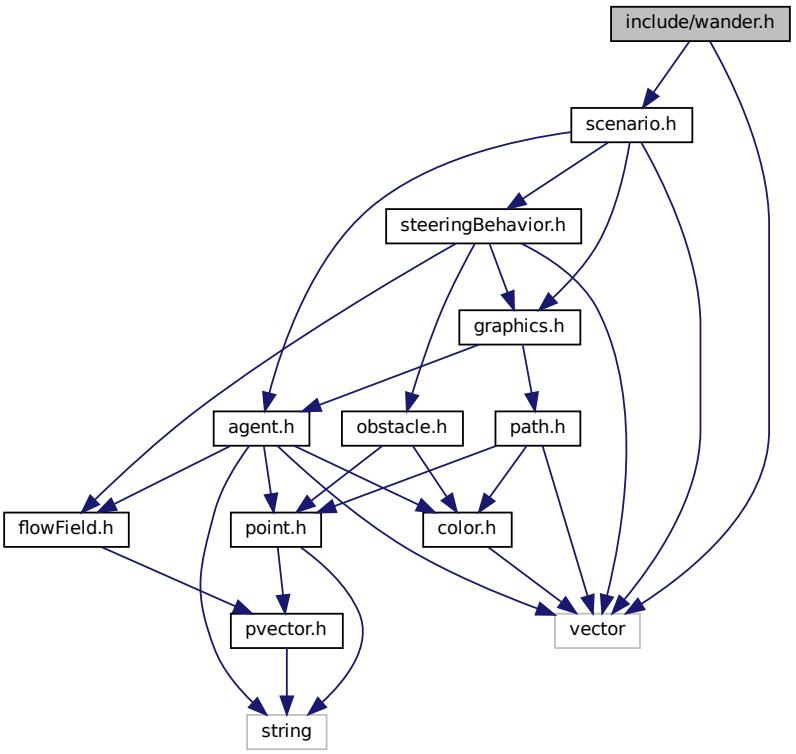
Definition at line 24 of file steeringBehavior.h.

## 7.20 include/wander.h File Reference

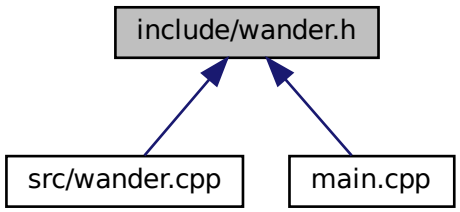
random wandering agents scenario

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

Include dependency graph for wander.h:



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




## Classes

- class wander

### 7.20.1 Detailed Description

random wandering agents scenario

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

15.05.2021

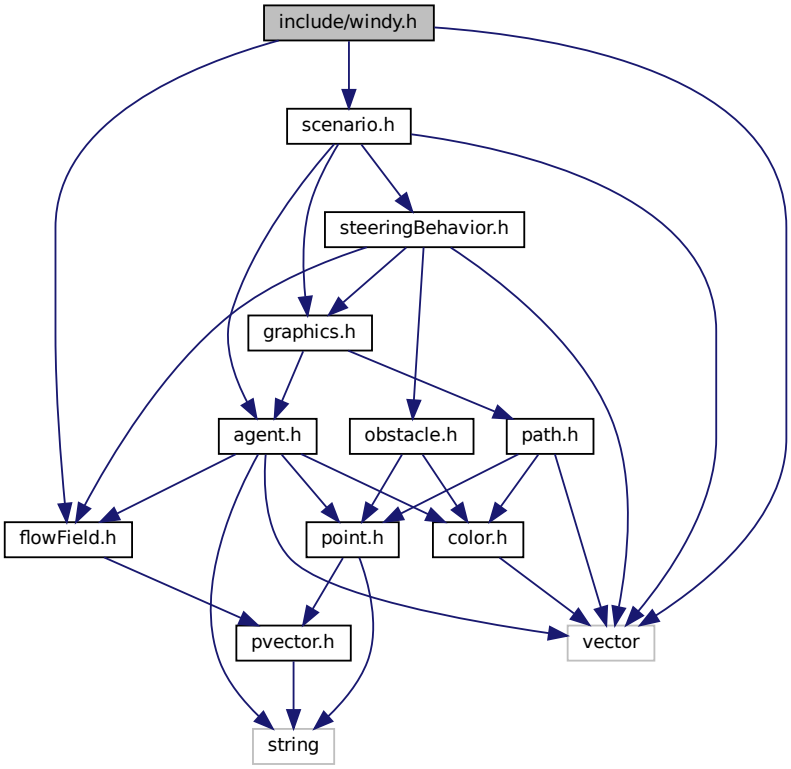
## 7.21 include/windy.h File Reference

windy air scenario

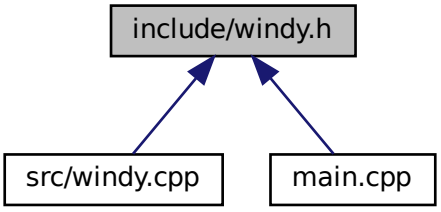
```
#include "scenario.h"
#include "flowField.h"
```

```
#include <vector>
```

Include dependency graph for windy.h:




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




## Classes

- class windy

### 7.21.1 Detailed Description

windy air scenario

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

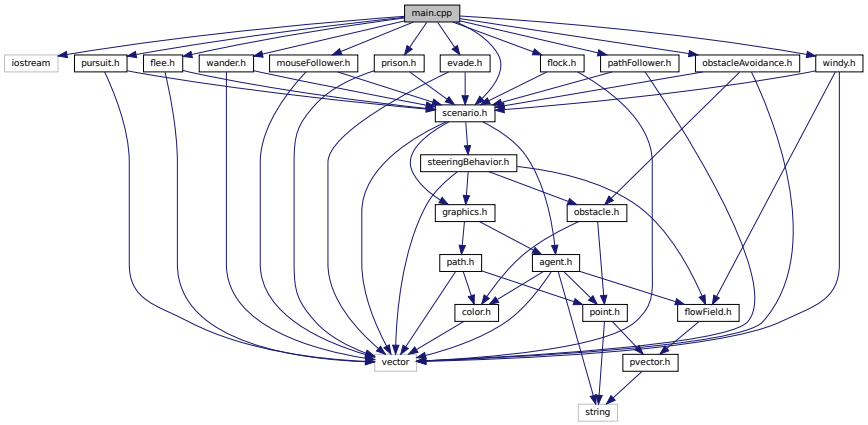
15.05.2021

## 7.22 main.cpp File Reference

client code

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

Include dependency graph for main.cpp:




## Functions

- void menu ()

  *displays menu*

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

  *main routine*

## Variables

- int mode

  *specifies user selected scenario*

### 7.22.1 Detailed Description

client code

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

15.05.2021

### 7.22.2 Function Documentation

#### 7.22.2.1 main()

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

main routine

Definition at line 48 of file main.cpp.

