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

1.3 Links

https://videotutorialsrock.com/index.php

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

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

Chapter 2

Todo List

Member wander::wander ()

business logic will be changed

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

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
- void draw (graphics view) override

  agent drawing

Public Attributes

- point position

  position of the agent
- pvector velocity

  velocity of the agent
- point targetPoint

  target of the agent
- float maxSpeed

  maximum speed of the agent
- float maxForce

  maximum force of the agent
- pvector steering

  steering force of the apply
- pvector force

  force of the agent
- pvector acceleration

  acceleration of the agent
- pvector desiredVelocity

  desired velocity of the agent
- float r

  radius of the agent
- int id

  id of the agent
- bool arrive = false

  has arriving behavior or not

6.1.1 Detailed Description

Definition at line 21 of file agent.h.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 agent() `[1/2]`

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

default constructor.

See also

agent(float x, float y)

Definition at line 16 of file agent.cpp.
```
{

}
```

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.

```
{
    position        = point(x, y);
    velocity        = pvector(0.6, 0.0);
    acceleration    = pvector(0.0, 0.0);
    steering        = pvector(0.0, 0.0);
    desiredVelocity = pvector(0.0, 0.0);
    force           = pvector(0.0, 0.0);
    targetPoint     = point(0.0, 0.0);
    entityColor     = RED;
}
```

6.1.2.3 ∼agent()

```
agent::∼agent ( )
```

destructor

Definition at line 79 of file agent.cpp.

```
{

}
```

6.1.3 Member Function Documentation

6.1.3.1 draw()

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

agent drawing

Parameters

view	graphics to draw

Implements entity.

Definition at line 84 of file agent.cpp.

```
                              {
    this->updatePosition(this->arrive);
    view.drawAgent(*this);
}
```

6.1.3.2 getMass()

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

mass attribute getter

Definition at line 29 of file agent.cpp.

```
                     {
    return mass;
}
```

6.1.3.3 getName()

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

name attribute getter

Definition at line 21 of file agent.cpp.

```
                      {
    return name;
}
```

6.1.3.4 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 71 of file agent.cpp.

```
{
     this->maxSpeed = s;
     this->maxForce = f;
```

```
    this->r = r;
    this->mass = m;
}
```

6.1.3.5 setMass()

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

mass attribute setter

Parameters

m	set value

Definition at line 33 of file agent.cpp.

```
                           {
    mass = m;
}
```

6.1.3.6 setName()

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

name attribute setter

Parameters

n	set value

Definition at line 25 of file agent.cpp.

```
                            {
    name = n;
}
```

6.1.3.7 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.

```
{
    force.limit(maxForce);
    acceleration = force;
    velocity += acceleration;

    //arriving behavior implementation
    if(arrive == true){
        pvector diff = targetPoint - position;
        if(diff.magnitude() > r)
            velocity.limit(maxSpeed);
        else
            velocity.limit(maxSpeed * diff.magnitude() / r);
    }
    else{
        velocity.limit(maxSpeed);
    }

    position = position + velocity;
    force = pvector(0,0);
}
```

6.1.4 Member Data Documentation

6.1.4.1 acceleration

`pvector agent::acceleration`

acceleration of the agent

Definition at line 124 of file agent.h.

6.1.4.2 arrive

`bool agent::arrive = false`

has arriving behavior or not

Definition at line 144 of file agent.h.

6.1.4.3 desiredVelocity

`pvector agent::desiredVelocity`

desired velocity of the agent

Definition at line 129 of file agent.h.

6.1.4.4 force

```
pvector agent::force
```

force of the agent

Definition at line 119 of file agent.h.

6.1.4.5 id

```
int agent::id
```

id of the agent

Definition at line 139 of file agent.h.

6.1.4.6 maxForce

```
float agent::maxForce
```

maximum force of the agent

Definition at line 109 of file agent.h.

6.1.4.7 maxSpeed

```
float agent::maxSpeed
```

maximum speed of the agent

Definition at line 104 of file agent.h.

6.1.4.8 position

```
point agent::position
```

position of the agent

Definition at line 89 of file agent.h.

6.1.4.9 r

```
float agent::r
```

radius of the agent

Definition at line 134 of file agent.h.

6.1.4.10 steering

```
pvector agent::steering
```

steering force of the apply

Definition at line 114 of file agent.h.

6.1.4.11 targetPoint

```
point agent::targetPoint
```

target of the agent

Definition at line 99 of file agent.h.

6.1.4.12 velocity

```
pvector agent::velocity
```

velocity of the agent

Definition at line 94 of file agent.h.

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

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

6.2 color Class Reference

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

Public Member Functions

- color ()

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

  constructor.

Static Public Member Functions

- static color getColor (int index)

  gets colorbar colors

Public Attributes

- float R

  portion of red color
- float G

  portion of green color
- float B

  portion of blue color

6.2.1 Detailed Description

Definition at line 23 of file color.h.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 color() [1/2]

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

default constructor.

See also

color(float r, float g, float b)

Definition at line 13 of file color.cpp.

```
{

}
```

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.

```
{
    R = r;
    G = g;
    B = b;
}
```

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.

```
                                {
    switch (index)
    {
       case 0: return WHITE; break;
       case 1: return BLUE; break;
       case 2: return RED; break;
       case 3: return YELLOW; break;
       case 4: return GREEN; break;
       case 5: return BLACK; break;
       case 6: return CYAN; break;
       case 7: return MAGENDA; break;

    }
    return RED;
}
```

6.2.4 Member Data Documentation

6.2.4.1 B

```
float color::B
```

portion of blue color

Definition at line 53 of file color.h.

6.2.4.2 G

```
float color::G
```

portion of green color

Definition at line 48 of file color.h.

6.2.4.3 R

```
float color::R
```

portion of red color

Definition at line 43 of file color.h.

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

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

6.3 entity Class Reference

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

Public Member Functions

- entity ()

  default constructor.
- string getName ()

  getter of the name
- void setName (string name)

  name attribute setter
- int getId ()

  getter of the id attibute
- void setId (int id)

  id attribute setter
- virtual void draw (graphics view)=0

  overriden by child classes

Public Attributes

- color entityColor

  color of the entity

6.3.1 Detailed Description

Definition at line 10 of file entity.h.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 entity()

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

default constructor.

Definition at line 10 of file entity.cpp.

```
                  {
    entityColor = RED;
}
```

6.3.3 Member Function Documentation

6.3.3.1 draw()

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

overriden by child classes

Parameters

view	graphics

Implemented in agent, obstacle, and path.

6.3.3.2 getId()

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

getter of the id attibute

Definition at line 22 of file entity.cpp.

```
                {
    return id;
}
```

6.3.3.3 getName()

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

getter of the name

Definition at line 14 of file entity.cpp.

```
                     {
   return name;
}
```

6.3.3.4 setId()

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

id attribute setter

Parameters

id	setter

Definition at line 26 of file entity.cpp.

```
                       {
    this->id = id;
}
```

6.3.3.5 setName()

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

name attribute setter

Parameters

name	setter

Definition at line 18 of file entity.cpp.

```
                              {
```

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

6.3.4 Member Data Documentation

6.3.4.1 entityColor

```
color entity::entityColor
```

color of the entity

Definition at line 48 of file entity.h.

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

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

6.4 evade Class Reference

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

Public Member Functions

- evade ()

  default constructor.

Static Public Member Functions

- static void loop ()

  loop function of evading scenario

Additional Inherited Members

6.4.1 Detailed Description

Definition at line 15 of file evade.h.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 evade()

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

default constructor.

Definition at line 31 of file evade.cpp.

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

6.4.3 Member Function Documentation

6.4.3.1 loop()

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

loop function of evading scenario

Note

opengl callback forces that function to be static

Definition at line 15 of file evade.cpp.

```
{
    for(auto it = agents.begin(); it < agents.end(); it++){
        if((*it).getName() == "lion"){
            (*it).targetPoint = view.getMousePosition();
            (*it).force  = behavior.seek(*it);
            (*it).arrive = true;
        }
        else{//gazelle
            (*it).force  = behavior.evade(agents, *it, view, "lion");
        }
    }

    refresh();
}
```

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

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

6.5 flee Class Reference

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

Public Member Functions

- flee ()

    default constructor.

Static Public Member Functions

- static void loop ()

  evading scenario loop function

Additional Inherited Members

6.5.1 Detailed Description

Definition at line 14 of file flee.h.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 flee()

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

default constructor.

Definition at line 24 of file flee.cpp.

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

6.5.3 Member Function Documentation

6.5.3.1 loop()

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

evading scenario loop function

Note

opengl callback forces that function to be static

Definition at line 15 of file flee.cpp.

```
{
    for(auto it = agents.begin(); it < agents.end(); it++){
        (*it).force = behavior.flee((*it), view, view.getMousePosition());
    }

    refresh();
}
```

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

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

6.6 flock Class Reference

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

Public Member Functions

- flock ()

  default constructor.

Static Public Member Functions

- static void loop ()

  flocking scenario loop function

Additional Inherited Members

6.6.1 Detailed Description

Definition at line 15 of file flock.h.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 flock()

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

default constructor.

Definition at line 38 of file flock.cpp.

```
{
    int agentCount = 50;
    float maxForce = 0.3;
    float maxSpeed = 0.8;
    name = "flocking agents";
    createAgent(RANDOM, &agentCount, &maxForce, &maxSpeed);
    callback = reinterpret_cast <void(*)()> ( (void *)(&loop) );
}
```

6.6.3 Member Function Documentation

6.6.3.1 loop()

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

flocking scenario loop function

Note

opengl callback forces that function to be static

Definition at line 15 of file flock.cpp.

```
{
    for(auto it = agents.begin(); it < agents.end(); it++){
        view.forceInScreen((*it));

        pvector sep = behavior.separation(agents, *it);
        sep.mul(1.5);
        pvector ali = behavior.align(agents, *it);
        ali.mul(4);
        pvector coh = behavior.cohesion(agents, *it);
        coh.mul(0.1);

        (*it).targetPoint = view.getMousePosition();
        pvector seek  = behavior.seek(*it);
        seek.mul(0.01);

        (*it).force = sep + ali + coh + seek;
        (*it).arrive = true;
    }

    refresh();
}
```

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

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

6.7 flowField Class Reference

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

Public Member Functions

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

6.7.1 Detailed Description

Definition at line 18 of file flowField.h.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 flowField() [1/2]

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

default constructor.

See also

flowField(pvector p)

Definition at line 15 of file flowField.cpp.

```
{

}
```

6.7.2.2 flowField() [2/2]

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

constructor.

Parameters

p	force vector

See also

flowField()

Definition at line 10 of file flowField.cpp.

```
{
    createFlowField(p);
}
```

6.7.3 Member Function Documentation

6.7.3.1 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.

```
{
    return uniformField[x][y];
}
```

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

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

6.8 graphics Class Reference

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

Public Member Functions

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

Static Public Member Functions

- static void timerEvent (int value)

    periodic timer event
- static void handleKeypress (unsigned char key, int x, int y)

    key press event
- static void mouseButton (int button, int state, int x, int y)

    mouse press event
- static void handleResize (int w, int h)

    event triggered with screen resizing
- static void mouseMove (int x, int y)

    event triggered with mouse movements

Static Public Attributes

- static int target_x = -WIDTH

    mouse position x
- static int target_y = HEIGHT

    mouse position y

6.8.1 Detailed Description

Definition at line 22 of file graphics.h.

6.8.2 Member Function Documentation

6.8.2.1 drawAgent()

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

drawing with corresponding angle

Parameters

agent	instance to change

Definition at line 159 of file graphics.cpp.

```
{
    glPushMatrix();
    glTranslatef(agent.position.x, agent.position.y, 0.0f);
    glRotatef(agent.velocity.getAngle(), 0.0f, 0.0f, 1.0f);
    glBegin(GL_TRIANGLES);
    glColor3f( agent.entityColor.R, agent.entityColor.G, agent.entityColor.B);
    glVertex3f( 1.0f,  0.0f, 0.0f);
    glVertex3f(-1.0f,  0.5f, 0.0f);
    glVertex3f(-1.0f, -0.5f, 0.0f);
    glEnd();
    glPopMatrix();
}
```

6.8.2.2 drawCircle()

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

draws circle

Parameters

p	center of the circle
radius	radius of the circle
color	of the circle

Definition at line 136 of file graphics.cpp.

```
{
   glColor3f(color.R, color.G, color.B);
   glBegin(GL_LINE_STRIP);
   glLineWidth(2);
   for (int i = 0; i <= 300; i++) {
     float angle = 2 * PI * i / 300;
     float x = cos(angle) * radius;
     float y = sin(angle) * radius;
     glVertex2d(p.x + x, p.y + y);
   }
   glEnd();
}
```

6.8.2.3 drawLine()

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

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.

```
{
    glColor3f( cl.R, cl.G, cl.B);
    glLineWidth(2);
    glBegin(GL_LINES);
    glVertex2f(p1.x, p1.y);
    glVertex2f(p2.x, p2.y);
    glEnd();
}
```

6.8.2.4 drawPath()

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

draws path

Parameters

path	to draw

Definition at line 112 of file graphics.cpp.

```
{
    point p1, p2;
    for(auto it = path.points.begin(); it < path.points.end()-1; it++){
        p1 = point((*it).x, (*it).y - path.width/2) ;
        p2 = point((*(it+1)).x, (*(it+1)).y - path.width/2);
        drawLine(p1, p2, path.entityColor);

        p1 = point((*it).x, (*it).y + path.width/2) ;
        p2 = point((*(it+1)).x, (*(it+1)).y + path.width/2);
        drawLine(p1, p2, path.entityColor);
    }
}
```

6.8.2.5 drawPoint()

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

draws point

Parameters

p	point to draw

Definition at line 150 of file graphics.cpp.

```
{
    glColor3f(1,1,1);
    glPointSize(4.0);
    glBegin(GL_POINTS);
    glVertex2f(p.x, p.y);
    glEnd();
}
```

6.8.2.6 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.

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

6.8.2.7 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.

```
{
    if(agent.position.x > WIDTH)
       agent.position.x -= 2 * WIDTH;
    if(agent.position.x < -WIDTH)
       agent.position.x += 2 * WIDTH;
    if(agent.position.y > HEIGHT)
       agent.position.y -= 2 * HEIGHT;
    if(agent.position.y < -HEIGHT)
       agent.position.y += 2 * HEIGHT;
}
```

6.8.2.8 getMousePosition()

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

gets mouse position

Returns

mouse point

Definition at line 56 of file graphics.cpp.

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

6.8.2.9 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.

```
{
    if (key == ESC){
        exit(0);
    }
}
```

6.8.2.10 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.

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

6.8.2.11 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.

```
{
    glutInit(argv, argc);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(400, 400);
    glutCreateWindow("Autonomous Steering Agents");
    glClearColor(0.7f, 0.7f, 0.7f, 1.0f); //set background color
    glEnable(GL_DEPTH_TEST);
    glutDisplayFunc(*callback);
    glutMouseFunc(graphics::mouseButton);
    glutPassiveMotionFunc(graphics::mouseMove);
    glutKeyboardFunc(graphics::handleKeypress);
    glutReshapeFunc(graphics::handleResize);
    glutTimerFunc(20, graphics::timerEvent, 0);
    glutMainLoop();
}
```

6.8.2.12 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.

```
{
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN){
    }
}
```

6.8.2.13 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.

```
{
    //TODO: mouse position to glut
    //TODO: magic numbers
    graphics::target_x = x / 5.88 - 34;
    graphics::target_y = 34 - y / 5.88;
}
```

6.8.2.14 refreshScene()

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

update agent position

Definition at line 30 of file graphics.cpp.

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

6.8.2.15 timerEvent()

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

periodic timer event

Parameters

value	period as ms

Definition at line 93 of file graphics.cpp.

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

6.8.3 Member Data Documentation

6.8.3.1 target_x

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

mouse position x

Definition at line 130 of file graphics.h.

6.8.3.2 target_y

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

mouse position y

Definition at line 135 of file graphics.h.

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

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

6.9 leaderFollower Class Reference

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

Public Member Functions

- leaderFollower ()

  default constructor.

Static Public Member Functions

- static void loop ()

  leader following scenario loop function

Static Public Attributes

- static pvector leaderVelocity

  leader velocity
- static point leaderPosition

  leader position

Additional Inherited Members

6.9.1 Detailed Description

Definition at line 14 of file leaderFollower.h.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 leaderFollower()

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

default constructor.

Definition at line 53 of file leaderFollower.cpp.

```
{
    int agentCount = 10;
    float maxForce = 0.4;
    float maxSpeed = 0.4;
    name = "leader following";

    //todo: refactor leader creation
    agent agent1 {-10.0,  0.0};
    agent1.id = 1;
    agent1.setName("leader");
    agent1.entityColor = BLUE;
    agent1.setFeatures(0.8, 0.4, 5, 1);
    agents.push_back(agent1);


    createAgent(RANDOM, &agentCount, &maxForce, &maxSpeed);
    callback = reinterpret_cast <void(*)()> ( (void *)(&loop) );
}
```

6.9.3 Member Function Documentation

6.9.3.1 loop()

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

leader following scenario loop function

Note

opengl callback forces that function to be static

Definition at line 18 of file leaderFollower.cpp.

```
{
    for(auto it = agents.begin(); it < agents.end(); it++){
        if((*it).getName() == "leader"){
            (*it).targetPoint = view.getMousePosition();
            (*it).force  = behavior.seek(*it);
            leaderVelocity = (*it).velocity;
            leaderVelocity.mul(-1);
            leaderVelocity.normalize().mul(20);
            leaderPosition = (*it).position;
        }
        else{
            (*it).targetPoint = leaderPosition + leaderVelocity;
            view.drawCircle((*it).targetPoint, 8, RED);

            pvector sep = behavior.separation(agents, *it);
            sep.mul(15);
            (*it).force = sep;

            pvector leaderDiff = (*it).position - leaderPosition;
            if(leaderDiff.magnitude() < 5){
                pvector fle = behavior.evade(agents, *it, view, "leader");
                fle.mul(40);
                (*it).force += fle;
            }

            pvector diff = (*it).position - (*it).targetPoint;
            if(diff.magnitude() > 5) { (*it).force   += behavior.seek(*it); }
            else                     { (*it).velocity = pvector(0,0);       }
        }
        (*it).arrive = true;
    }
    refresh();
}
```

6.9.4 Member Data Documentation

6.9.4.1 leaderPosition

`point leaderFollower::leaderPosition  [static]`

leader position

Definition at line 24 of file leaderFollower.h.

6.9.4.2 leaderVelocity

`pvector leaderFollower::leaderVelocity  [static]`

leader velocity

Definition at line 19 of file leaderFollower.h.

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

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

6.10 mouseFollower Class Reference

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

Public Member Functions

- mouseFollower ()

  default constructor.

Static Public Member Functions

- static void loop ()

  mouse following scenario loop function

Additional Inherited Members

6.10.1 Detailed Description

Definition at line 14 of file mouseFollower.h.

6.10.2 Constructor & Destructor Documentation

6.10.2.1 mouseFollower()

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

default constructor.