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

#### 7.22.2.2 menu()

```
void menu ( )
```

displays menu

Definition at line 31 of file main.cpp.

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

### 7.22.3 Variable Documentation

#### 7.22.3.1 mode

```
int mode
```

specifies user selected scenario

Definition at line 26 of file main.cpp.

## 7.23 README.md File Reference

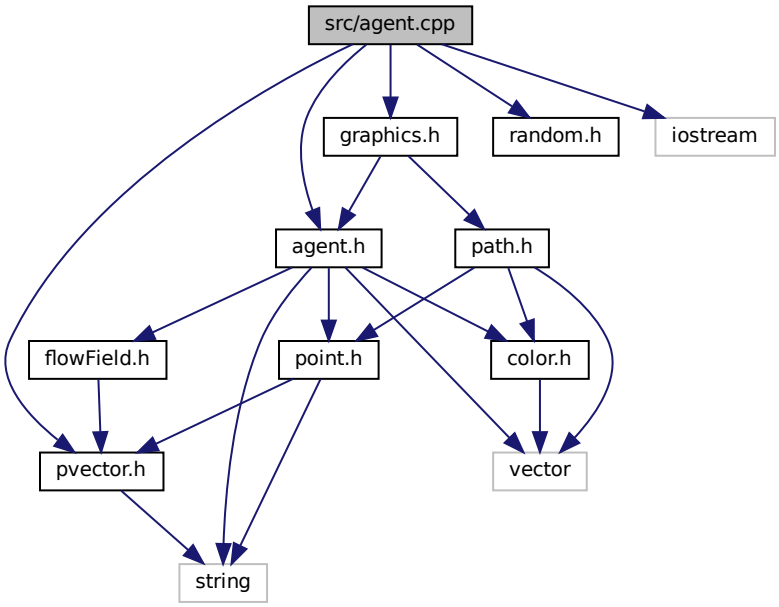
## 7.24 src/agent.cpp File Reference

implementation of the agent class

```
#include "agent.h"
#include "pvector.h"
#include "graphics.h"
#include "random.h"
```

```
#include <iostream>
```

Include dependency graph for agent.cpp:




### 7.24.1 Detailed Description

implementation of the agent class

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

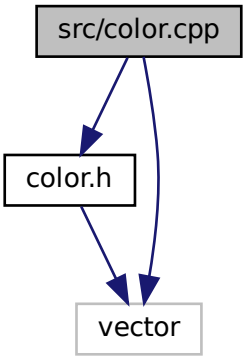
14.05.2021

## 7.25 src/color.cpp File Reference

color class implementation

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

Include dependency graph for color.cpp:




### 7.25.1 Detailed Description

color class implementation

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

13.05.2021

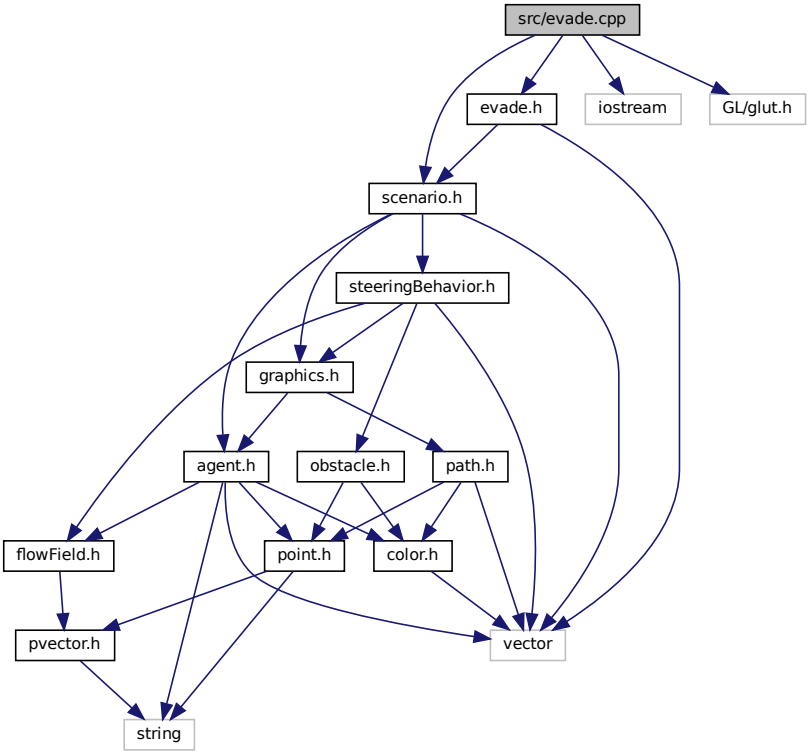
## 7.26 src/evade.cpp File Reference

evade class implementation

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

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

Include dependency graph for evade.cpp:




### 7.26.1 Detailed Description

evade class implementation

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

15.05.2021

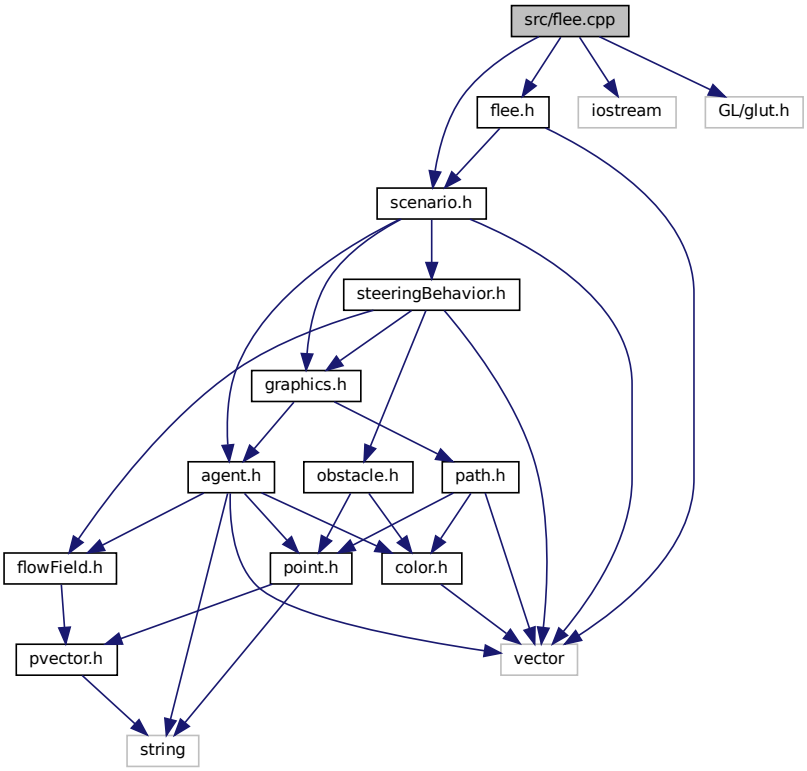
## 7.27 src/flee.cpp File Reference

flee class implementation

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

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

Include dependency graph for flee.cpp:




### 7.27.1 Detailed Description

flee class implementation

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

15.05.2021

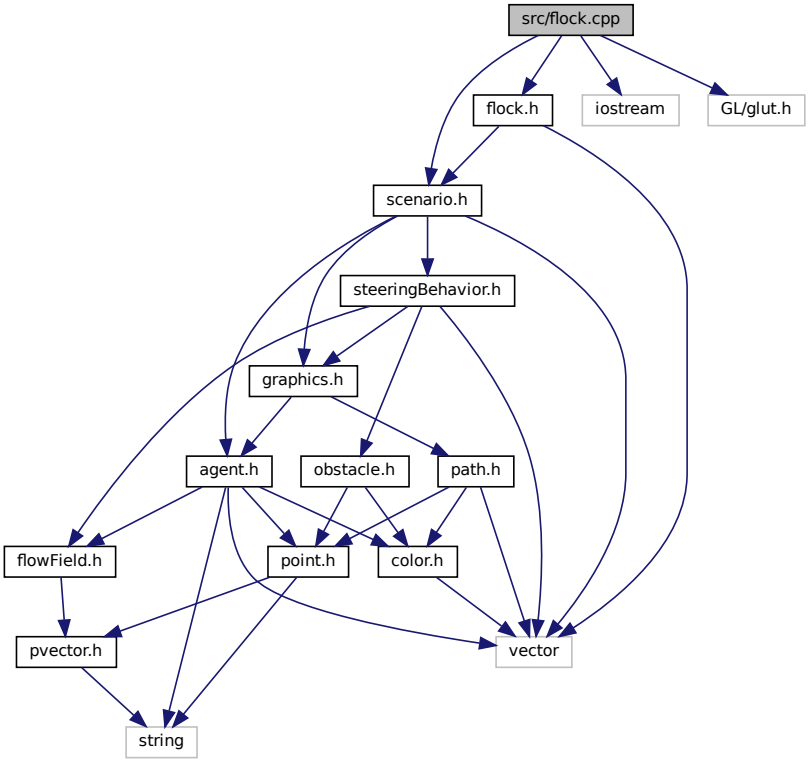
## 7.28 src/flock.cpp File Reference

flock class implementation

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

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

Include dependency graph for flock.cpp:




### 7.28.1 Detailed Description

flock class implementation

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

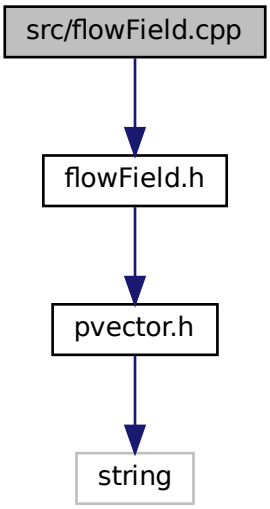
15.05.2021

## 7.29 src/flowField.cpp File Reference

flowField class implementation

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

Include dependency graph for flowField.cpp:




### 7.29.1 Detailed Description

flowField class implementation

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

13.05.2021

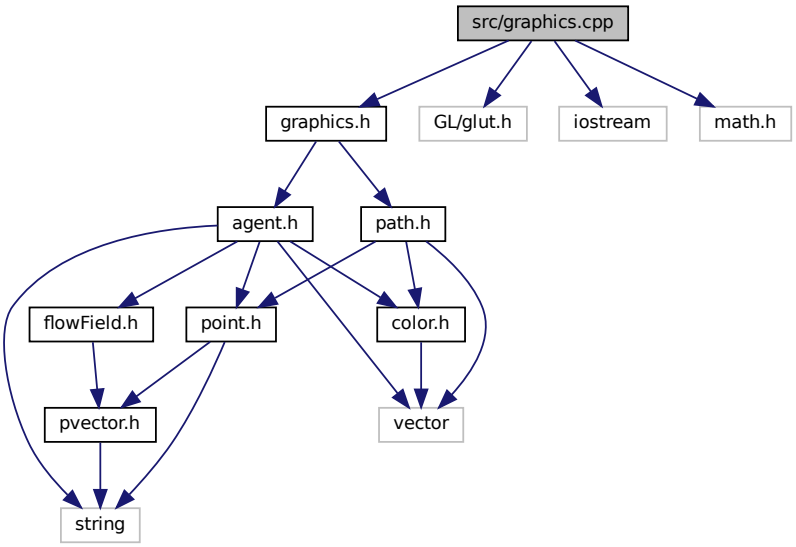
## 7.30 src/graphics.cpp File Reference

graphics class implementation

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

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

Include dependency graph for graphics.cpp:




### 7.30.1 Detailed Description

graphics class implementation

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

15.05.2021

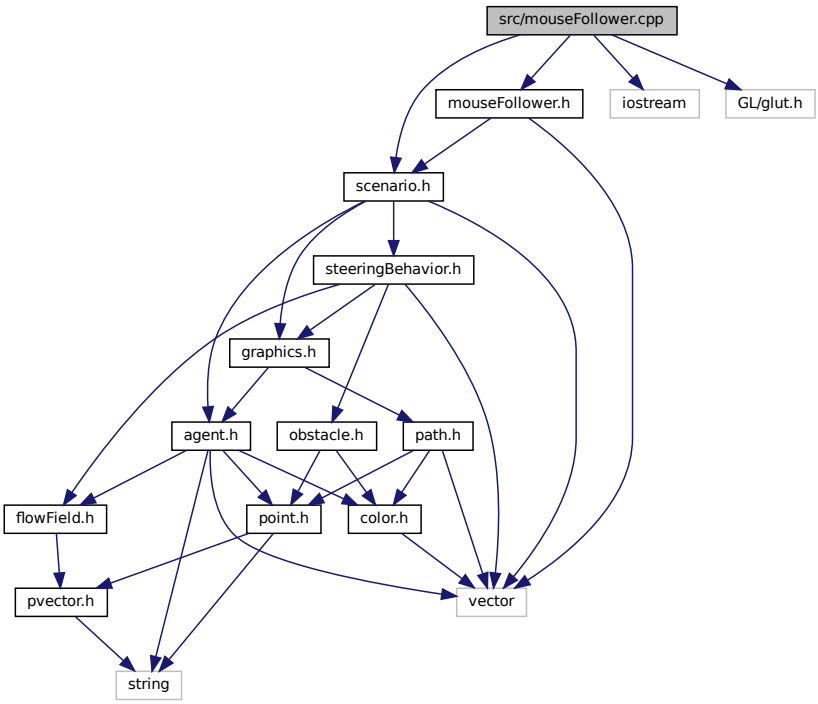
## 7.31 src/mouseFollower.cpp File Reference

mouseFollower class implementation

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

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

Include dependency graph for mouseFollower.cpp:




### 7.31.1 Detailed Description

mouseFollower class implementation

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

15.05.2021

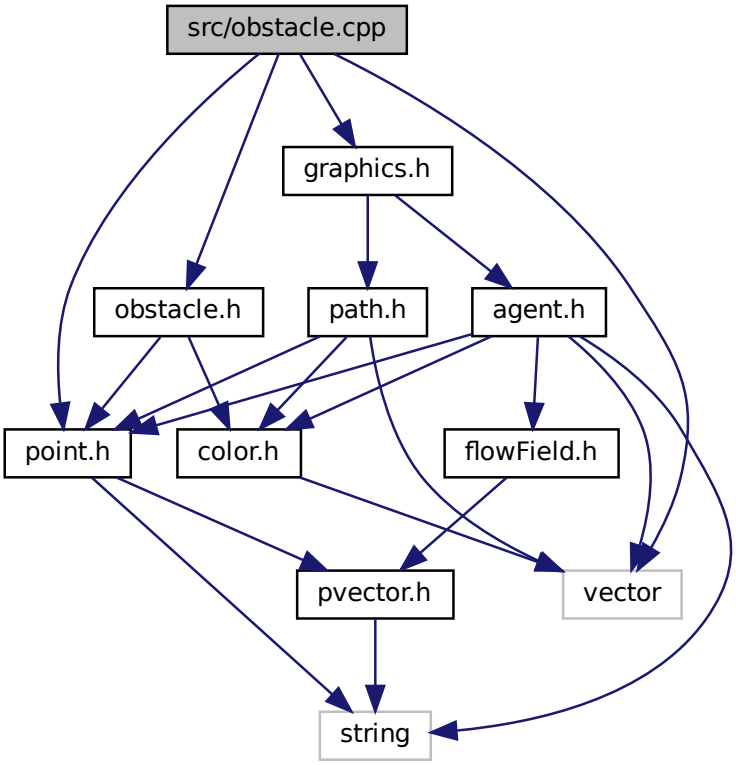
## 7.32 src/obstacle.cpp File Reference

obstacle class implementation

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

```
#include <vector>
```

Include dependency graph for obstacle.cpp:




### 7.32.1 Detailed Description

obstacle class implementation

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

12.05.2021

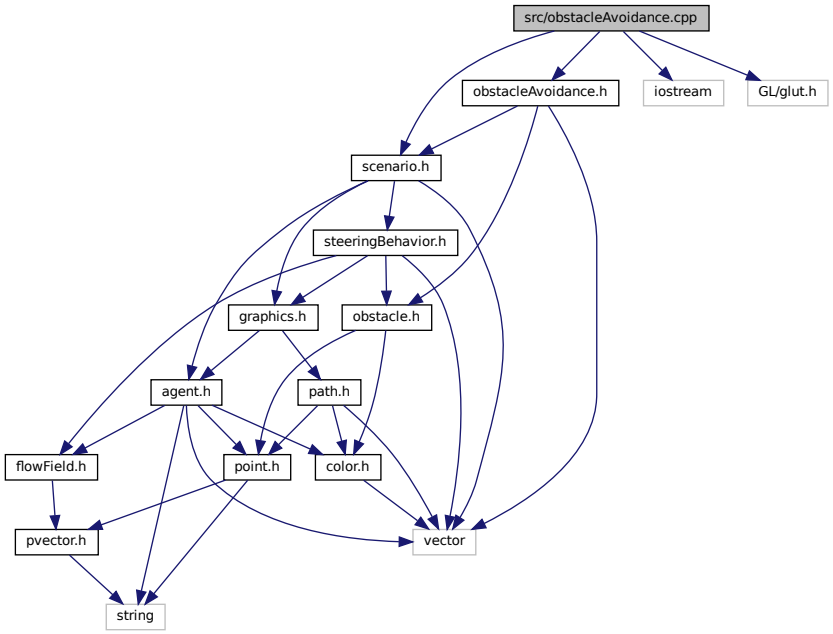
## 7.33 src/obstacleAvoidance.cpp File Reference

obstacleAvoidance class implementation

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

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

Include dependency graph for obstacleAvoidance.cpp:




### 7.33.1 Detailed Description

obstacleAvoidance class implementation

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

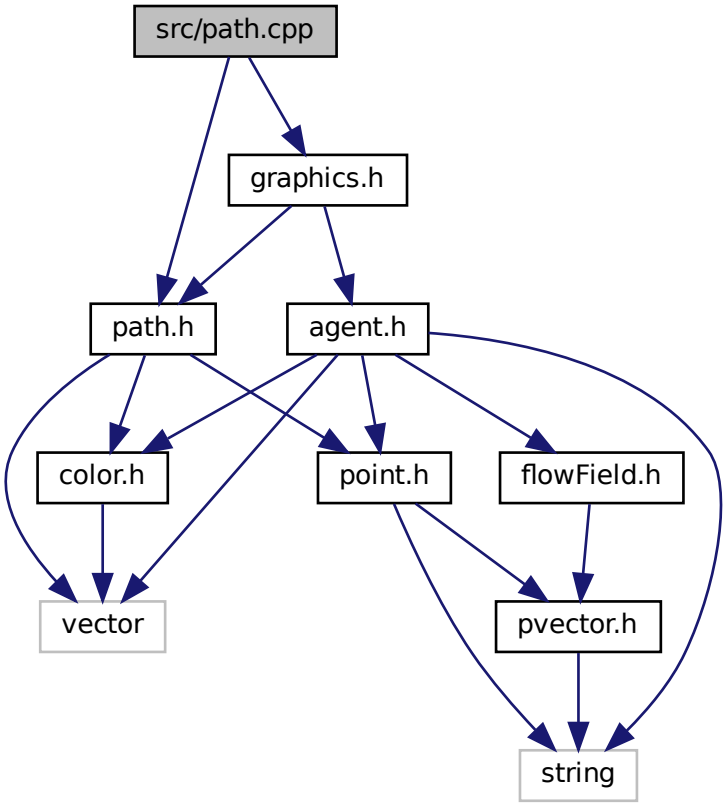
15.05.2021

## 7.34 src/path.cpp File Reference

path class implementation

```
#include "path.h"
#include "graphics.h"
```

Include dependency graph for path.cpp:




### 7.34.1 Detailed Description

path class implementation

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

12.05.2021

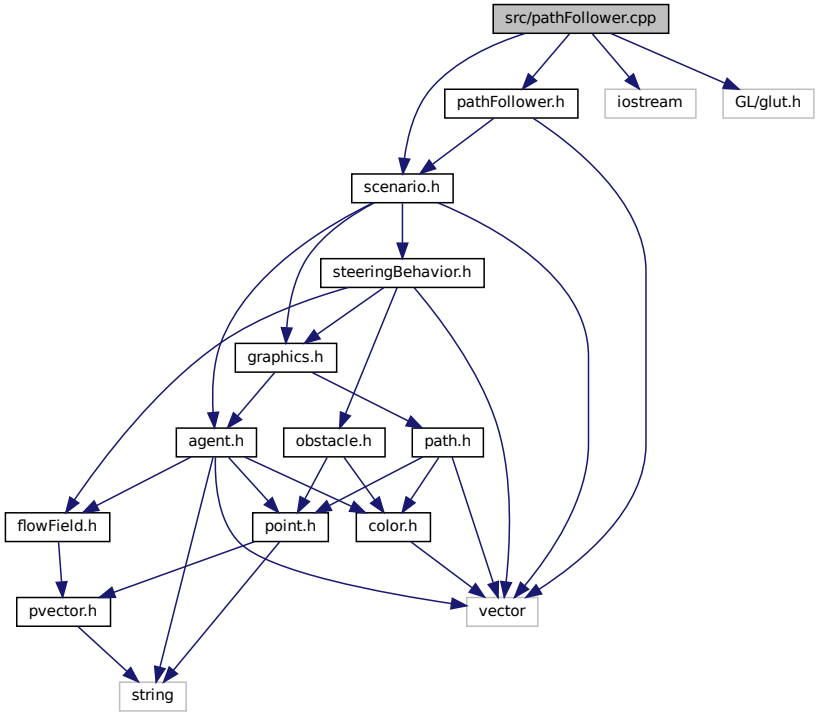
## 7.35 src/pathFollower.cpp File Reference

pathFollower class implementation

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

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

Include dependency graph for pathFollower.cpp:




### 7.35.1 Detailed Description

pathFollower class implementation

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

15.05.2021

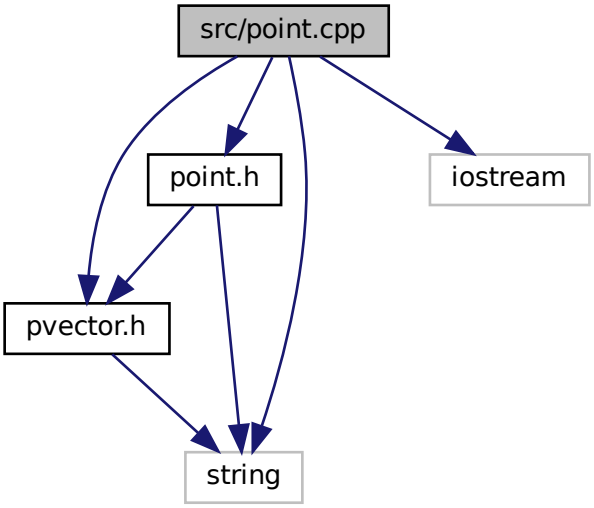
## 7.36 src/point.cpp File Reference

point class implementation file

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

```
#include <iostream>
```

Include dependency graph for point.cpp:




### 7.36.1 Detailed Description

point class implementation file

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

15.05.2021

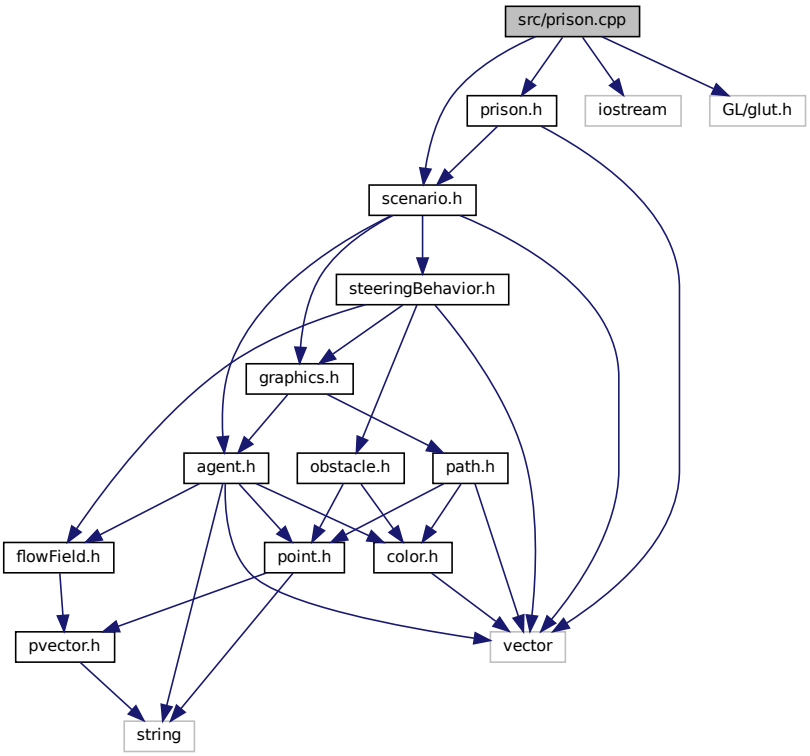
## 7.37 src/prison.cpp File Reference

prison class implementation

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

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

Include dependency graph for prison.cpp:




## Macros

- #define WALL 30
- #define DISTANCE 2

### 7.37.1 Detailed Description

prison class implementation

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

15.05.2021

### 7.37.2 Macro Definition Documentation

#### 7.37.2.1 DISTANCE

```
#define DISTANCE 2
```

Definition at line 14 of file prison.cpp.