Definition at line 25 of file mouseFollower.cpp.

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

6.10.3 Member Function Documentation

6.10.3.1 loop()

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

mouse following scenario loop function

Note

opengl callback forces that function to be static

Definition at line 15 of file mouseFollower.cpp.

```
{
    for(auto it = agents.begin(); it < agents.end(); it++){
       (*it).targetPoint = view.getMousePosition();
       (*it).force  = behavior.seek(*it);
       (*it).arrive = true;
    }
    refresh();
}
```

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

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

6.11 obstacle Class Reference

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

Public Member Functions

- obstacle ()

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

  constructor
- void draw (graphics view) override

  overriden draw implementation

Public Attributes

- point p

  center point of the obstacle
- float r

  radius of the obstacle

6.11.1 Detailed Description

Definition at line 15 of file obstacle.h.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 obstacle() [1/2]

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

default constructor.

See also

obstacle(point p, float r

Definition at line 16 of file obstacle.cpp.

```
{
   p = point(0,0);
   r = 5;
   entityColor = RED;
}
```

6.11.2.2 obstacle() [2/2]

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

constructor

Parameters

p	center of the circular obstacle
r	radius of the obstacle

See also

obstacle(point p, float r);

Definition at line 23 of file obstacle.cpp.

```
{
   this->p = p;
   this->r = r;
   entityColor = RED;
}
```

6.11.3 Member Function Documentation

6.11.3.1 draw()

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

overriden draw implementation

Implements entity.

Definition at line 30 of file obstacle.cpp.

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

6.11.4 Member Data Documentation

6.11.4.1 p

```
point obstacle::p
```

center point of the obstacle

Definition at line 34 of file obstacle.h.

6.11.4.2 r

```
float obstacle::r
```

radius of the obstacle

Definition at line 39 of file obstacle.h.

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

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

6.12 obstacleAvoidance Class Reference

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

Public Member Functions

- obstacleAvoidance ()

    default constructor.

Static Public Member Functions

- static void loop ()

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

    creation of list of obstacles

Static Public Attributes

- static vector< obstacle > obstacles

    list of obstacles

Additional Inherited Members

6.12.1 Detailed Description

Definition at line 15 of file obstacleAvoidance.h.

6.12.2 Constructor & Destructor Documentation

6.12.2.1 obstacleAvoidance()

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

default constructor.

Definition at line 42 of file obstacleAvoidance.cpp.

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

6.12.3 Member Function Documentation

6.12.3.1 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 35 of file obstacleAvoidance.cpp.

```
{
    obstacles.push_back(obstacle(point(0,0), 8));
    obstacles.push_back(obstacle(point(-20,0), 3));
    obstacles.push_back(obstacle(point(20,-10), 4));
}
```

6.12.3.2 loop()

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

obstacle avoidance scenario loop function

Note

opengl callback forces that function to be static

Definition at line 17 of file obstacleAvoidance.cpp.

```
{
    for(auto it = agents.begin(); it < agents.end(); it++){
        (*it).targetPoint = view.getMousePosition();
        pvector seek  = behavior.seek(*it);
        seek.mul(0.5);

        pvector avoid = behavior.avoid(obstacles, *it);
        (*it).force = avoid + seek;
        (*it).arrive = true;

        for(auto it = obstacles.begin(); it < obstacles.end(); it++){
            (*it).draw(view);
        }
    }
    refresh();
}
```

6.12.4 Member Data Documentation

6.12.4.1 obstacles

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

list of obstacles

Note

opengl callback forces that function to be static

Definition at line 32 of file obstacleAvoidance.h.

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

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

6.13 path Class Reference

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

Public Member Functions

- path ()

    default constructor.
- path (float width)

    donstructor.
- void addPoint (point p)

    adds a new point to the path
- void draw (graphics view)

    overriden draw implementation

Public Attributes

- vector< point > points

    list of points added to the path
- int width

    width of the path

6.13.1 Detailed Description

Definition at line 17 of file path.h.

6.13.2 Constructor & Destructor Documentation

6.13.2.1 path() [1/2]

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

default constructor.

See also

path(float width)

Definition at line 16 of file path.cpp.

```
{
   entityColor = BLUE;
   width = 8;
}
```

6.13.2.2 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.

```
{
    this->width = width;
    entityColor = BLUE;
}
```

6.13.3 Member Function Documentation

6.13.3.1 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.

```
{
    points.push_back(p);
}
```

6.13.3.2 draw()

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

overriden draw implementation

Implements entity.

Definition at line 28 of file path.cpp.

```
                              {
    view.drawPath(*this);
}
```

6.13.4 Member Data Documentation

6.13.4.1 points

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

list of points added to the path

Definition at line 41 of file path.h.

6.13.4.2 width

```
int path::width
```

width of the path

Definition at line 46 of file path.h.

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

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

6.14 pathFollower Class Reference

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

Public Member Functions

- pathFollower ()

  default constructor.

Static Public Member Functions

- static void loop ()

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

  creates path

Static Public Attributes

- static path myPath

  path that will be followed

Additional Inherited Members

6.14.1 Detailed Description

Definition at line 14 of file pathFollower.h.

6.14.2 Constructor & Destructor Documentation

6.14.2.1 pathFollower()

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

default constructor.

Definition at line 38 of file pathFollower.cpp.
```
{
    int agentCount = 40;
    float maxForce = 0.2;
    float maxSpeed = 0.4;
    myPath = path(8);
    createPath(myPath);
    name = "path following";
    createAgent(RANDOM, &agentCount, &maxForce, &maxSpeed);
    callback = reinterpret_cast <void(*)()> ( (void *)(&loop) );
}
```

6.14.3 Member Function Documentation

6.14.3.1 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 30 of file pathFollower.cpp.
```
{
    p.addPoint(point(-40,  5));
    p.addPoint(point(-14, 15));
    p.addPoint(point( 10,  7));
    p.addPoint(point( 40, 12));
}
```

6.14.3.2 loop()

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

path follower scenario loop function

Note

opengl callback forces that function to be static

Definition at line 17 of file pathFollower.cpp.

```
{
    for(auto it = agents.begin(); it < agents.end(); it++){
        pvector flwpth = behavior.stayInPath(*it, myPath, view);
        pvector sep  = behavior.separation(agents, *it);
        sep.mul(5);
        (*it).force = sep + flwpth;

        myPath.draw(view);
    }
    refresh();
}
```

6.14.4 Member Data Documentation

6.14.4.1 myPath

`path pathFollower::myPath  [static]`

path that will be followed

Note

opengl callback forces that function to be static

Definition at line 38 of file pathFollower.h.

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

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

6.15 point Class Reference

`#include <point.h>`

Public Member Functions

- point ()

  default constructor
- point (float x, float y)

  constructor
- void div (float d)

  divide point
- void mul (float d)

  multiply point
- void print (const string &s)

  debug function
- void getNormalPoint (point predicted, point start, point end)

  provides normal point on a vector of a point
- point operator+ (pvector const &obj)

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

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

  overloaded - operator
- bool operator== (point const &obj)

  overloaded == operator

Public Attributes

- float x

  x position
- float y

  y position

6.15.1 Detailed Description

Definition at line 15 of file point.h.

6.15.2 Constructor & Destructor Documentation

6.15.2.1 point() [1/2]

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

default constructor

See also

point(float x, float y)

Definition at line 21 of file point.cpp.

```
{
   x = 0;
   y = 0;
}
```

6.15.2.2 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.

```
{
    this->x = x;
    this->y = y;
}
```

6.15.3 Member Function Documentation

6.15.3.1 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.

```
{
    x = x / d;
    y = y / d;
}
```

6.15.3.2 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.

```
{
    pvector a = predicted - start;
    pvector b = end - start;
    b.normalize();
    float a_dot_b = a.dotProduct(b);
    b.mul(a_dot_b);
    point normalPoint = start + b;
    this->x = normalPoint.x;
    this->y = normalPoint.y;
}
```

6.15.3.3 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.

```
{
    x = x * d;
    y = y * d;
}
```

6.15.3.4 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.

```
{
    point res;
    res.x = x + obj.x;
    res.y = y + obj.y;
    return res;
}
```

6.15.3.5 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.

```
{
    point res;
    res.x = x + obj.x;
    res.y = y + obj.y;
    return res;
}
```

6.15.3.6 operator-()

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

overloaded - operator

Parameters

obj	point to substract

Returns

difference

Definition at line 63 of file point.cpp.

```
{
   pvector res;
   res.x = x - obj.x;
   res.y = y - obj.y;
   return res;
}
```

6.15.3.7 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.

```
{
   if(x == obj.x && y == obj.y)
      return true;
   return false;
}
```

6.15.3.8 print()

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

debug function

Parameters

s	explanation string of the log

Definition at line 83 of file point.cpp.

```
{
   cout « " " « s « " " « x « " " « y « endl;
}
```

6.15.4 Member Data Documentation

6.15.4.1 x

```
float point::x
```

x position

Definition at line 88 of file point.h.

6.15.4.2 y

```
float point::y
```

y position

Definition at line 93 of file point.h.

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

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

6.16 prison Class Reference

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

Public Member Functions

- prison ()

  default constructor.

Static Public Member Functions

- static void loop ()

  prisoning scenario loop function

Additional Inherited Members

6.16.1 Detailed Description

Definition at line 15 of file prison.h.

6.16.2 Constructor & Destructor Documentation

6.16.2.1 prison()

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

default constructor.

Definition at line 31 of file prison.cpp.

```
{
    int agentCount = 30;
    float maxForce = 0.6;
    float maxSpeed = 0.6;

    name = "stay in prison";
    createAgent(RANDOM, &agentCount, &maxForce, &maxSpeed);
    callback = reinterpret_cast <void(*)()> ( (void *)(&loop) );
}
```

6.16.3 Member Function Documentation

6.16.3.1 loop()

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

prisoning scenario loop function

prison loop function

Note

opengl callback forces that function to be static

Definition at line 18 of file prison.cpp.

```
{
    for(auto it = agents.begin(); it < agents.end(); it++){
        view.drawLine(point(-WALL,  WALL), point( WALL,  WALL), BLUE);
        view.drawLine(point( WALL,  WALL), point( WALL, -WALL), BLUE);
        view.drawLine(point( WALL, -WALL), point(-WALL, -WALL), BLUE);
        view.drawLine(point(-WALL,  WALL), point(-WALL, -WALL), BLUE);
        (*it).force  = behavior.stayInArea(*it, WALL - DISTANCE);
        (*it).force += behavior.separation(agents, *it);
    }
    refresh();
}
```

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

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

6.17 pursuit Class Reference

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

Public Member Functions

- pursuit ()

  default constructor.

Static Public Member Functions

- static void loop ()

  pursuing scenario loop function

Additional Inherited Members

6.17.1 Detailed Description

Definition at line 14 of file pursuit.h.

6.17.2 Constructor & Destructor Documentation

6.17.2.1 pursuit()

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

default constructor.

Definition at line 31 of file pursuit.cpp.

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

6.17.3 Member Function Documentation

6.17.3.1 loop()

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

pursuing scenario loop function

Note

opengl callback forces that function to be static

Definition at line 15 of file pursuit.cpp.

```
{
    for(auto it = agents.begin(); it < agents.end(); it++){
        if((*it).getName() == "gazelle"){
            (*it).targetPoint = view.getMousePosition();
            (*it).force  = behavior.seek(*it);
        }
        else{//lion
            (*it).force  = behavior.pursuit(agents, *it, view, "gazelle");
        }
        (*it).arrive = true;
    }

    refresh();
}
```

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

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

6.18 pvector Class Reference

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

Public Member Functions

- pvector ()

  default constructor
- pvector (float x, float y)

  constructor
- float magnitude ()

  calculates magnitude of the vector
- pvector & normalize ()

  normalize
- void div (float i)

  vector division
- void mul (float i)

  vector multiplication
- void add (pvector p)

  addition of vectors
- void limit (float limit)

  vector limitation
- float getAngle ()

calculates vector angle
- float dotProduct (pvector v)

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

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

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

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

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

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

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

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

  overloaded == operator

Public Attributes

- float x

  x magnitude of the vector
- float y

  y magnitude of the vector

6.18.1 Detailed Description

Definition at line 17 of file pvector.h.

6.18.2 Constructor & Destructor Documentation

6.18.2.1 pvector() [1/2]

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

default constructor

See also

pvector(float x, float y)

Definition at line 35 of file pvector.cpp.
```
{
    x = 0;
    y = 0;
}
```

6.18.2.2 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.

```
{
    this->x = x;
    this->y = y;
}
```

6.18.3 Member Function Documentation

6.18.3.1 add()

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

addition of vectors

Parameters

p	vector to add

Definition at line 59 of file pvector.cpp.

```
{
    x = x + p.x;
    y = y + p.y;
}
```

6.18.3.2 angleBetween()

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

angle calculation between two vectors

Parameters

v	vector to calculate angle

Returns

angle

Definition at line 23 of file pvector.cpp.

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

6.18.3.3 div()

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

vector division

Parameters

i	scalar value to divide

Definition at line 47 of file pvector.cpp.

```
{
    x = x / i;
    y = y / i;
}
```

6.18.3.4 dotProduct()

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

dot product of two vectors

Parameters

v	vector to calculate dot product

Returns

returns scalar dot product

Definition at line 30 of file pvector.cpp.

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

6.18.3.5 getAngle()

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

calculates vector angle

Returns

angle

Definition at line 16 of file pvector.cpp.

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

6.18.3.6 limit()

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

vector limitation

Parameters

limit	value to restrict vector magnitude

Definition at line 84 of file pvector.cpp.

```
{
    this->normalize();
    this->mul(limit);
}
```

6.18.3.7 magnitude()

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

calculates magnitude of the vector

Returns

magnitude of the vector

Definition at line 65 of file pvector.cpp.

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

6.18.3.8 mul()

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

vector multiplication

Parameters

i	scalar value to multiply

Definition at line 53 of file pvector.cpp.

```
{
   x = x * i;
   y = y * i;
}
```

6.18.3.9 normalize()

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

normalize

Returns

normalized vector

Definition at line 70 of file pvector.cpp.

```
{
   float magnitude = this->magnitude();
   if(magnitude != 0){
      this->x = this->x / magnitude;
      this->y = this->y / magnitude;
   }
   else{
      this->x = 0;
      this->y = 0;
   }
   return *this;
}
```

6.18.3.10 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.

```
{
   pvector res;
   res.x = x + obj.x;
   res.y = y + obj.y;
   return res;
}
```

6.18.3.11 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.

```
{
   pvector res;
   res.x = x + obj.x;
   res.y = y + obj.y;
   return res;
}
```

6.18.3.12 operator+=()

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

overloaded += operator

Parameters

obj	vector to add

Returns

sum

Definition at line 98 of file pvector.cpp.

```
{
    x = x + obj.x;
    y = y + obj.y;
    return *this;
}
```

6.18.3.13 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.

```
{
    pvector res;
    res.x = x - obj.x;
    res.y = y - obj.y;
    return res;
}
```

6.18.3.14 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.

```
{
    pvector res;
    res.x = x - obj.x;
    res.y = y - obj.y;
    return res;
}
```

6.18.3.15 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.

```
{
    if(x == obj.x && y == obj.y)
        return true;
    return false;
}
```

6.18.3.16 print()

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

debug function

Parameters

s	identification text

Definition at line 128 of file pvector.cpp.

```
{
    cout « s « " " « x « " " « y « endl;
}
```

6.18.4 Member Data Documentation

6.18.4.1 x

```
float pvector::x
```

x magnitude of the vector

Definition at line 140 of file pvector.h.

6.18.4.2 y

```
float pvector::y
```

y magnitude of the vector

Definition at line 145 of file pvector.h.

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

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

6.19 random Class Reference

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

Static Public Member Functions

- static void createRandomArray (int ∗arr, int size)

  random array generation

6.19.1 Detailed Description

Definition at line 9 of file random.h.

6.19.2 Member Function Documentation

6.19.2.1 createRandomArray()

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

random array generation

Parameters

arr	struct that includes random values
size	of the array

Definition at line 14 of file random.cpp.

```
                                             {
    srand(time(NULL));
    for(int i=0; i<size; i++)
       arr[i] = i+1;

    for (int i=0; i < size; i++){
       int r = rand() % size;
       swap(arr[i], arr[r]);
    }
}
```

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

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

6.20 scenario Class Reference

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

Public Member Functions

- scenario ()

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

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

  graphics initialization

Static Public Member Functions

- static void refresh ()

  refreshes all items

Public Attributes

- void(∗ callback)()

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

Static Public Attributes

- static vector< agent > agents

  structure stores agents
- static graphics view

  graphics instance used
- static steeringBehavior behavior

  behavior instance used
- static string name

  scenario name

6.20.1 Detailed Description

Definition at line 19 of file scenario.h.

6.20.2 Constructor & Destructor Documentation

6.20.2.1 scenario()

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

default constructor.

Definition at line 28 of file scenario.cpp.

```
{
    view = graphics();
}
```

6.20.3 Member Function Documentation

6.20.3.1 createAgent()

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

agent creation

Parameters

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 109 of file scenario.cpp.

```
{
    if(type == TROOP){
        createTroop(*count);
    }
    else if(type == RANDOM){
        createRandomAgents(*count, *force, *speed);
    }
    else if(type == STATIC){
        createStaticAgents("gazelle", "lion");
    }
    else{
        //error message
    }
}
```

6.20.3.2 initGL()

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

graphics initialization

Parameters

argv	list of user arguments
argc	number of user arguments

Definition at line 23 of file scenario.cpp.

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

6.20.3.3 refresh()

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

refreshes all items

Note

opengl callback forces that function to be static

Definition at line 33 of file scenario.cpp.

```
{
    point textPosition = point(-34, 32.25);

    for(auto it = agents.begin(); it < agents.end(); it++){
       (*it).draw(view);
    }

    view.drawText(name, textPosition);
    view.refreshScene();
}
```

6.20.4 Member Data Documentation

6.20.4.1 agents

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

structure stores agents

Note

opengl callback forces that function to be static

Definition at line 52 of file scenario.h.

6.20.4.2 behavior

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

behavior instance used

Note

opengl callback forces that function to be static

Definition at line 64 of file scenario.h.

6.20.4.3 callback

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

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

Definition at line 75 of file scenario.h.

6.20.4.4 name

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

scenario name

Note

opengl callback forces that function to be static

Definition at line 70 of file scenario.h.

6.20.4.5 view

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

graphics instance used

Note

opengl callback forces that function to be static

Definition at line 58 of file scenario.h.

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

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

6.21 steeringBehavior Class Reference

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

Public Member Functions

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

6.21.1 Detailed Description

Definition at line 36 of file steeringBehavior.h.