#### 7.37.2.2 WALL

```
#define WALL 30
```

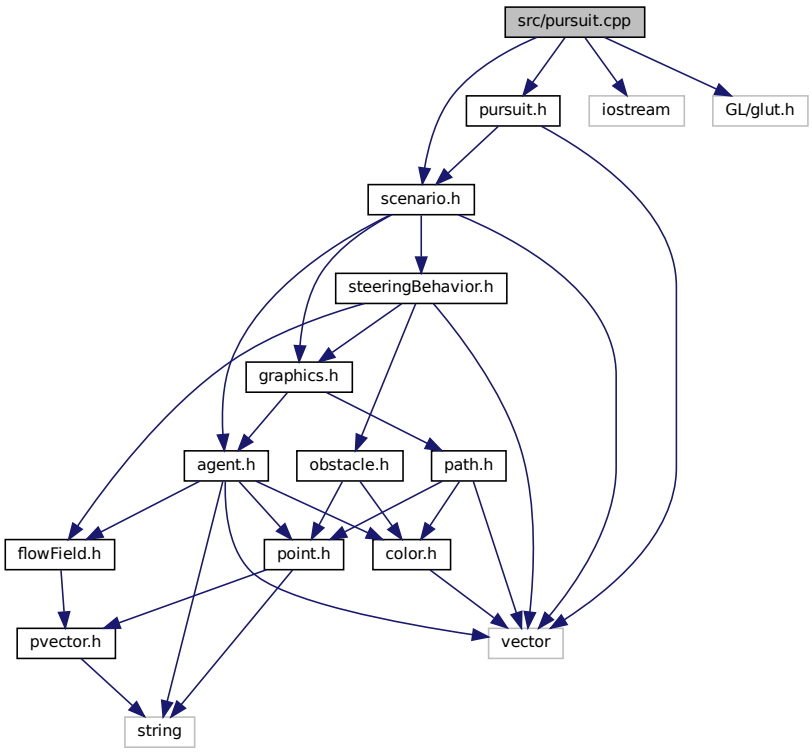
Definition at line 13 of file prison.cpp.

## 7.38 src/pursuit.cpp File Reference

prison class implementation

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

Include dependency graph for pursuit.cpp:



### 7.38.1 Detailed Description

prison class implementation

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

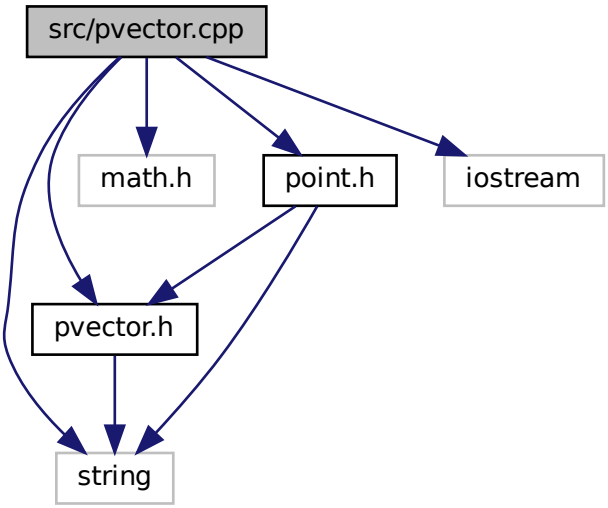
15.05.2021

## 7.39 src/pvector.cpp File Reference

pvector class implementation

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

Include dependency graph for pvector.cpp:




### 7.39.1 Detailed Description

pvector class implementation

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

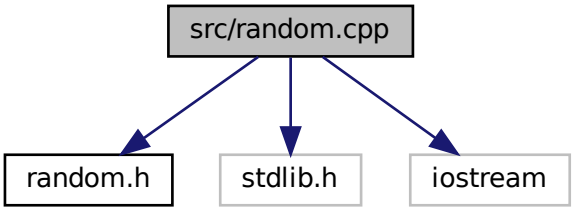
15.05.2021

## 7.40 src/random.cpp File Reference

utility class for random operations

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

Include dependency graph for random.cpp:




### 7.40.1 Detailed Description

utility class for random operations

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

15.05.2021

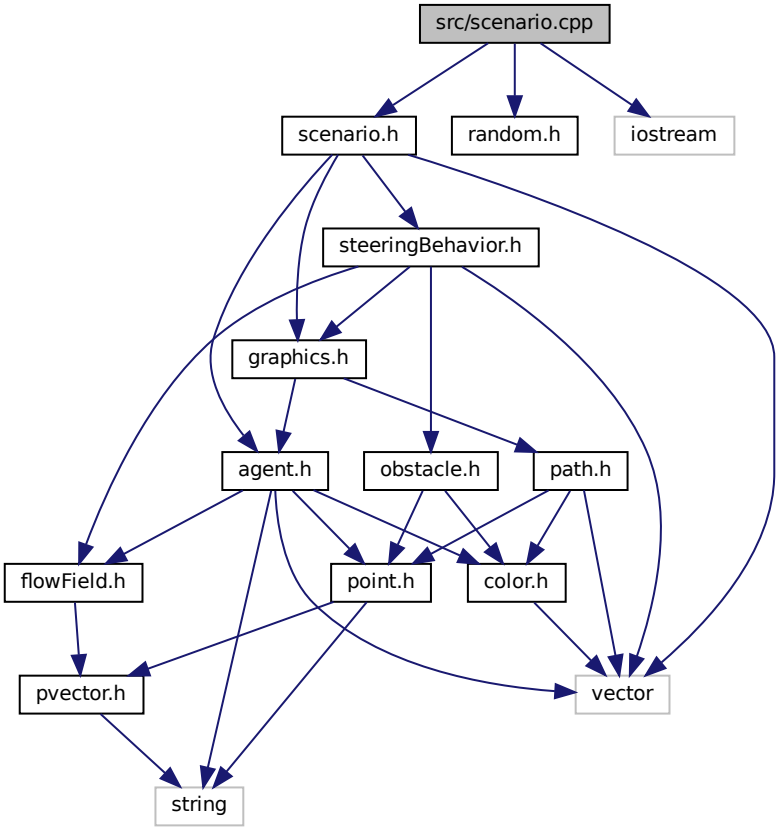
## 7.41 src/scenario.cpp File Reference

scenario base class implementation

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

```
#include <iostream>
```