6.21.2 Member Function Documentation

6.21.2.1 align()

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

force to align

Parameters

agent	to be aligned
boids	list of all the agents

Returns

force to be applied

Definition at line 120 of file steeringBehavior.cpp.

```
{
   float neighborDist = 30;
   pvector sum {0,0};
   int count = 0;
   for(auto it = boids.begin(); it < boids.end(); it++){
      float d = (agent.position - (*it).position).magnitude();
      if( (d >0) && (d < neighborDist) ){
         sum += (*it).velocity;
         count++;
      }
   }
   if(count>0){
      sum.div(count);
      sum.normalize().mul(agent.maxSpeed);
      agent.steering = sum - agent.velocity;
      return agent.steering;
   }
   return pvector(0,0);
}
```

6.21.2.2 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 184 of file steeringBehavior.cpp.

```
{
   float dynamic_length = agent.velocity.magnitude() / agent.maxSpeed;
   pvector vel = agent.velocity;
   vel.normalize().mul(dynamic_length);
   pvector ahead  = vel + agent.position;
   vel.mul(6);
   pvector ahead2 = vel + agent.position;
   //view.drawPoint(point(ahead.x, ahead.y));
   //view.drawPoint(point(ahead2.x, ahead2.y));

   for(auto it = obstacles.begin(); it < obstacles.end(); it++){
      float dist  = (ahead  - (*it).p).magnitude();
      float dist2 = (ahead2 - (*it).p).magnitude();
      if(dist < (*it).r + 2 || dist2 < (*it).r + 2){
         pvector avoidance = ahead - (*it).p;
         avoidance.normalize().mul(20);
         /*a = point(avoidance.x, avoidance.y);
         view.drawLine(agent.position, agent.position + a, color(0,1,0));*/
         return avoidance;
      }
   }
   return pvector(0,0);
}
```

6.21.2.3 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 141 of file steeringBehavior.cpp.

```
{
   float neighborDist = 20;
   point sum {0,0};
   int count = 0;
   for(auto it = boids.begin(); it < boids.end(); it++){
      float d = (agent.position - (*it).position).magnitude();
      if( (d >0) && (d < neighborDist) ){
            sum = sum + (*it).position;
            count++;
        }
    }
    if(count>0){
        sum.div(count);
        agent.targetPoint = sum;
        return seek(agent);
    }
    return pvector(0,0);
}
```

6.21.2.4 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.

```
{
    agent target;
    for(auto it = boids.begin(); it < boids.end(); it++){
        if((*it).getName() == name){
            target = *it;
        }
    }

    point p = point(evader.position.x + 2, evader.position.y - 2);
    //view.drawText(evader.getName(), p);
    p = point(target.position.x + 2, target.position.y - 2);
    //view.drawText(target.getName(), p);

    pvector targetVel = target.velocity;
    targetVel.mul(5);//TODO: magic number

    point futurePos = target.position + targetVel;
    //view.drawPoint(futurePos);

    pvector dist = evader.position - futurePos;
    dist.normalize().mul( 1 / dist.magnitude() );

    evader.targetPoint = evader.position + dist;
    return flee(evader, view, futurePos);
}
```

6.21.2.5 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.

```
{
   int radius = 15;

   pvector dist = agent.targetPoint - p;
   //view.drawPoint(agent.targetPoint);

   if(dist.magnitude() < radius){
      agent.arrive = false;
      agent.desiredVelocity = agent.position - p;
   }
   else{
      agent.arrive = true;
      agent.desiredVelocity = agent.targetPoint - agent.position;
   }
   agent.steering = agent.desiredVelocity - agent.velocity;
   return agent.steering;
}
```

6.21.2.6 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 239 of file steeringBehavior.cpp.

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

6.21.2.7 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.

```
{
   agent target;
   for(auto it = boids.begin(); it < boids.end(); it++){
      if((*it).getName() == name){
         target = *it;
      }
   }

   point p = point(target.position.x + 2, target.position.y - 2);
   view.drawText(target.getName(), p);

   p = point(pursuer.position.x + 2, pursuer.position.y - 2);
   view.drawText(pursuer.getName(), p);

   float dist = (target.position - pursuer.position).magnitude();
   float t = dist / target.maxSpeed;

   pvector targetVel = target.velocity;
   targetVel.mul(t);
   point futurePos = target.position + targetVel;
   pursuer.targetPoint = futurePos;
   return seek(pursuer);
}
```

6.21.2.8 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 209 of file steeringBehavior.cpp.
```
{
   agent.desiredVelocity = agent.targetPoint - agent.position;
   agent.steering = agent.desiredVelocity - agent.velocity;
   return agent.steering;
}
```

6.21.2.9 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 161 of file steeringBehavior.cpp.
```
{
   float desiredSeparation = 3;
   pvector sum = pvector(0,0);
   int count = 0;
   for(auto it = agents.begin(); it < agents.end(); it++){
      float d = (agent.position - (*it).position).magnitude();
      if( (d > 0) && (d < desiredSeparation) ){
         pvector diff = agent.position - (*it).position;
         diff.normalize().div(d);
         sum = sum + diff;
         count++;
      }
   }
   if(count > 0){
      sum.div(count);
      sum.normalize().mul(agent.maxSpeed);
      agent.steering = sum - agent.velocity;
      return agent.steering;
   }
   return pvector(0,0);
}
```

6.21.2.10 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.

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

6.21.2.11 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 248 of file steeringBehavior.cpp.

```
{
   if(agent.position.x >= turnPoint){
      agent.desiredVelocity = pvector( -agent.maxSpeed, agent.velocity.y );
      agent.steering = agent.desiredVelocity - agent.velocity;
      return agent.steering;
   }
   else if(agent.position.x <= -turnPoint){
      agent.desiredVelocity = pvector( agent.maxSpeed, agent.velocity.y );
      agent.steering = agent.desiredVelocity - agent.velocity;
      return agent.steering;
   }
   else if(agent.position.y >= turnPoint){
      agent.desiredVelocity = pvector( agent.velocity.x, -agent.maxSpeed );
      agent.steering = agent.desiredVelocity - agent.velocity;
      return agent.steering;
   }
   else if(agent.position.y <= -turnPoint){
      agent.desiredVelocity = pvector( agent.velocity.x, agent.maxSpeed );
      agent.steering = agent.desiredVelocity - agent.velocity;
      return agent.steering;
   }
   return pvector(0,0);
}
```

6.21.2.12 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 216 of file steeringBehavior.cpp.

```
{
   float worldRecord = 1000000;
   point normalPoint, predictedPos, start, end;
   pvector distance;
   for(auto it = path.points.begin(); it < path.points.end()-1; it++){
      start = point((*it).x, (*it).y);
      end   = point((*(it+1)).x, (*(it+1)).y);
      predictedPos = agent.position + agent.velocity;
      normalPoint.getNormalPoint(predictedPos, start, end);
      if (normalPoint.x < start.x || normalPoint.x > end.x){
         normalPoint = end;
      }
      distance = predictedPos - normalPoint;
      if (distance.magnitude() < worldRecord){
         worldRecord = distance.magnitude();
         agent.targetPoint = end;
      }
      view.drawPoint(agent.targetPoint);
   }
   return seek(agent);
}
```

6.21.2.13 wander()

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

force to wander

Parameters

agent	agent that will wander

Returns

force to be applied

Definition at line 99 of file steeringBehavior.cpp.

```
{
   pvector circleCenter = agent.velocity;
   circleCenter.normalize().mul(CIRCLE_DISTANCE + CIRCLE_RADIUS);

   int wanderAngle = (rand() % 360);
   pvector displacement {0, 1};
   setAngle(displacement, wanderAngle);
   displacement.mul(CIRCLE_RADIUS);

   agent.desiredVelocity = displacement + circleCenter;
   agent.steering = agent.desiredVelocity - agent.velocity;

   //move it to the center when it is out of screen
   if(agent.position.x > WIDTH || agent.position.x < -WIDTH ||
      agent.position.y > HEIGHT || agent.position.y < -HEIGHT)
      agent.position = point(0,0);

   return agent.steering;
}
```

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

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

6.22 wander Class Reference

`#include <wander.h>`

Public Member Functions

- wander ()

  default constructor

Static Public Member Functions

- static void loop ()

  wander scenario loop function

Additional Inherited Members

6.22.1 Detailed Description

Definition at line 14 of file wander.h.

6.22.2 Constructor & Destructor Documentation

6.22.2.1 wander()

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

default constructor

Todo business logic will be changed

Definition at line 24 of file wander.cpp.

```
{
    int agentCount = 30;
    float maxForce = 0.3;
    float maxSpeed = 0.6;

    name = "wandering objects";
    createAgent(RANDOM, &agentCount, &maxForce, &maxSpeed);
    callback = reinterpret_cast <void(*)()> ( (void *)(&loop) );
}
```

6.22.3 Member Function Documentation

6.22.3.1 loop()

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

wander scenario loop function

Note

opengl callback forces that function to be static

Definition at line 15 of file wander.cpp.

```
{
    for(auto it = agents.begin(); it < agents.end(); it++){
        (*it).force = behavior.wander(*it);
    }

    refresh();
}
```

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

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

6.23 windy Class Reference

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

Public Member Functions

- windy ()

    default constructor.

Static Public Member Functions

- static void loop ()

    windy scenario loop function

Static Public Attributes

- static flowField flow

    flow field used

Additional Inherited Members

6.23.1 Detailed Description

Definition at line 15 of file windy.h.

6.23.2 Constructor & Destructor Documentation

6.23.2.1 windy()

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

default constructor.

Definition at line 29 of file windy.cpp.

```
{
    int agentCount = 30;
    float maxForce = 0.3;
    float maxSpeed = 0.6;

    name = "flow field";
    createAgent(RANDOM, &agentCount, &maxForce, &maxSpeed);
    callback = reinterpret_cast <void(*)()> ( (void *)(&loop) );
}
```

6.23.3 Member Function Documentation

6.23.3.1 loop()

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

windy scenario loop function

Note

opengl callback forces that function to be static

Definition at line 17 of file windy.cpp.

```
{
    for(auto it = agents.begin(); it < agents.end(); it++){
        flow = flowField(pvector(GRAVITY));
        (*it).force  = behavior.inFlowField(*it, flow);

        flow = flowField(pvector(WIND_WEST));
        (*it).force += behavior.inFlowField(*it, flow);
    }
    refresh();
}
```

6.23.4 Member Data Documentation

6.23.4.1 flow

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

flow field used

Note

opengl callback forces that function to be static

Definition at line 32 of file windy.h.

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

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

Chapter 7

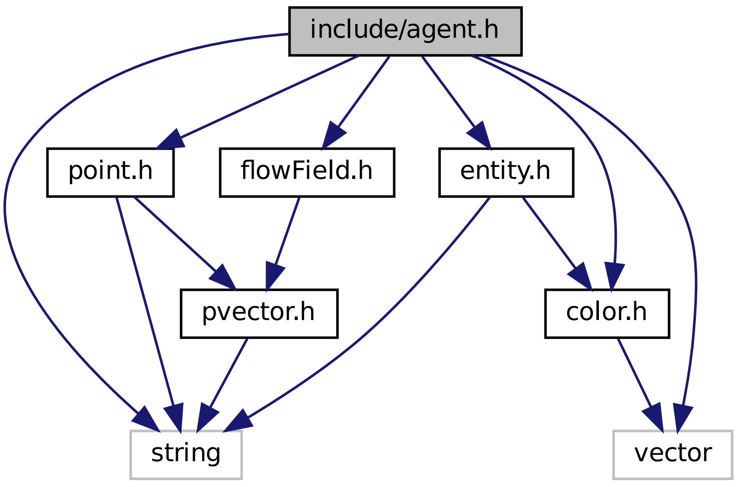
File Documentation

7.1 include/agent.h File Reference

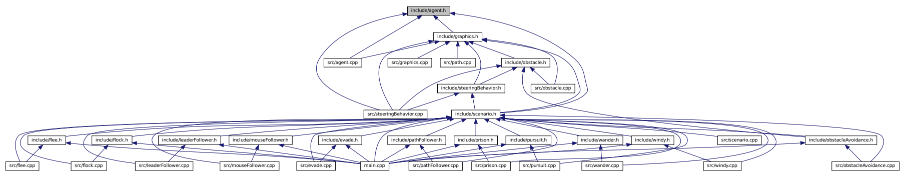
agent class defines all agent specifications

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

Include dependency graph for agent.h:



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



Classes

- class agent

7.1.1 Detailed Description

agent class defines all agent specifications

Author

Mehmet Rıza Öz - `mehmetrizaoz@gmail.com`

Date

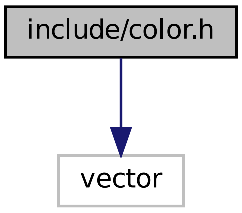
14.05.2021

7.2 include/color.h File Reference

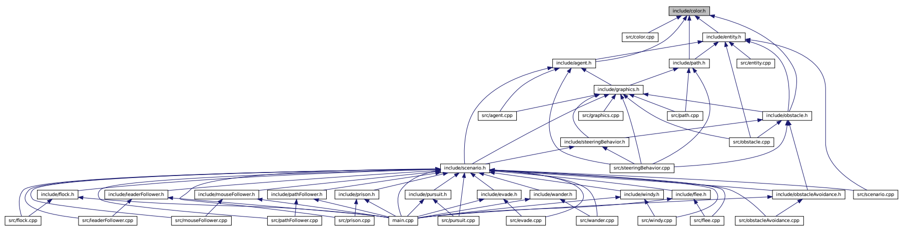
color class used for agent, path, wall etc. color

```
#include <vector>
```

Include dependency graph for color.h:



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



Classes

- class color

Macros

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

7.2.1 Detailed Description

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

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

13.05.2021

7.2.2 Macro Definition Documentation

7.2.2.1 BLACK

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

Definition at line 10 of file color.h.

7.2.2.2 BLUE

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

Definition at line 11 of file color.h.

7.2.2.3 CYAN

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

Definition at line 13 of file color.h.

7.2.2.4 GREEN

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

Definition at line 12 of file color.h.

7.2.2.5 MAGENDA

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

Definition at line 16 of file color.h.

7.2.2.6 RED

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

Definition at line 14 of file color.h.

7.2.2.7 WHITE

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

Definition at line 17 of file color.h.

7.2.2.8 YELLOW

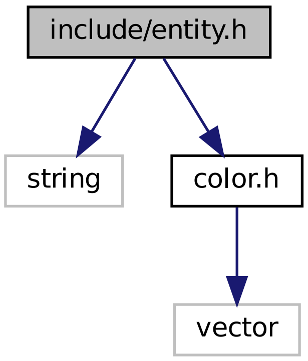
```
#define YELLOW color(1,1,0)
```

Definition at line 15 of file color.h.

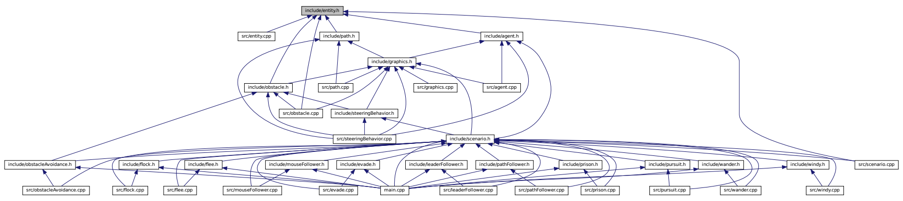
7.3 include/entity.h File Reference

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

Include dependency graph for entity.h:



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



Classes

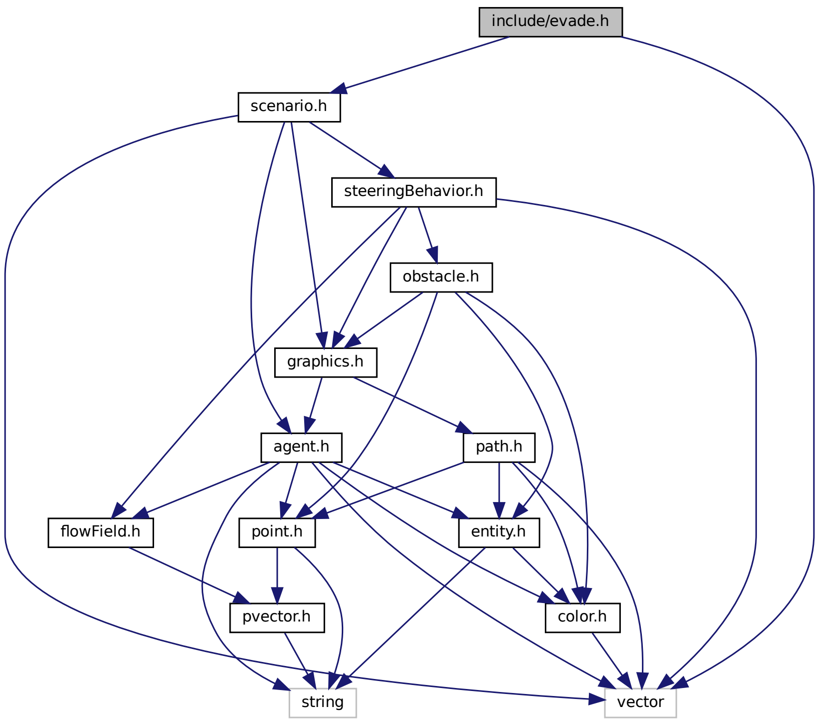
- class entity

7.4 include/evade.h File Reference

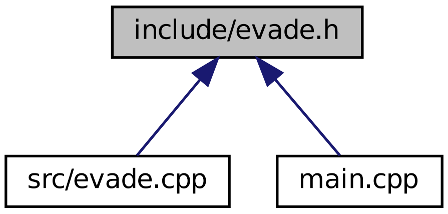
evade class inherited from scenario class

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

Include dependency graph for evade.h:



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



Classes

- class evade

7.4.1 Detailed Description

evade class inherited from scenario class

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

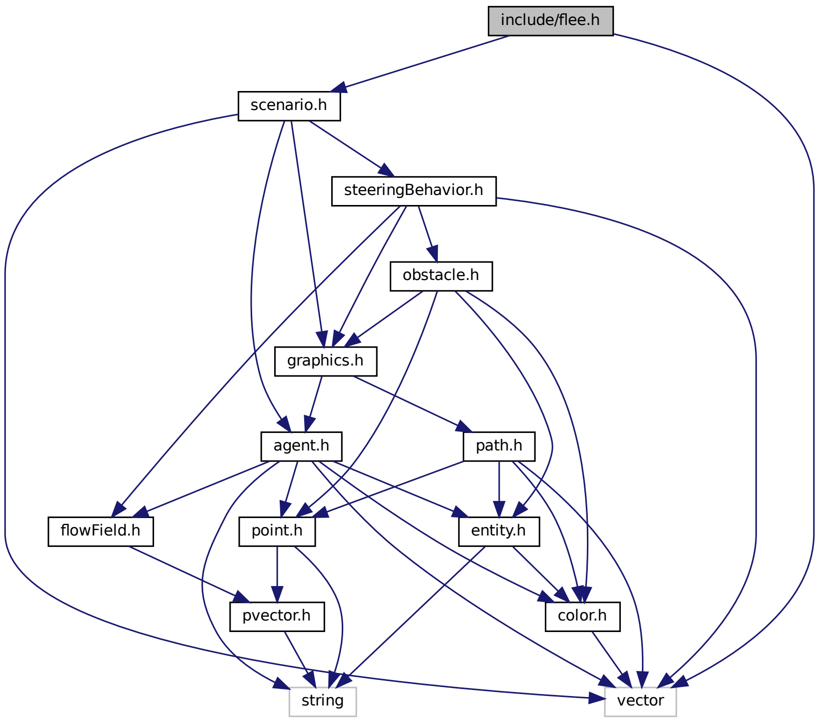
15.05.2021

7.5 include/flee.h File Reference

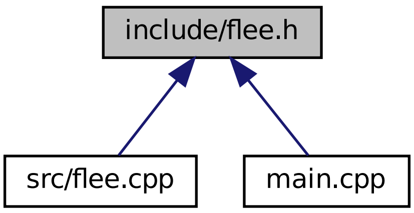
agents flee from mouse scenario

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

Include dependency graph for flee.h:



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



Classes

- class flee

7.5.1 Detailed Description

agents flee from mouse scenario

Author

Mehmet Rıza Öz - `mehmetrizaoz@gmail.com`

Date

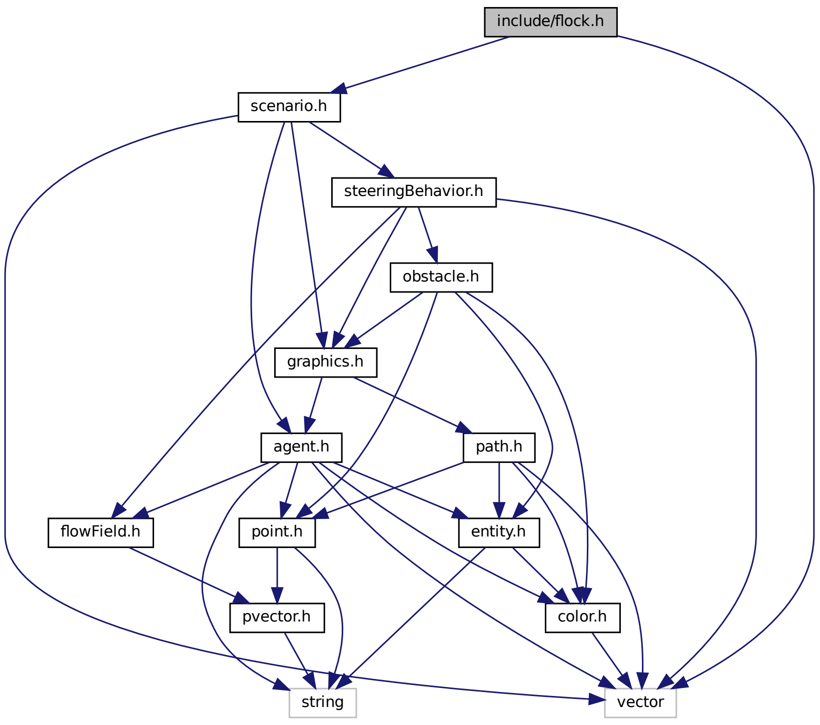
15.05.2021

7.6 include/flock.h File Reference

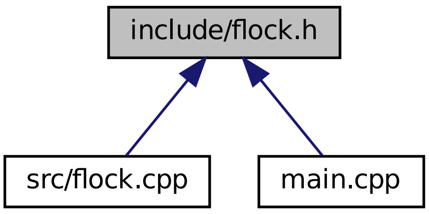
flocking agents scenario

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

Include dependency graph for flock.h:



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



Classes

- class flock

7.6.1 Detailed Description

flocking agents scenario

Author

Mehmet Rıza Öz - `mehmetrizaoz@gmail.com`

Date

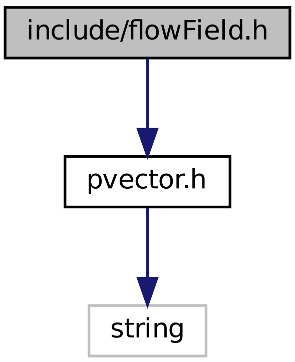
15.05.2021

7.7 include/flowField.h File Reference

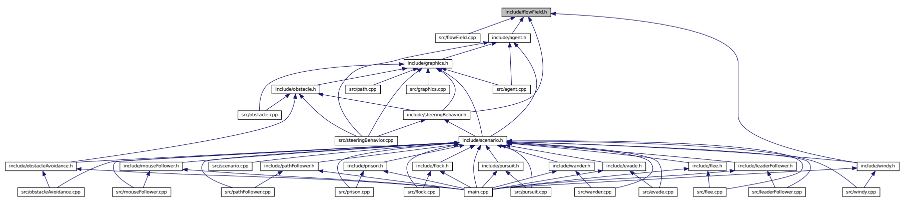
flowField class, screen can be filled with a force for each pixel

`#include "pvector.h"`

Include dependency graph for flowField.h:



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



Classes

- class flowField

Macros

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

7.7.1 Detailed Description

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

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

13.05.2021

7.7.2 Macro Definition Documentation

7.7.2.1 FIELD_HEIGHT

```
#define FIELD_HEIGHT 34
```

Definition at line 13 of file flowField.h.

7.7.2.2 FIELD_WIDTH

```
#define FIELD_WIDTH 34
```

Definition at line 12 of file flowField.h.

7.7.2.3 GRAVITY

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

Definition at line 16 of file flowField.h.

7.7.2.4 WIND_WEST

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

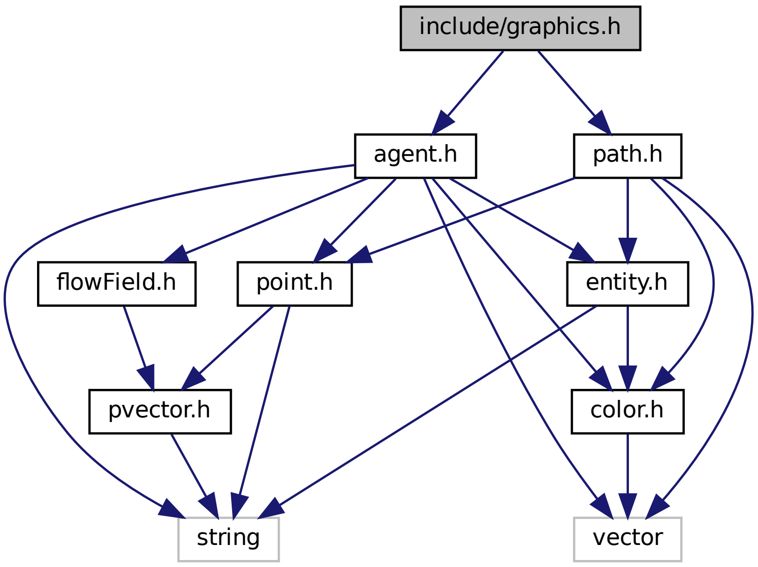
Definition at line 15 of file flowField.h.

7.8 include/graphics.h File Reference

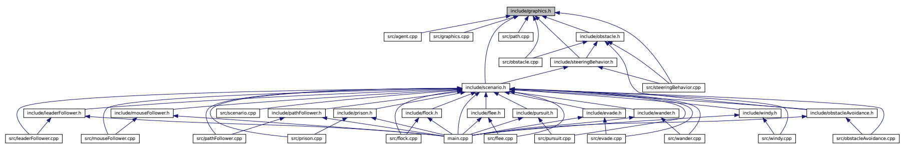
graphics class, drives openGL

```
#include "agent.h"
#include "path.h"
```

Include dependency graph for graphics.h:



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



Classes

- class graphics

Macros

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

7.8.1 Detailed Description

graphics class, drives openGL

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

7.8.2 Macro Definition Documentation

7.8.2.1 ESC

```
#define ESC 27
```

Definition at line 16 of file graphics.h.

7.8.2.2 HEIGHT

```
#define HEIGHT 34
```

Definition at line 14 of file graphics.h.

7.8.2.3 PI

```
#define PI 3.14159265
```

Definition at line 17 of file graphics.h.

7.8.2.4 WIDTH

```
#define WIDTH 34
```

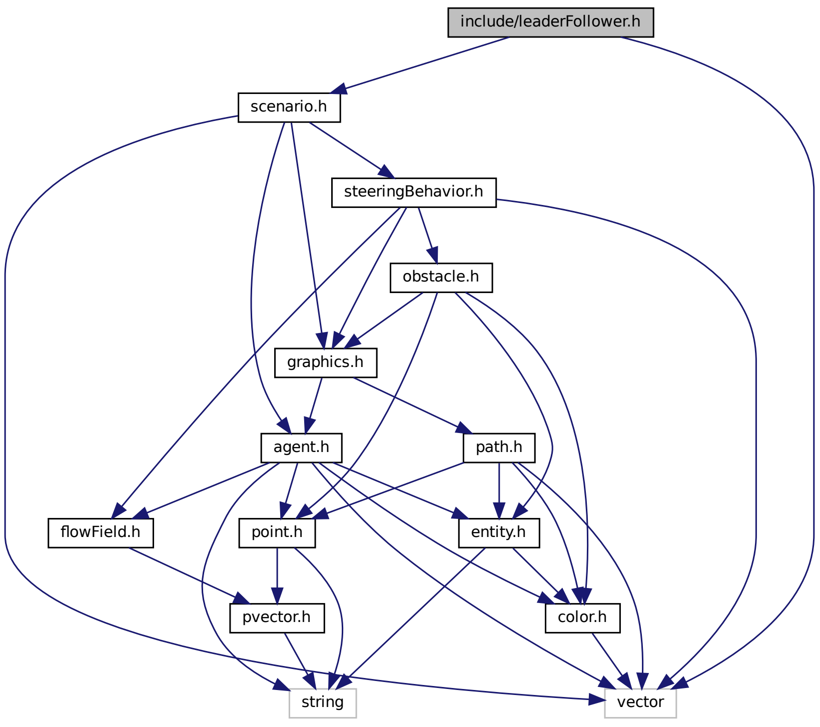
Definition at line 13 of file graphics.h.

7.9 include/leaderFollower.h File Reference

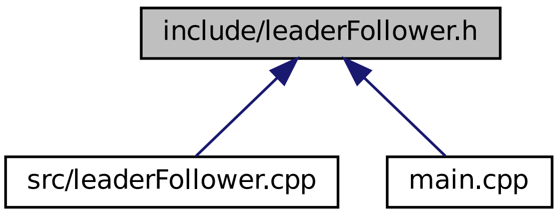
agents follow leader scenario

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

Include dependency graph for leaderFollower.h:



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



Classes

- class leaderFollower

7.9.1 Detailed Description

agents follow leader scenario

Author

Mehmet Rıza Öz - `mehmetrizaoz@gmail.com`

Date

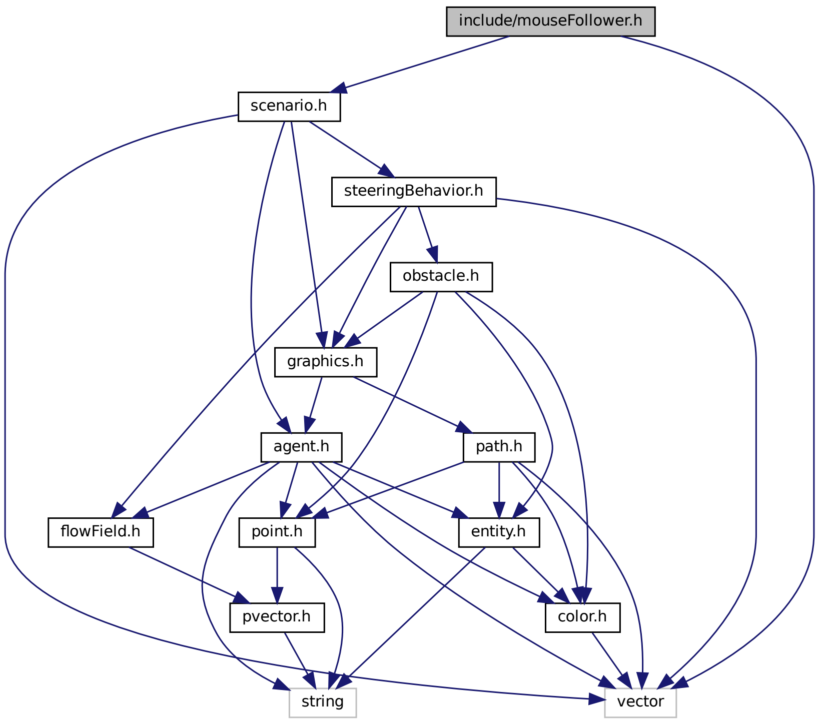
19.05.2021

7.10 include/mouseFollower.h File Reference

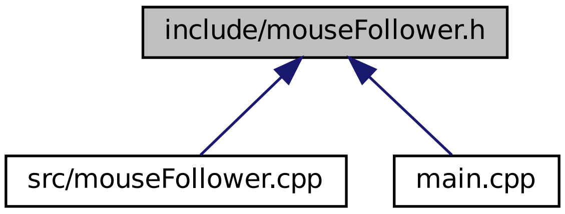
agents follow mouse scenario

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

Include dependency graph for mouseFollower.h:



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



Classes

- class mouseFollower

7.10.1 Detailed Description

agents follow mouse scenario

Author

Mehmet Rıza Öz - `mehmetrizaoz@gmail.com`

Date

15.05.2021

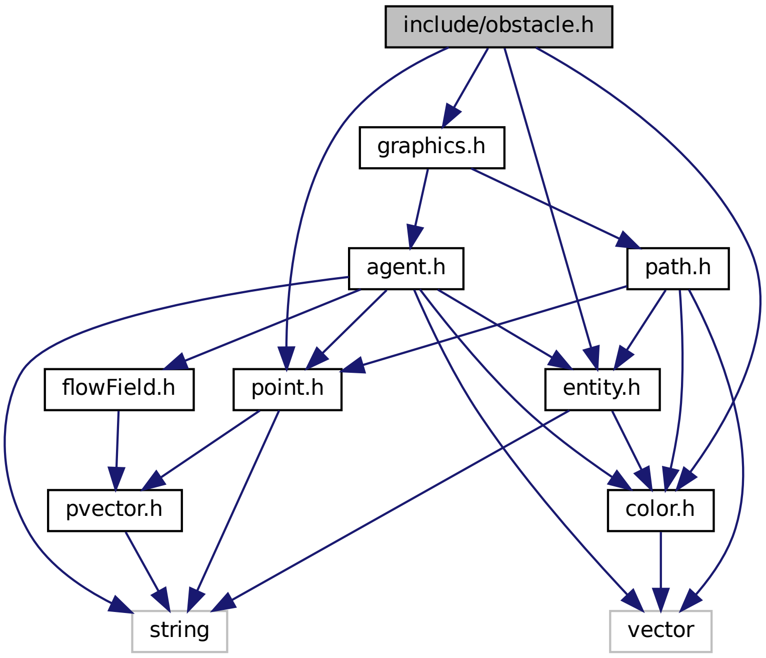
7.11 include/obstacle.h File Reference

circular obstacles for agent avoidance behaviors

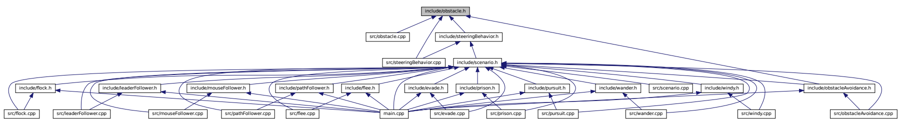
```
#include "point.h"
#include "graphics.h"
#include "color.h"
```

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

Include dependency graph for obstacle.h:



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



Classes

- class obstacle

7.11.1 Detailed Description

circular obstacles for agent avoidance behaviors

Author

Mehmet Rıza Öz - `mehmetrizaoz@gmail.com`

Date

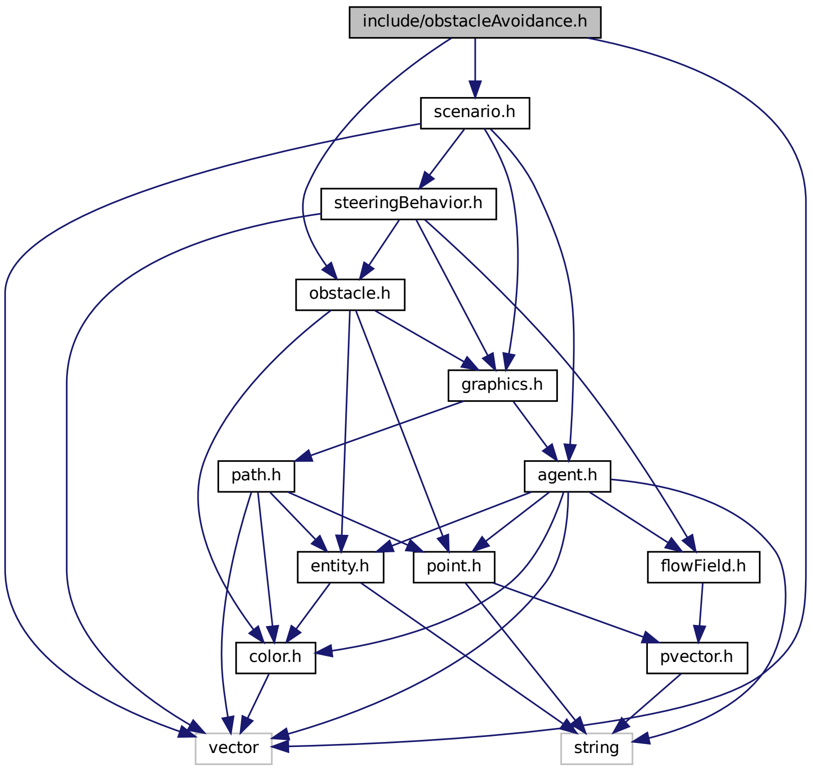
12.05.2021

7.12 include/obstacleAvoidance.h File Reference

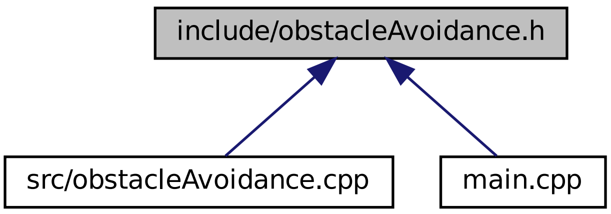
agents avoid from obstacles scenario

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

Include dependency graph for obstacleAvoidance.h:



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



Classes

- class obstacleAvoidance

7.12.1 Detailed Description

agents avoid from obstacles scenario

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

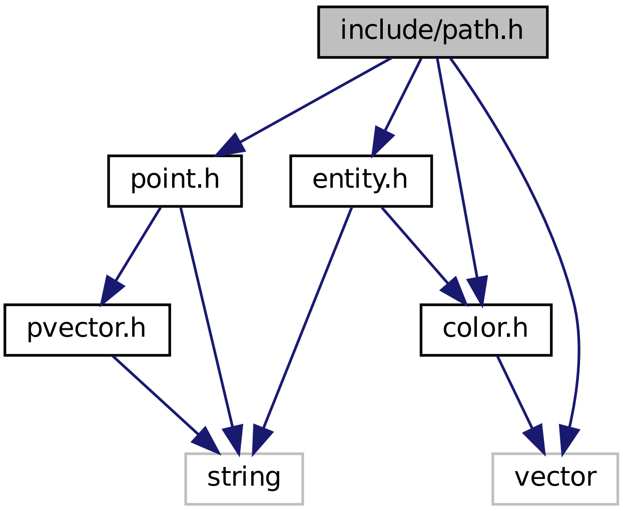
15.05.2021

7.13 include/path.h File Reference

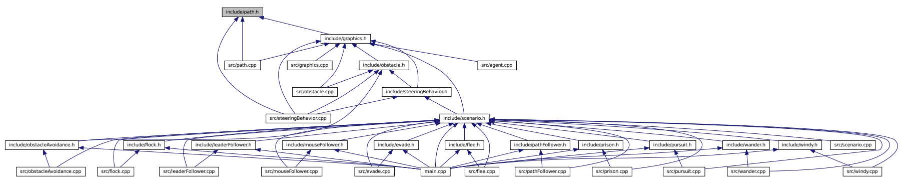
path class used for path following steering behaviors.

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

Include dependency graph for path.h:



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



Classes

- class path

7.13.1 Detailed Description

path class used for path following steering behaviors.

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

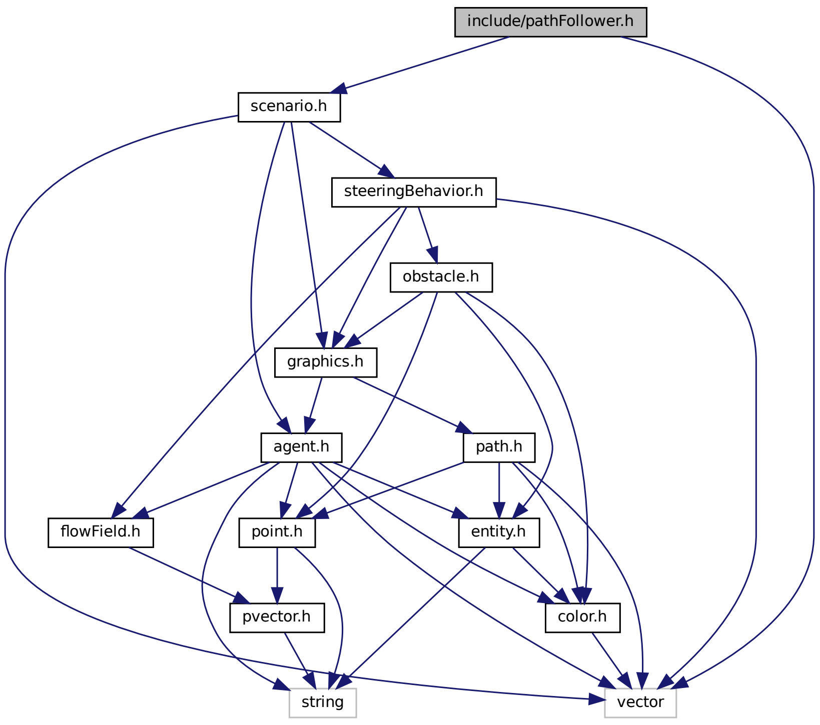
12.05.2021

7.14 include/pathFollower.h File Reference

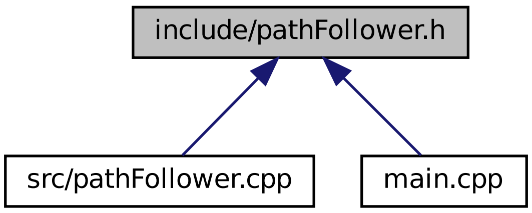
path following scenario

```
#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

7.14.1 Detailed Description

path following scenario

Author

Mehmet Rıza Öz - `mehmetrizaoz@gmail.com`

Date

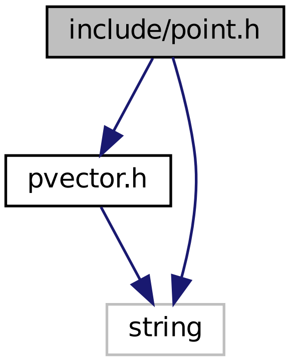
15.05.2021

7.15 include/point.h File Reference

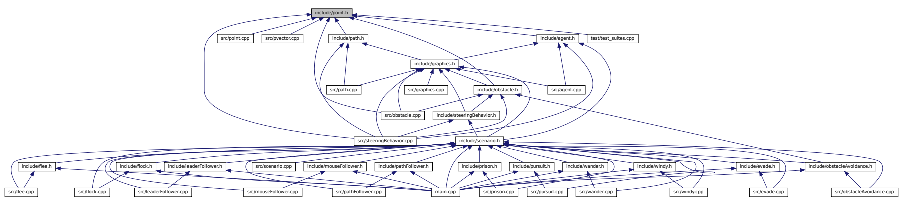
point class used for point operations

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

Include dependency graph for point.h:



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



Classes

- class point

7.15.1 Detailed Description

point class used for point operations

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

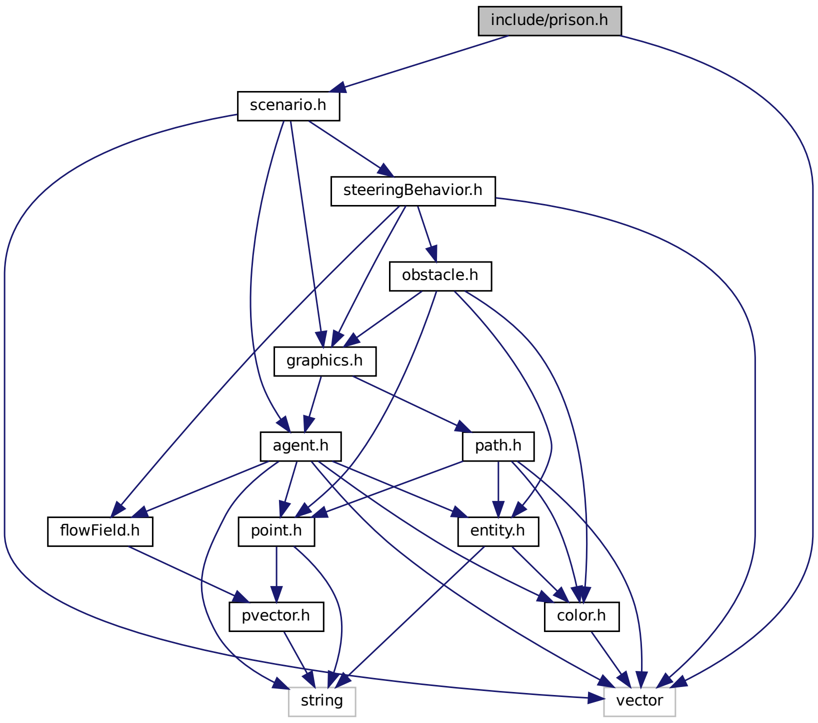
15.05.2021

7.16 include/prison.h File Reference

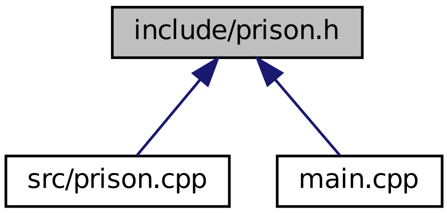
agents cant escape from field scenario

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

Include dependency graph for prison.h:



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



Classes

- class prison

7.16.1 Detailed Description

agents cant escape from field scenario

Author

Mehmet Rıza Öz - `mehmetrizaoz@gmail.com`

Date

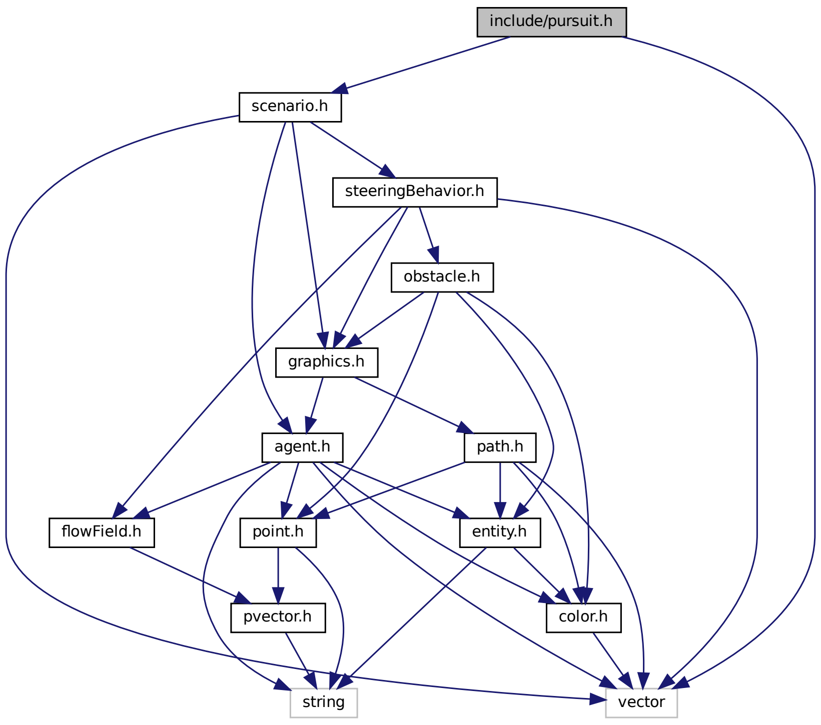
15.05.2021

7.17 include/pursuit.h File Reference

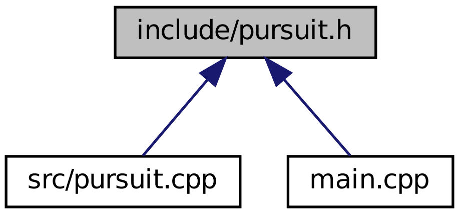
one agent pursue other one scenario

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

Include dependency graph for pursuit.h:



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



Classes

- class pursuit

7.17.1 Detailed Description

one agent pursue other one scenario

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

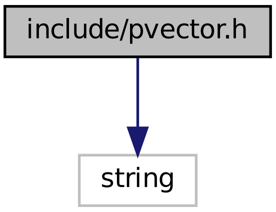
15.05.2021

7.18 include/pvector.h File Reference

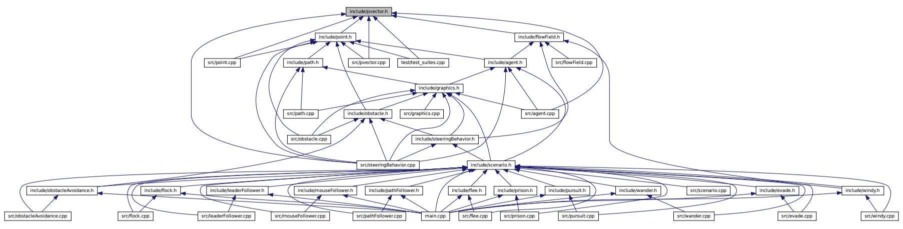
pvector class used for 2D vector operations

`#include <string>`

Include dependency graph for pvector.h:



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



Classes

- class pvector

Macros

- #define PI 3.14159265

7.18.1 Detailed Description

pvector class used for 2D vector operations

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

7.18.2 Macro Definition Documentation

7.18.2.1 PI

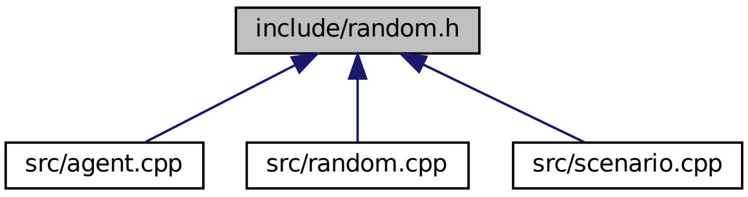
```
#define PI 3.14159265
```

Definition at line 11 of file pvector.h.

7.19 include/random.h File Reference

utility class for random operations

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



Classes

- class random

7.19.1 Detailed Description

utility class for random operations

Author

Mehmet Rıza Öz - `mehmetrizaoz@gmail.com`

Date

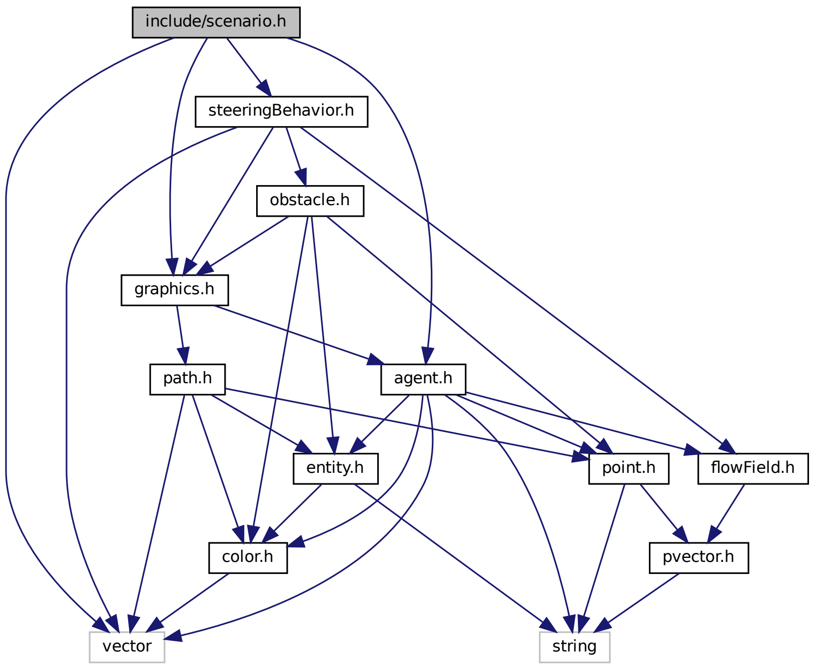
15.05.2021

7.20 include/scenario.h File Reference

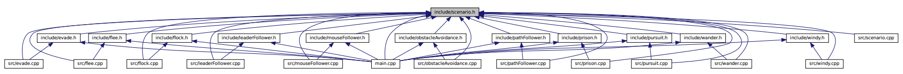
base class for all scenarios

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

Include dependency graph for scenario.h:



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



Classes

- class scenario

Enumerations

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

7.20.1 Detailed Description

base class for all scenarios

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

7.20.2 Enumeration Type Documentation

7.20.2.1 types

```
enum types
```

Enumerator

RANDOM	
STATIC	
TROOP	

Definition at line 17 of file scenario.h.

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

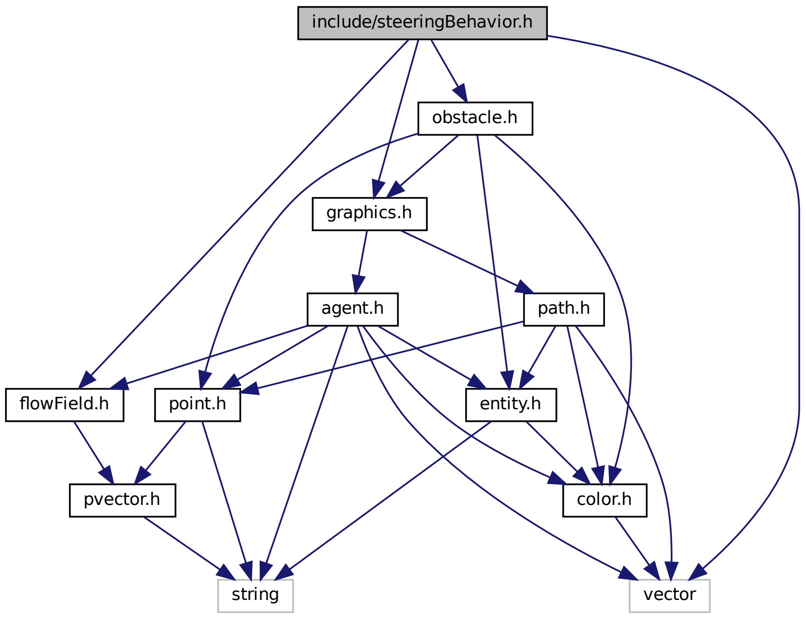
7.21 include/steeringBehavior.h File Reference

functions for autonomous steering behaviors

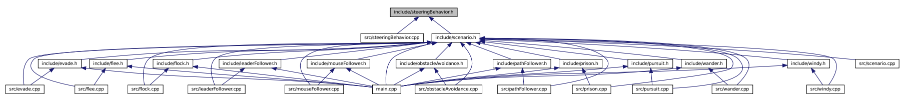
```
#include "flowField.h"
#include <vector>
#include "graphics.h"
```

`#include "obstacle.h"`

Include dependency graph for steeringBehavior.h:



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



Classes

- class steeringBehavior

Macros

- #define CIRCLE_DISTANCE 0.1
- #define CIRCLE_RADIUS 0.4
- #define FOLLOW_MOUSE 1
- #define STAY_IN_FIELD 2
- #define IN_FLOW_FIELD 3
- #define AVOID_OBSTACLE 4
- #define STAY_IN_PATH 5
- #define FLOCK 6
- #define WANDER 7
- #define FLEE 8
- #define PURSUIT 9
- #define EVADE 10
- #define LEADER_FOLLOWER 11

7.21.1 Detailed Description

functions for autonomous steering behaviors

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

7.21.2 Macro Definition Documentation

7.21.2.1 AVOID_OBSTACLE

```
#define AVOID_OBSTACLE 4
```

Definition at line 21 of file steeringBehavior.h.

7.21.2.2 CIRCLE_DISTANCE

```
#define CIRCLE_DISTANCE 0.1
```

Definition at line 15 of file steeringBehavior.h.

7.21.2.3 CIRCLE_RADIUS

```
#define CIRCLE_RADIUS 0.4
```

Definition at line 16 of file steeringBehavior.h.

7.21.2.4 EVADE

```
#define EVADE 10
```

Definition at line 27 of file steeringBehavior.h.

7.21.2.5 FLEE

```
#define FLEE 8
```

Definition at line 25 of file steeringBehavior.h.

7.21.2.6 FLOCK

```
#define FLOCK 6
```

Definition at line 23 of file steeringBehavior.h.

7.21.2.7 FOLLOW_MOUSE

```
#define FOLLOW_MOUSE 1
```

Definition at line 18 of file steeringBehavior.h.

7.21.2.8 IN_FLOW_FIELD

```
#define IN_FLOW_FIELD 3
```

Definition at line 20 of file steeringBehavior.h.

7.21.2.9 LEADER_FOLLOWER

```
#define LEADER_FOLLOWER 11
```

Definition at line 28 of file steeringBehavior.h.

7.21.2.10 PURSUIT

```
#define PURSUIT 9
```

Definition at line 26 of file steeringBehavior.h.

7.21.2.11 STAY_IN_FIELD

```
#define STAY_IN_FIELD 2
```

Definition at line 19 of file steeringBehavior.h.

7.21.2.12 STAY_IN_PATH

```
#define STAY_IN_PATH 5
```

Definition at line 22 of file steeringBehavior.h.

7.21.2.13 WANDER