Include dependency graph for scenario.cpp:




## Macros

- #define MAX_NUMBER_OF_AGENTS 50

### 7.41.1 Detailed Description

scenario base class implementation

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

15.05.2021

### 7.41.2 Macro Definition Documentation

#### 7.41.2.1 MAX_NUMBER_OF_AGENTS

```
#define MAX_NUMBER_OF_AGENTS 50
```

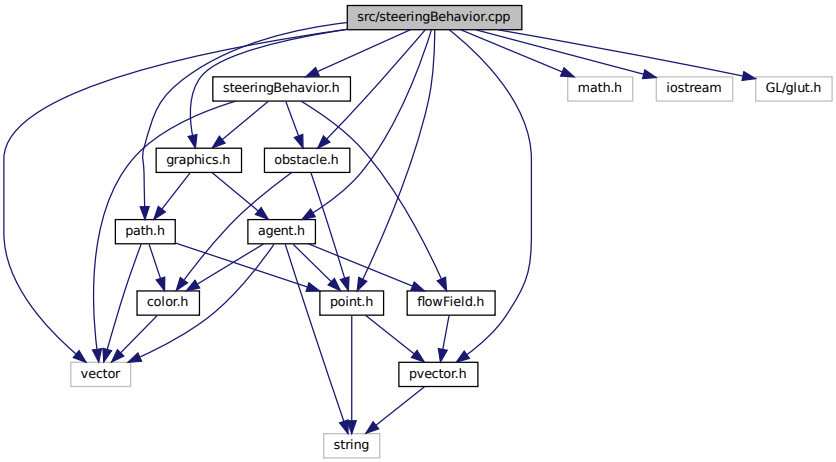
Definition at line 12 of file scenario.cpp.

## 7.42 src/steeringBehavior.cpp File Reference

implementation of autonomous steering behaviors

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

Include dependency graph for steeringBehavior.cpp:




### 7.42.1 Detailed Description

implementation of autonomous steering behaviors

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

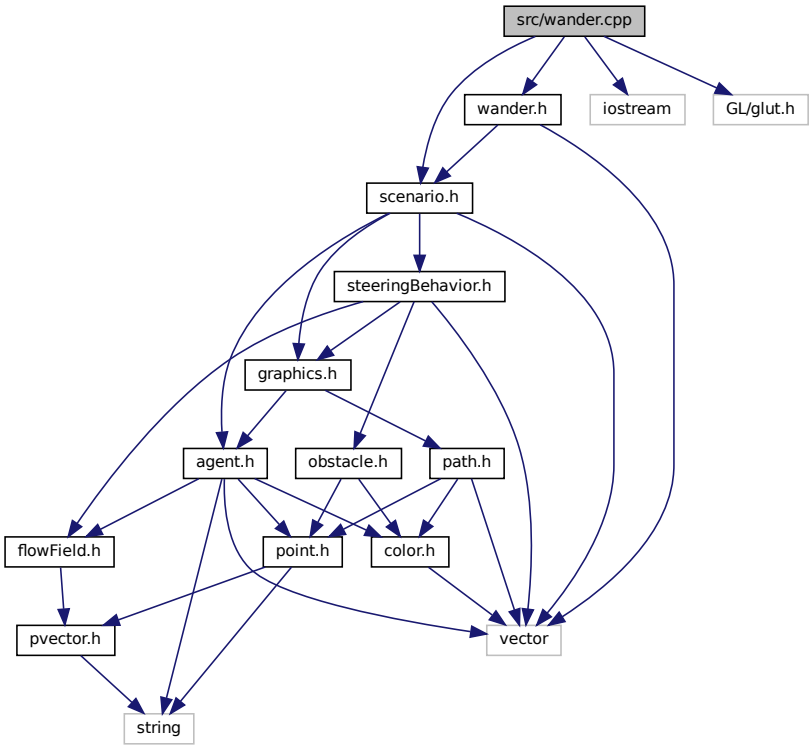
15.05.2021

## 7.43 src/wander.cpp File Reference

wander class implementation

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

Include dependency graph for wander.cpp:




### 7.43.1 Detailed Description

wander class implementation

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

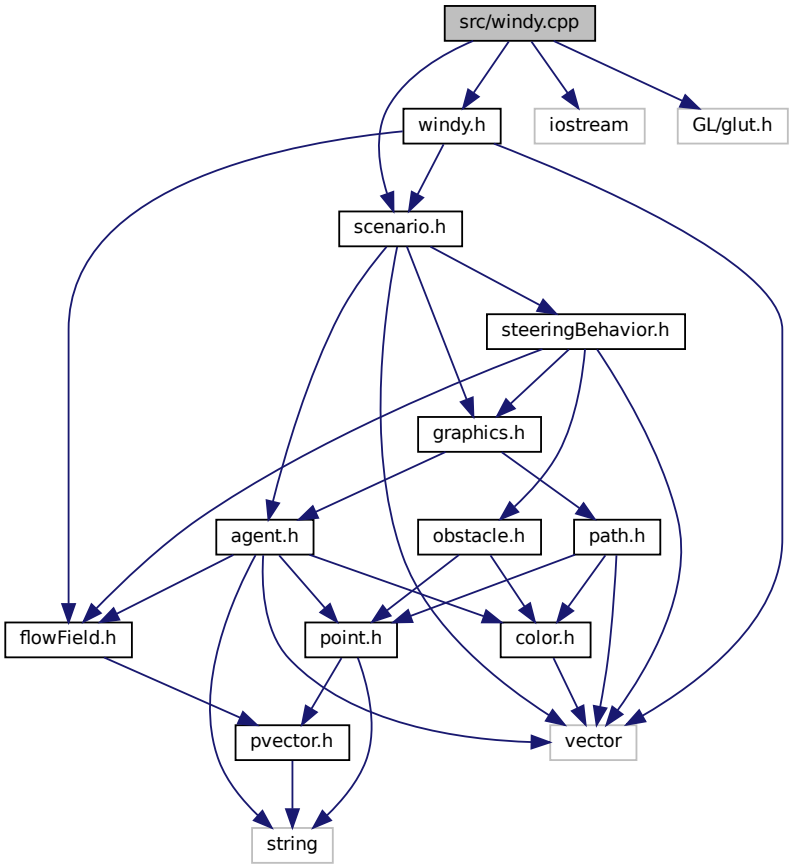
15.05.2021

## 7.44 src/windy.cpp File Reference

windy class implementation

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

Include dependency graph for windy.cpp:




### 7.44.1 Detailed Description

windy class implementation

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

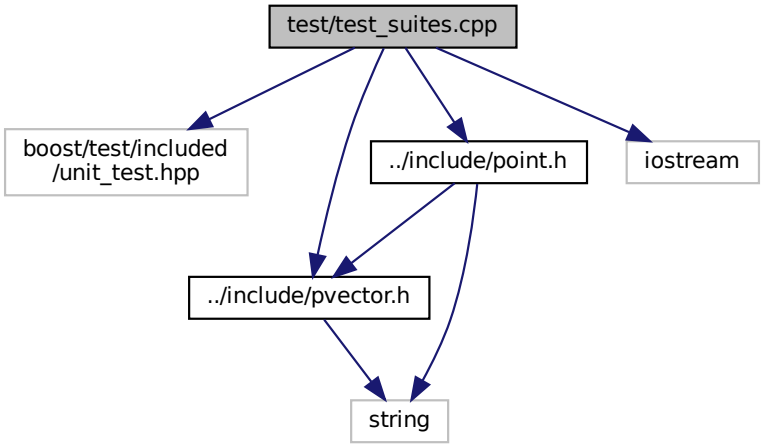
15.05.2021

## 7.45 test/test_suites.cpp File Reference

unit test suites

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

Include dependency graph for test_suites.cpp:




### Macros

- #define BOOST_TEST_MODULE test_suites

### Functions

- BOOST_AUTO_TEST_CASE (s1t1)

  *pvector magnitude test case*
- BOOST_AUTO_TEST_CASE (s1t2)

  *pvector mul test case*
- BOOST_AUTO_TEST_CASE (s1t3)

  *pvector div test case*
- BOOST_AUTO_TEST_CASE (s1t4)

  *pvector dotproduct test case*
- BOOST_AUTO_TEST_CASE (s1t5)

  *pvector angle between vectors test case*
- BOOST_AUTO_TEST_CASE (s1t6)

  *pvector get vector angle test case*
- BOOST_AUTO_TEST_CASE (s1t7)

  *pvector normalize test case*
- BOOST_AUTO_TEST_CASE (s1t8)

*pvector limit test case*

- BOOST_AUTO_TEST_CASE (s1t9)

  *pvector overloaded operators test case*
- BOOST_AUTO_TEST_CASE (s2t1)

  *point multiplication test case*
- BOOST_AUTO_TEST_CASE (s2t2)

  *point division test case*
- BOOST_AUTO_TEST_CASE (s2t3)

  *point overloaded operators test case*

## 7.45.1 Detailed Description

unit test suites

**Author**

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

**Date**

15.05.2021

## 7.45.2 Macro Definition Documentation

### 7.45.2.1 BOOST_TEST_MODULE

```
#define BOOST_TEST_MODULE test_suites
```

Definition at line 8 of file test_suites.cpp.

## 7.45.3 Function Documentation

### 7.45.3.1 BOOST_AUTO_TEST_CASE() [1/12]

```
BOOST_AUTO_TEST_CASE (
              s1t1  )
```

pvector magnitude test case

Definition at line 22 of file test_suites.cpp.

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

#### 7.45.3.2 BOOST_AUTO_TEST_CASE() [2/12]

```
BOOST_AUTO_TEST_CASE (
              s1t2  )
```

pvector mul test case

Definition at line 33 of file test_suites.cpp.

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

#### 7.45.3.3 BOOST_AUTO_TEST_CASE() [3/12]

```
BOOST_AUTO_TEST_CASE (
              s1t3  )
```

pvector div test case

Definition at line 44 of file test_suites.cpp.

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

#### 7.45.3.4 BOOST_AUTO_TEST_CASE() [4/12]

```
BOOST_AUTO_TEST_CASE (
              s1t4  )
```

pvector dotproduct test case

Definition at line 55 of file test_suites.cpp.

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

#### 7.45.3.5 BOOST_AUTO_TEST_CASE() [5/12]

```
BOOST_AUTO_TEST_CASE (
              s1t5  )
```

pvector angle between vectors test case

Definition at line 66 of file test_suites.cpp.

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

#### 7.45.3.6 BOOST_AUTO_TEST_CASE() [6/12]

```
BOOST_AUTO_TEST_CASE (
              s1t6  )
```

pvector get vector angle test case

Definition at line 77 of file test_suites.cpp.

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

#### 7.45.3.7 BOOST_AUTO_TEST_CASE() [7/12]

```
BOOST_AUTO_TEST_CASE (
              s1t7  )
```

pvector normalize test case

Definition at line 87 of file test_suites.cpp.

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

#### 7.45.3.8 BOOST_AUTO_TEST_CASE() [8/12]

```
BOOST_AUTO_TEST_CASE (
              s1t8  )
```

pvector limit test case

Definition at line 99 of file test_suites.cpp.

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

#### 7.45.3.9 BOOST_AUTO_TEST_CASE() [9/12]

```
BOOST_AUTO_TEST_CASE (
              s1t9  )
```

pvector overloaded operators test case

Definition at line 111 of file test_suites.cpp.

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

#### 7.45.3.10 BOOST_AUTO_TEST_CASE() [10/12]

```
BOOST_AUTO_TEST_CASE (
              s2t1  )
```

point multiplication test case

Definition at line 133 of file test_suites.cpp.

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

#### 7.45.3.11 BOOST_AUTO_TEST_CASE() [11/12]

```
BOOST_AUTO_TEST_CASE (
              s2t2  )
```

point division test case

Definition at line 144 of file test_suites.cpp.

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

#### 7.45.3.12 BOOST_AUTO_TEST_CASE() [12/12]

```
BOOST_AUTO_TEST_CASE (
              s2t3  )
```

point overloaded operators test case

Definition at line 155 of file test_suites.cpp.

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

# Index