```
#define WANDER 7
```

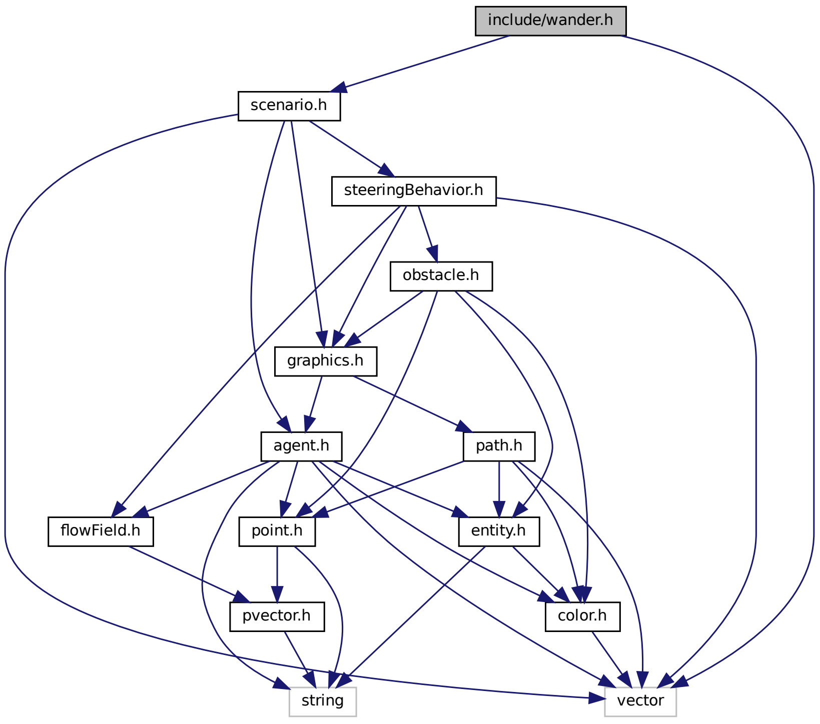
Definition at line 24 of file steeringBehavior.h.

7.22 include/wander.h File Reference

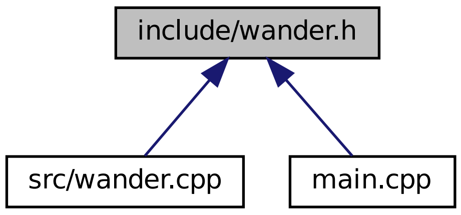
random wandering agents scenario

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

Include dependency graph for wander.h:



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



Classes

- class wander

7.22.1 Detailed Description

random wandering agents scenario

Author

Mehmet Rıza Öz - `mehmetrizaoz@gmail.com`

Date

15.05.2021

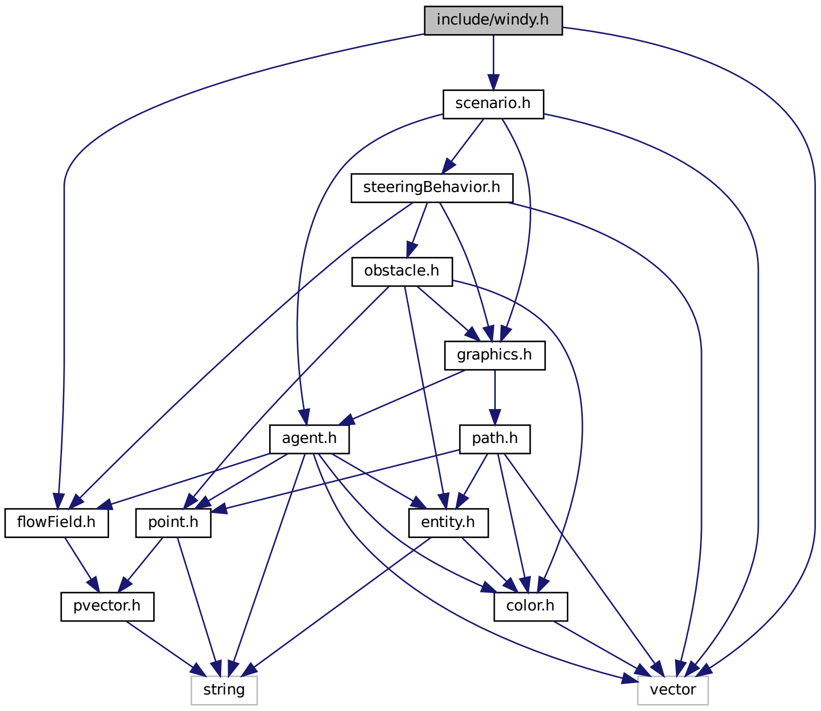
7.23 include/windy.h File Reference

windy air scenario

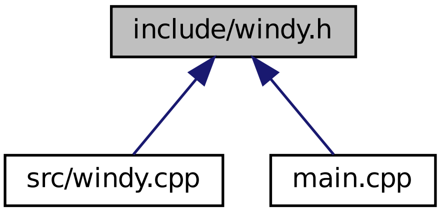
```
#include "scenario.h"
#include "flowField.h"
```

```
#include <vector>
```

Include dependency graph for windy.h:



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



Classes

- class windy

7.23.1 Detailed Description

windy air scenario

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

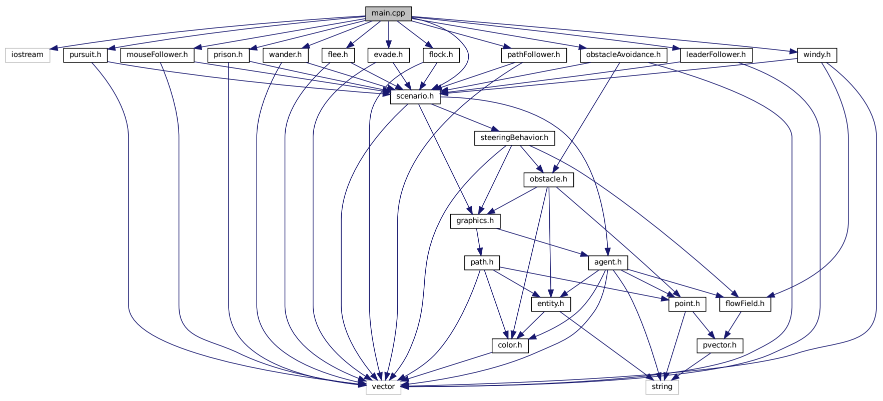
15.05.2021

7.24 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 "leaderFollower.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

7.24.1 Detailed Description

client code

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

7.24.2 Function Documentation

7.24.2.1 main()

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

main routine

Definition at line 50 of file main.cpp.

```
                                {
    menu();

    scenario* sc;

    if(mode == FOLLOW_MOUSE){
       *sc = mouseFollower();
    }
    else if(mode == STAY_IN_FIELD){
       *sc = prison();
    }
    else if(mode == IN_FLOW_FIELD){
       *sc = windy();
    }
    else if(mode == WANDER){
       *sc = wander();
    }
    else if(mode == PURSUIT){
       *sc = pursuit();
    }
    else if(mode == FLEE){
       *sc = flee();
    }
    else if(mode == EVADE){
       *sc = evade();
    }
    else if(mode == FLOCK){
       *sc = flock();
    }
    else if(mode == STAY_IN_PATH){
       *sc = pathFollower();
    }
    else if(mode == AVOID_OBSTACLE){
       *sc = obstacleAvoidance();
    }
    else if(mode == LEADER_FOLLOWER){
       *sc = leaderFollower();
    }

    sc->initGL(&argc, argv);

    return 0;
}
```

7.24.2.2 menu()

```
void menu ( )
```

displays menu

Definition at line 32 of file main.cpp.

```
          {
    cout « "Follow Mouse       : 1" « endl;
    cout « "Stay in Field      : 2" « endl;
    cout « "In Flow Field      : 3" « endl;
    cout « "Avoid Obstacles    : 4" « endl;
    cout « "Stay in Path       : 5" « endl;
    cout « "FLOCK              : 6" « endl;
    cout « "WANDER             : 7" « endl;
    cout « "FLEE               : 8" « endl;
    cout « "PURSUIT            : 9" « endl;
    cout « "EVADE              : 10" « endl;
    cout « "Follow Leader      : 11" « endl;
    cin » mode;
}
```

7.24.3 Variable Documentation

7.24.3.1 mode

```
int mode
```

specifies user selected scenario

Definition at line 27 of file main.cpp.

7.25 README.md File Reference

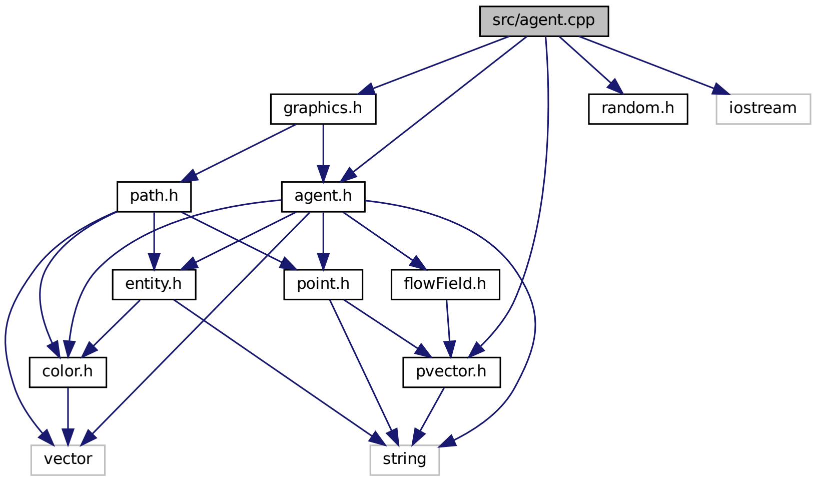
7.26 src/agent.cpp File Reference

implementation of the agent class

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

```
#include <iostream>
```

Include dependency graph for agent.cpp:



7.26.1 Detailed Description

implementation of the agent class

Author

Mehmet Rıza Öz - `mehmetrizaoz@gmail.com`

Date

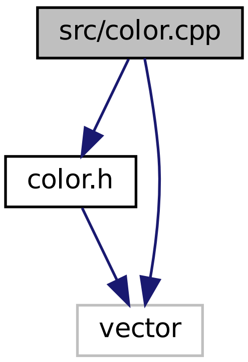
14.05.2021

7.27 src/color.cpp File Reference

color class implementation

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

Include dependency graph for color.cpp:



7.27.1 Detailed Description

color class implementation

Author

Mehmet Rıza Öz - `mehmetrizaoz@gmail.com`

Date

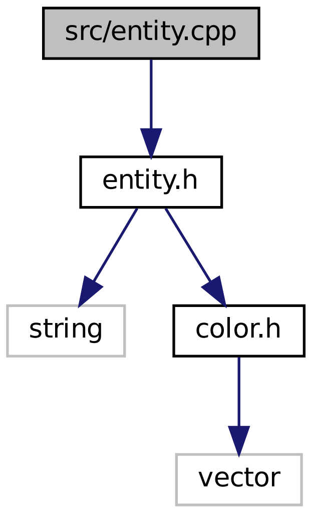
13.05.2021

7.28 src/entity.cpp File Reference

entity class implementation

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

Include dependency graph for entity.cpp:



7.28.1 Detailed Description

entity class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

18.05.2021

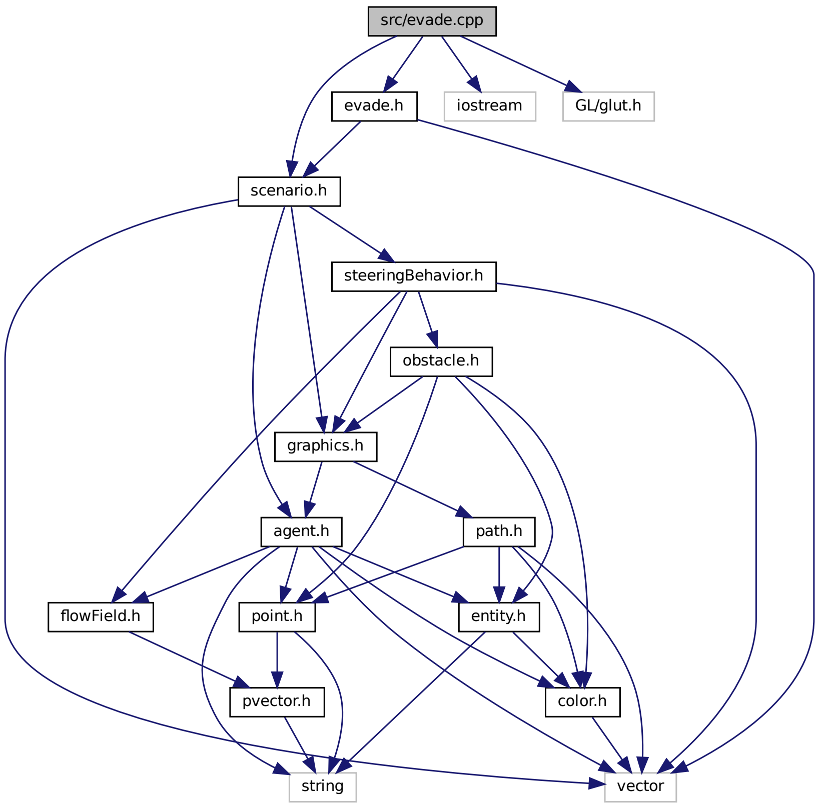
7.29 src/evade.cpp File Reference

evade class implementation

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

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

Include dependency graph for evade.cpp:



7.29.1 Detailed Description

evade class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

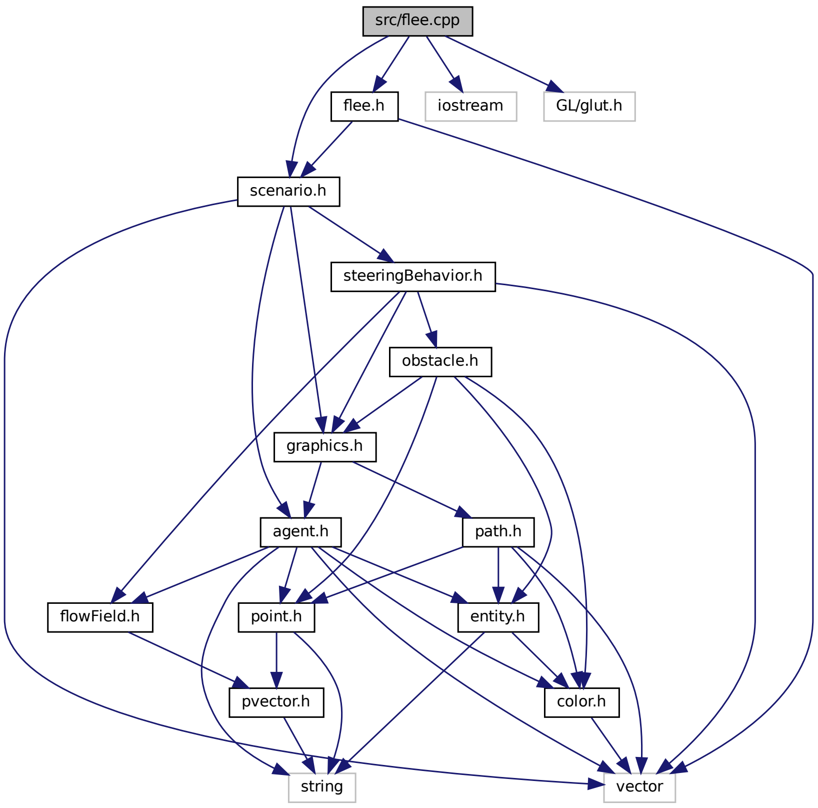
7.30 src/flee.cpp File Reference

flee class implementation

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

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

Include dependency graph for flee.cpp:



7.30.1 Detailed Description

flee class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

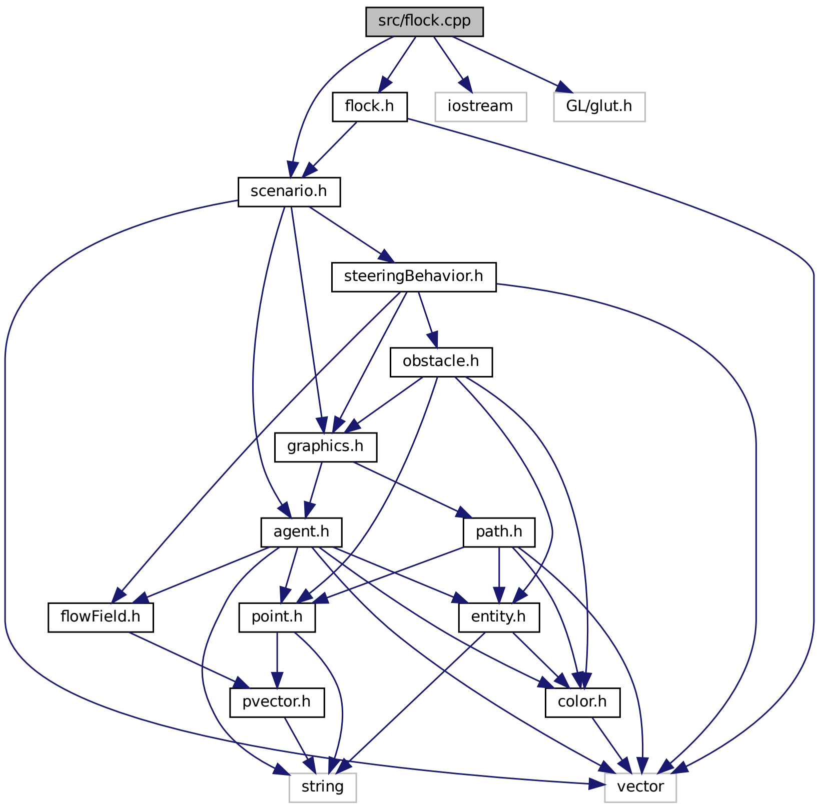
15.05.2021

7.31 src/flock.cpp File Reference

flock class implementation

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

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



7.31.1 Detailed Description

flock class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

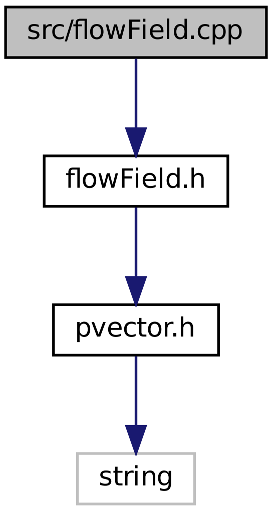
15.05.2021

7.32 src/flowField.cpp File Reference

flowField class implementation

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

Include dependency graph for flowField.cpp:



7.32.1 Detailed Description

flowField class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

13.05.2021

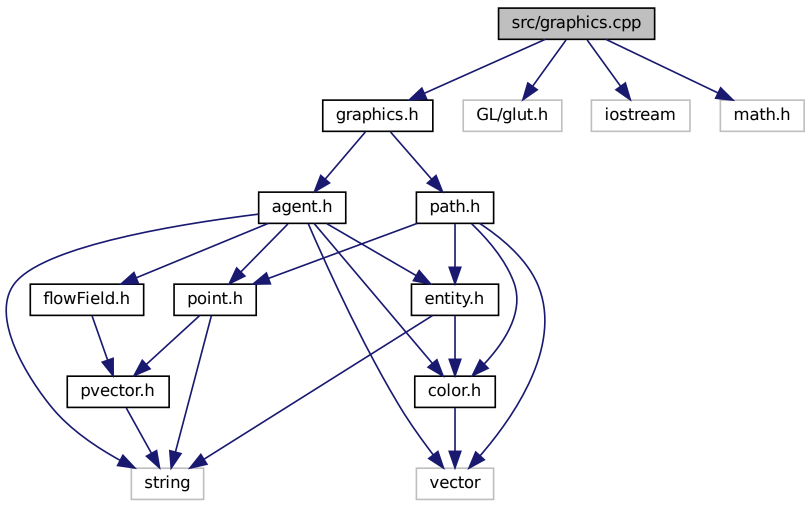
7.33 src/graphics.cpp File Reference

graphics class implementation

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

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

Include dependency graph for graphics.cpp:



7.33.1 Detailed Description

graphics class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

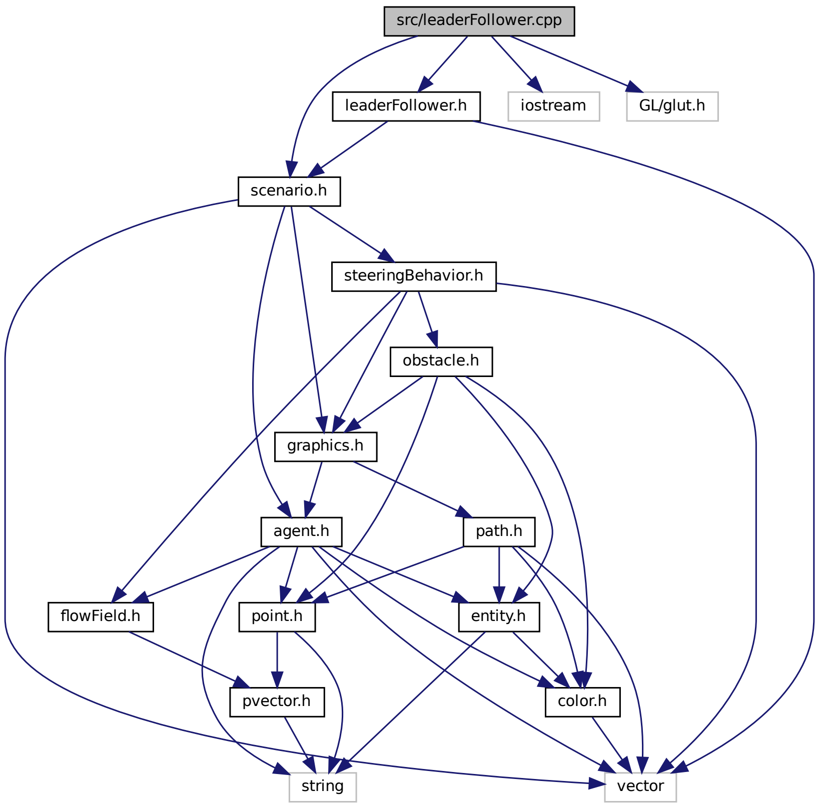
7.34 src/leaderFollower.cpp File Reference

leaderFollower class implementation

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

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

Include dependency graph for leaderFollower.cpp:



7.34.1 Detailed Description

leaderFollower class implementation

Author

Mehmet Rıza Öz - `mehmetrizaoz@gmail.com`

Date

19.05.2021

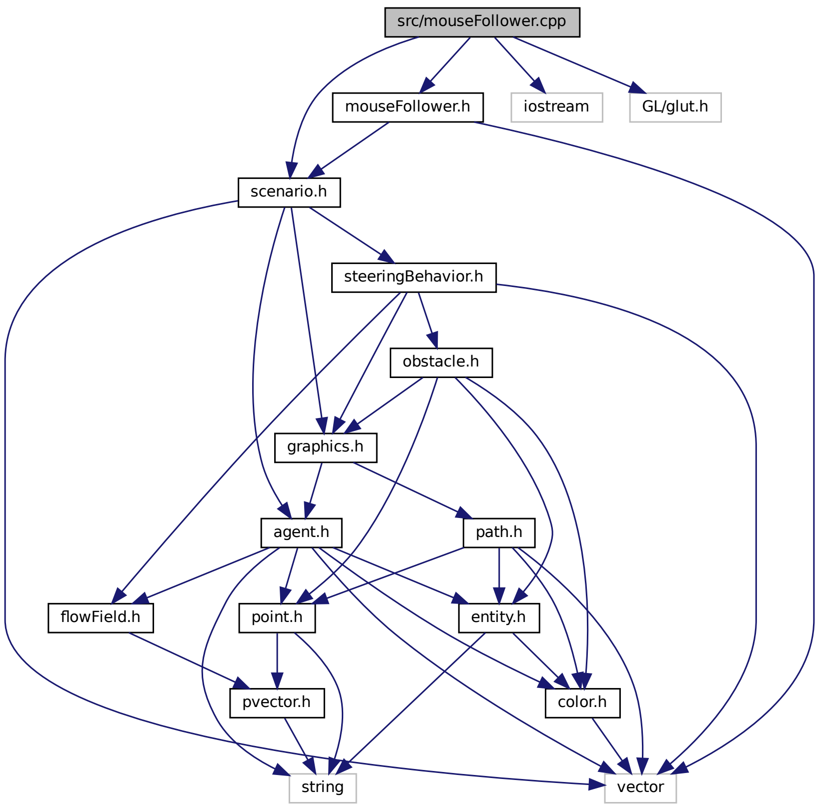
7.35 src/mouseFollower.cpp File Reference

mouseFollower class implementation

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

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

Include dependency graph for mouseFollower.cpp:



7.35.1 Detailed Description

mouseFollower class implementation

Author

Mehmet Rıza Öz - `mehmetrizaoz@gmail.com`

Date

15.05.2021

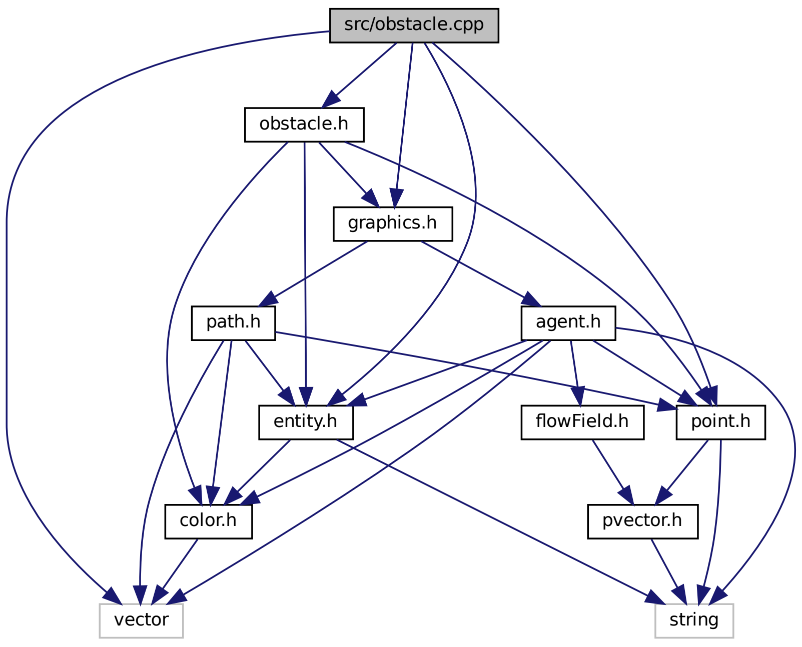
7.36 src/obstacle.cpp File Reference

obstacle class implementation

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

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

Include dependency graph for obstacle.cpp:



7.36.1 Detailed Description

obstacle class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

12.05.2021

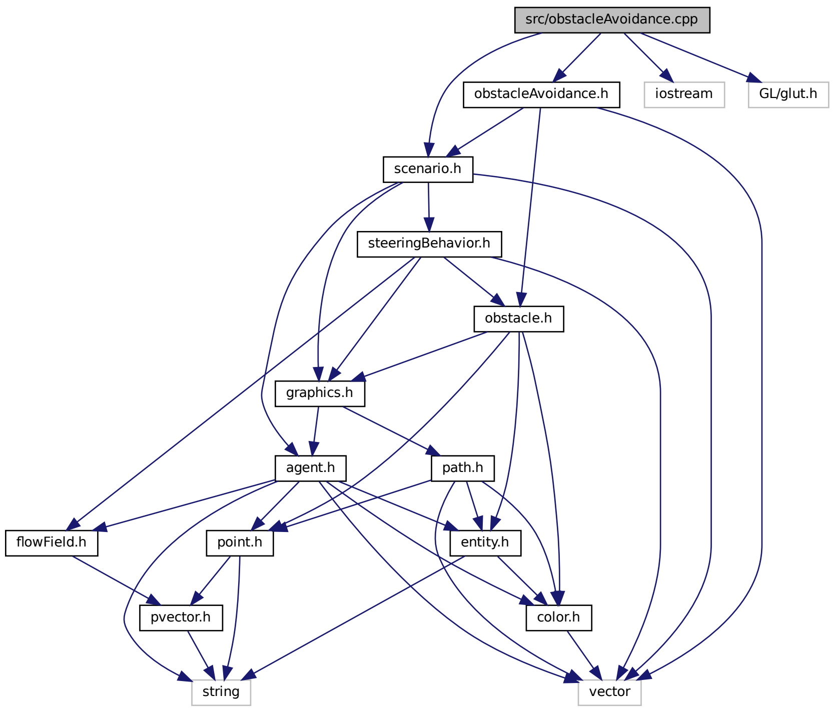
7.37 src/obstacleAvoidance.cpp File Reference

obstacleAvoidance class implementation

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

`#include <GL/glut.h>`

Include dependency graph for obstacleAvoidance.cpp:



7.37.1 Detailed Description

obstacleAvoidance class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

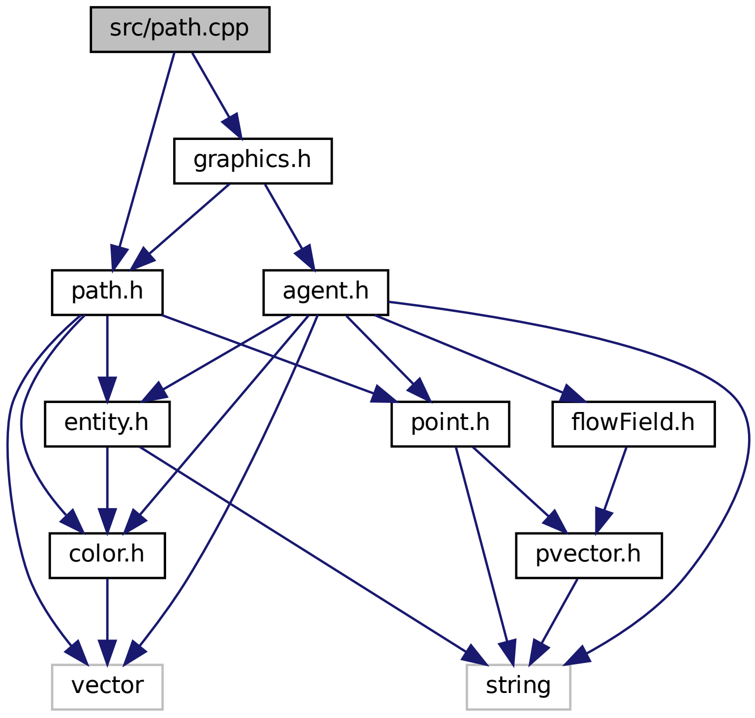
15.05.2021

7.38 src/path.cpp File Reference

path class implementation

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

Include dependency graph for path.cpp:



7.38.1 Detailed Description

path class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

12.05.2021

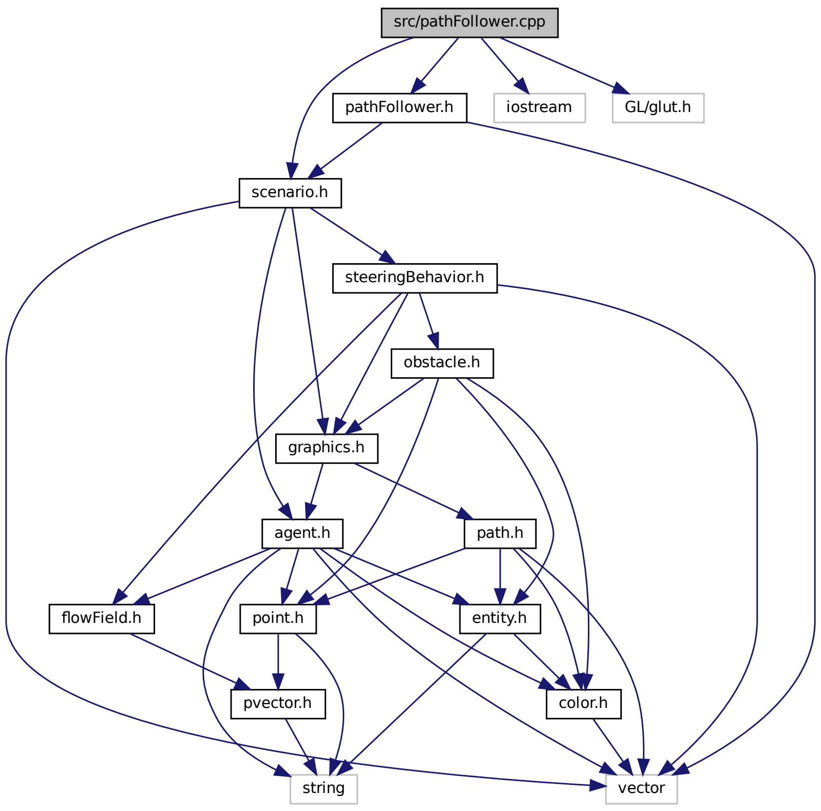
7.39 src/pathFollower.cpp File Reference

pathFollower class implementation

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

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

Include dependency graph for pathFollower.cpp:



7.39.1 Detailed Description

pathFollower class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

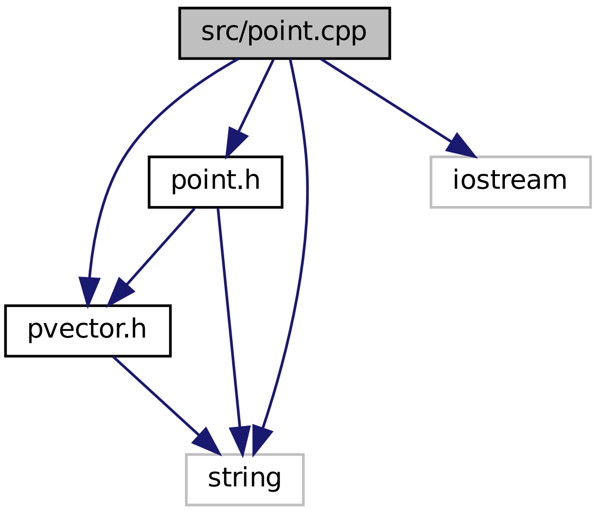
7.40 src/point.cpp File Reference

point class implementation file

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

```
#include <iostream>
```

Include dependency graph for point.cpp:



7.40.1 Detailed Description

point class implementation file

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

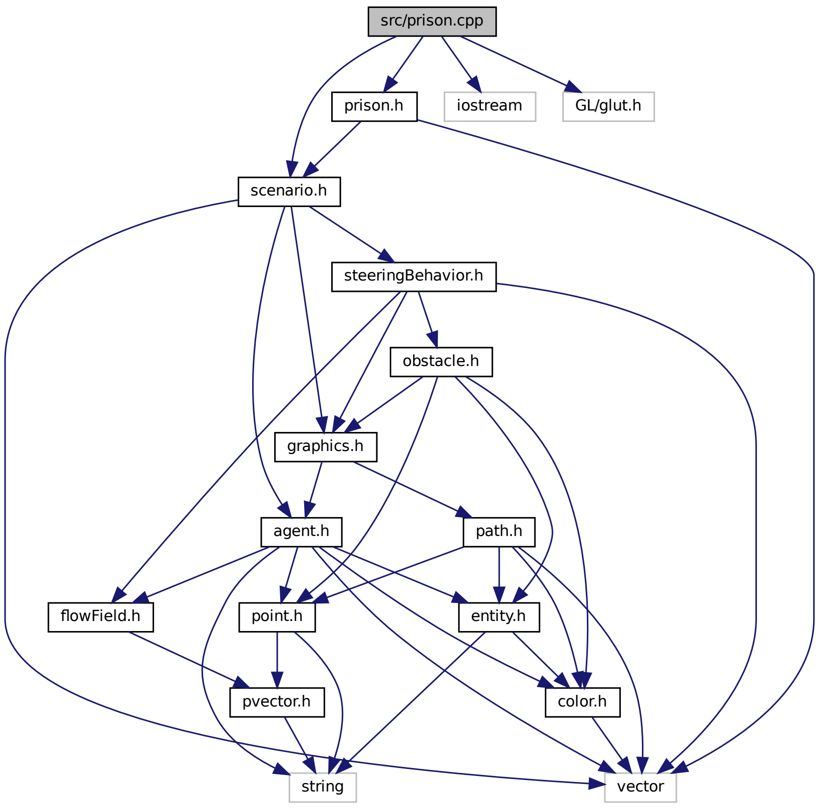
7.41 src/prison.cpp File Reference

prison class implementation

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

`#include <GL/glut.h>`

Include dependency graph for prison.cpp:



Macros

- #define WALL 30
- #define DISTANCE 2

7.41.1 Detailed Description

prison class implementation

Author

Mehmet Rıza Öz - `mehmetrizaoz@gmail.com`

Date

15.05.2021

7.41.2 Macro Definition Documentation

7.41.2.1 DISTANCE

```
#define DISTANCE 2
```

Definition at line 14 of file prison.cpp.

7.41.2.2 WALL

```
#define WALL 30
```

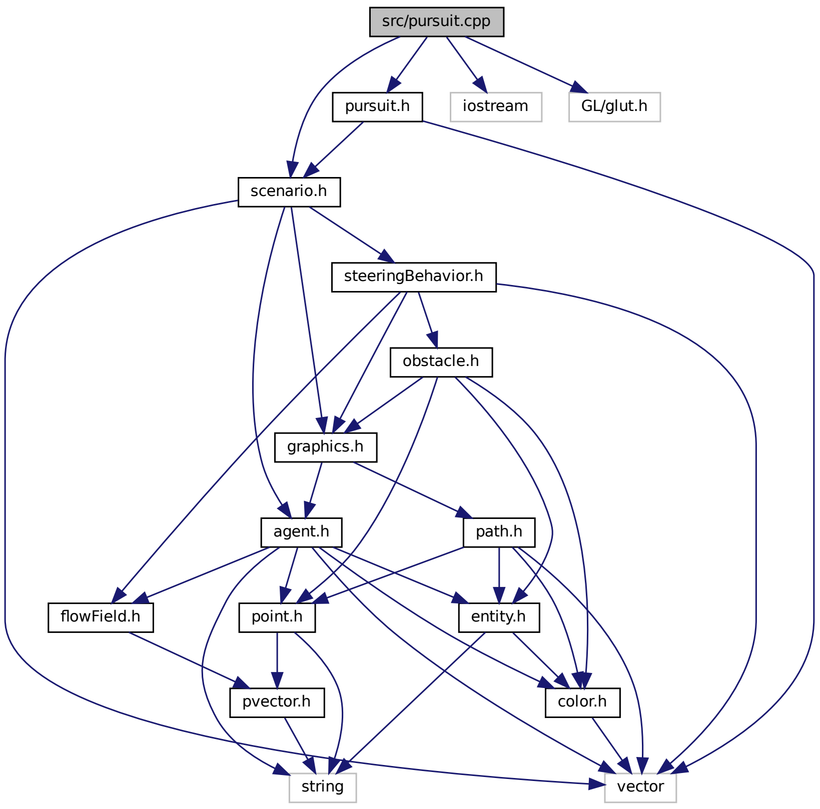
Definition at line 13 of file prison.cpp.

7.42 src/pursuit.cpp File Reference

prison class implementation

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

Include dependency graph for pursuit.cpp:



7.42.1 Detailed Description

prison class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

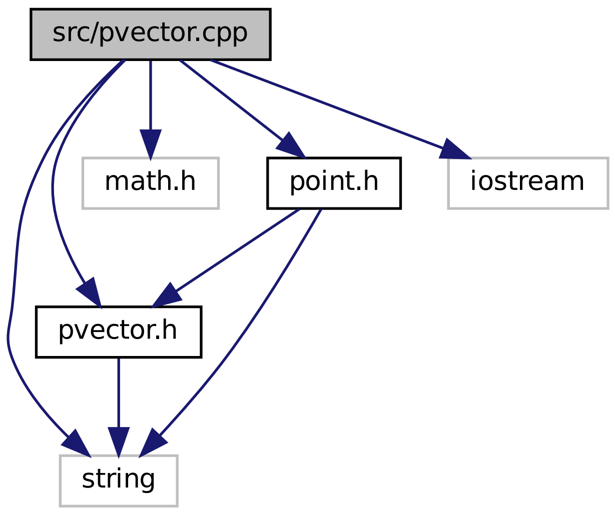
15.05.2021

7.43 src/pvector.cpp File Reference

pvector class implementation

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

Include dependency graph for pvector.cpp:



7.43.1 Detailed Description

pvector class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

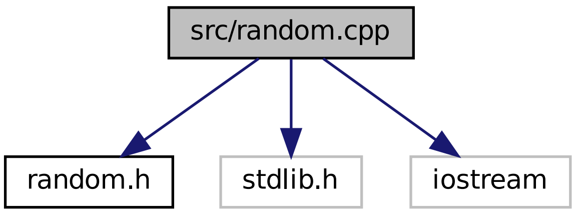
15.05.2021

7.44 src/random.cpp File Reference

utility class for random operations

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

Include dependency graph for random.cpp:



7.44.1 Detailed Description

utility class for random operations

Author

Mehmet Rıza Öz - `mehmetrizaoz@gmail.com`

Date

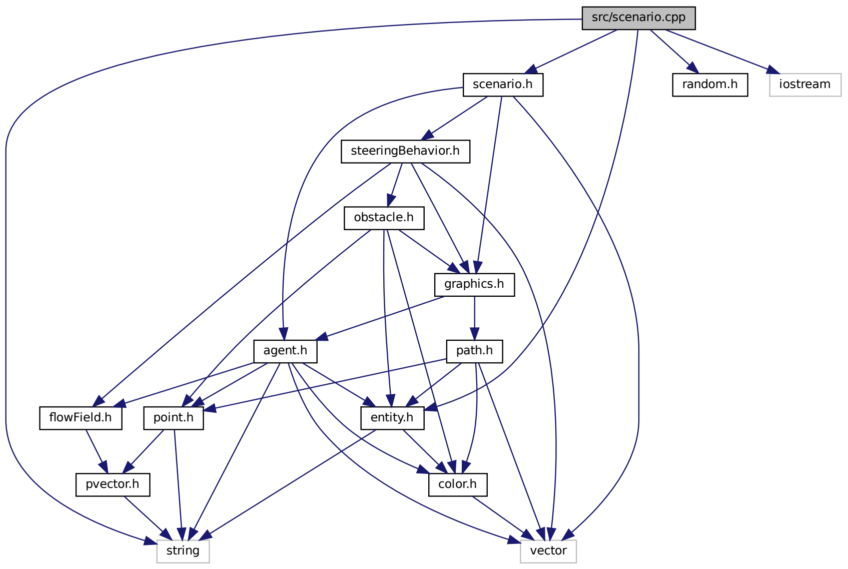
15.05.2021

7.45 src/scenario.cpp File Reference

scenario base class implementation

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

```
#include <string>
```
Include dependency graph for scenario.cpp:



Macros

- #define MAX_NUMBER_OF_AGENTS 50

7.45.1 Detailed Description

scenario base class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

7.45.2 Macro Definition Documentation

7.45.2.1 MAX_NUMBER_OF_AGENTS

```
#define MAX_NUMBER_OF_AGENTS 50
```

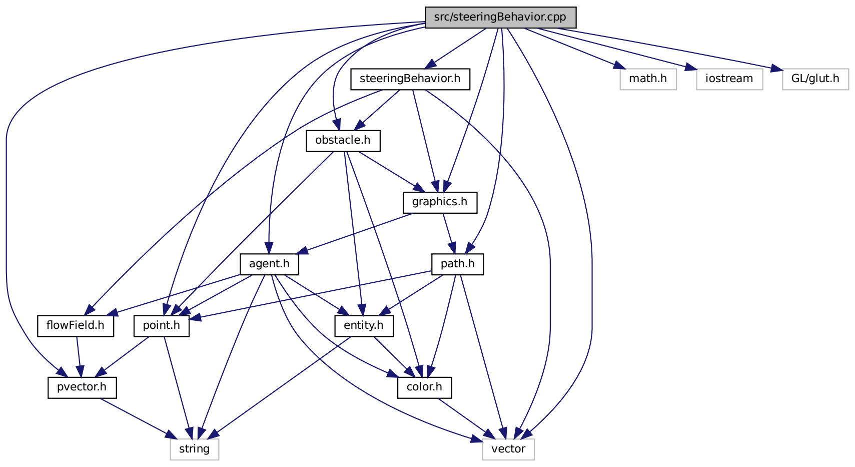
Definition at line 14 of file scenario.cpp.

7.46 src/steeringBehavior.cpp File Reference

implementation of autonomous steering behaviors

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

Include dependency graph for steeringBehavior.cpp:



7.46.1 Detailed Description

implementation of autonomous steering behaviors

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

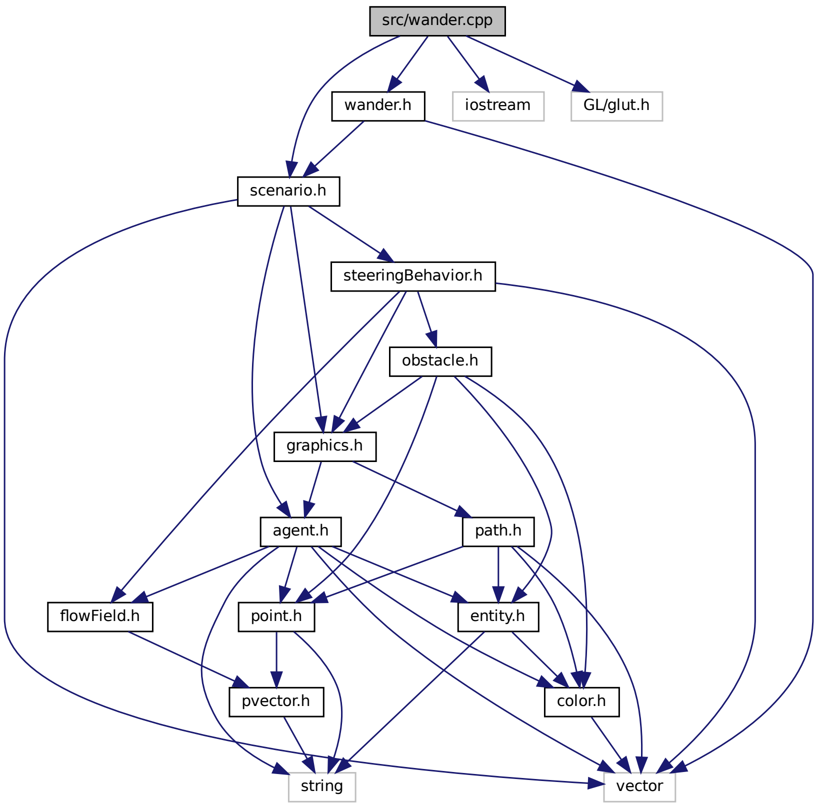
15.05.2021

7.47 src/wander.cpp File Reference

wander class implementation

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

Include dependency graph for wander.cpp:



7.47.1 Detailed Description

wander class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

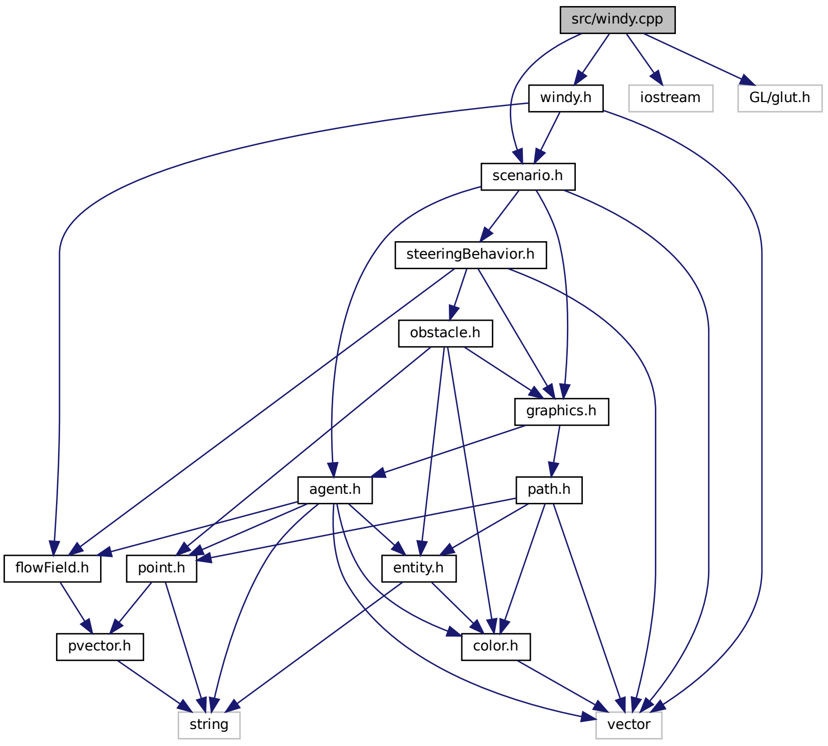
15.05.2021

7.48 src/windy.cpp File Reference

windy class implementation

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

Include dependency graph for windy.cpp:



7.48.1 Detailed Description

windy class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

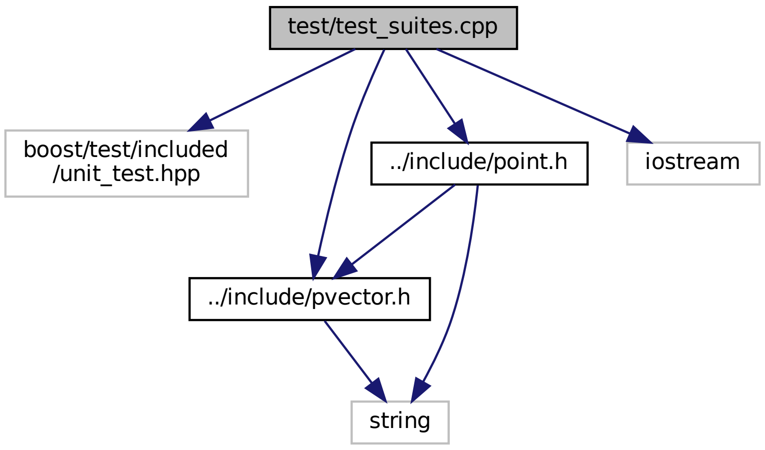
15.05.2021

7.49 test/test_suites.cpp File Reference

unit test suites

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

Include dependency graph for test_suites.cpp:



Macros

- #define BOOST_TEST_MODULE test_suites

Functions

- BOOST_AUTO_TEST_CASE (s1t1)

  pvector magnitude test case
- BOOST_AUTO_TEST_CASE (s1t2)

  pvector mul test case
- BOOST_AUTO_TEST_CASE (s1t3)

  pvector div test case
- BOOST_AUTO_TEST_CASE (s1t4)

  pvector dotproduct test case
- BOOST_AUTO_TEST_CASE (s1t5)

  pvector angle between vectors test case
- BOOST_AUTO_TEST_CASE (s1t6)

  pvector get vector angle test case
- BOOST_AUTO_TEST_CASE (s1t7)

  pvector normalize test case
- BOOST_AUTO_TEST_CASE (s1t8)

pvector limit test case
- BOOST_AUTO_TEST_CASE (s1t9)

  pvector overloaded operators test case
- BOOST_AUTO_TEST_CASE (s2t1)

  point multiplication test case
- BOOST_AUTO_TEST_CASE (s2t2)

  point division test case
- BOOST_AUTO_TEST_CASE (s2t3)

  point overloaded operators test case

7.49.1 Detailed Description

unit test suites

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

7.49.2 Macro Definition Documentation

7.49.2.1 BOOST_TEST_MODULE

```
#define BOOST_TEST_MODULE test_suites
```

Definition at line 8 of file test_suites.cpp.

7.49.3 Function Documentation

7.49.3.1 BOOST_AUTO_TEST_CASE() [1/12]

```
BOOST_AUTO_TEST_CASE (
             s1t1  )
```

pvector magnitude test case

Definition at line 22 of file test_suites.cpp.

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

7.49.3.2 BOOST_AUTO_TEST_CASE() [2/12]

```
BOOST_AUTO_TEST_CASE (
              s1t2 )
```

pvector mul test case

Definition at line 33 of file test_suites.cpp.

```
{
  pvector p1 = pvector(1, 1);
  p1.mul(3);
  pvector p2 = pvector(3, 3);
  BOOST_CHECK(p1 == p2);
}
```

7.49.3.3 BOOST_AUTO_TEST_CASE() [3/12]

```
BOOST_AUTO_TEST_CASE (
              s1t3 )
```

pvector div test case

Definition at line 44 of file test_suites.cpp.

```
{
  pvector p1 = pvector(5, 5);
  p1.div(5);
  pvector p2 = pvector(1, 1);
  BOOST_CHECK(p1 == p2);
}
```

7.49.3.4 BOOST_AUTO_TEST_CASE() [4/12]

```
BOOST_AUTO_TEST_CASE (
              s1t4 )
```

pvector dotproduct test case

Definition at line 55 of file test_suites.cpp.

```
{
  pvector p1 = pvector(1, 4);
  pvector p2 = pvector(3, 2);
  float dotProduct = p1.dotProduct(p2);
  BOOST_CHECK(dotProduct == 11);
}
```

7.49.3.5 BOOST_AUTO_TEST_CASE() [5/12]

```
BOOST_AUTO_TEST_CASE (
              s1t5 )
```

pvector angle between vectors test case

Definition at line 66 of file test_suites.cpp.

```
{
  pvector p1 = pvector(10, 10);
  pvector p2 = pvector(0, 10);
  float angle = p1.angleBetween(p2);
  BOOST_CHECK(angle == 45);
}
```

7.49.3.6 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.

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

7.49.3.7 BOOST_AUTO_TEST_CASE() [7/12]

```
BOOST_AUTO_TEST_CASE (
             s1t7  )
```

pvector normalize test case

Definition at line 87 of file test_suites.cpp.

```
{
  pvector p1 = pvector(2, 2);
  p1.normalize();
  float range = 0.01;
  BOOST_CHECK_CLOSE_FRACTION(0.707, p1.x, range);
  BOOST_CHECK_CLOSE_FRACTION(0.707, p1.y, range);
}
```

7.49.3.8 BOOST_AUTO_TEST_CASE() [8/12]

```
BOOST_AUTO_TEST_CASE (
             s1t8  )
```

pvector limit test case

Definition at line 99 of file test_suites.cpp.

```
{
  pvector p1 = pvector(2, 2);
  p1.limit(3);
  float range = 0.01;
  BOOST_CHECK_CLOSE_FRACTION(2.12, p1.x, range);
  BOOST_CHECK_CLOSE_FRACTION(2.12, p1.y, range);
}
```

7.49.3.9 BOOST_AUTO_TEST_CASE() [9/12]

```
BOOST_AUTO_TEST_CASE (
                s1t9  )
```

pvector overloaded operators test case

Definition at line 111 of file test_suites.cpp.

```
{
  pvector p1 = pvector(1, 1);
  p1 += pvector(1,1);
  BOOST_CHECK(p1 == pvector(2,2));
  p1 = pvector(1,1) + pvector(3,3);
  BOOST_CHECK(p1 == pvector(4,4));
  p1 = pvector(4,1) - pvector(3,3);
  BOOST_CHECK(p1 == pvector(1,-2));
  p1 = pvector(4,1) - point(3,3);
  BOOST_CHECK(p1 == pvector(1,-2));
  p1 = pvector(4,1) + point(3,3);
  BOOST_CHECK(p1 == pvector(7,4));
}
```

7.49.3.10 BOOST_AUTO_TEST_CASE() [10/12]

```
BOOST_AUTO_TEST_CASE (
                s2t1  )
```

point multiplication test case

Definition at line 133 of file test_suites.cpp.

```
{
  point p1 = point(1, 1);
  p1.mul(3);
  point p2 = point(3, 3);
  BOOST_CHECK(p1 == p2);
}
```

7.49.3.11 BOOST_AUTO_TEST_CASE() [11/12]

```
BOOST_AUTO_TEST_CASE (
                s2t2  )
```

point division test case

Definition at line 144 of file test_suites.cpp.

```
{
  point p1 = point(4, 4);
  p1.div(4);
  point p2 = point(1, 1);
  BOOST_CHECK(p1 == p2);
}
```

7.49.3.12 BOOST_AUTO_TEST_CASE() [12/12]

```
BOOST_AUTO_TEST_CASE (
                s2t3  )
```

point overloaded operators test case

Definition at line 155 of file test_suites.cpp.

```
{
  point p1 = point(1,1) + point(3,3);
  BOOST_CHECK(p1 == point(4,4));
  p1 = point(1,1) + pvector(3,3);
  BOOST_CHECK(p1 == point(4,4));
  pvector p2 = point(1,1) - point(3,3);
  BOOST_CHECK(p2 == pvector(-2,-2));
}
```

Index