Autonomous Steering Agents

Generated by Doxygen 1.8.17

Chapter 1

Intent

1- implementing Craig Raynolds autonomous steering agents

2- implementing genetics algorithms

3- implementing neural network

1.1 Dependencies

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

$sudo apt-get install libboost-all-dev

1.2 Resources

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

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

https://videotutorialsrock.com/index.php

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

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

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Class Documentation

5.1 agent Class Reference

`#include <agent.h>`

Collaboration diagram for agent:

Public Member Functions

- agent ()
  default constructor.
- agent (float x, float y)
  Constructor.
- ~agent ()
  agent destructor
- void updatePosition (bool arrive)
  calculates next position in each update using force applied
- void setFeatures (float s, float f, float r, float m)
  used to initialize the agent

Public Attributes

- string name
  name of the agent
- color fillColor
  color of the agent
- point position
  x and y coordinates 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 to apply
- pvector force
  total force to apply
- pvector acceleration
  added to velocity in each update
- pvector desiredVelocity
  get using target point and used to get steering force
- float r
  agent slows down as target point gets smaller than radius
- float mass
  used to get acceleration from force
- int id
  used to distinguish specific agent
- bool arrive = false
  defines if agent will have arriving behavior

5.1.1 Detailed Description

Definition at line 20 of file agent.h.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 agent() [1/2]

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

default constructor.

Creates new agent object.

See also

agent(float x, float y)

Definition at line 16 of file agent.cpp.

```
17 {
18
19 }
```

5.1.2.2 agent() [2/2]

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

Constructor.

Creates new agent object.

Parameters

x	position x of the agent
x	position y of the agent

See also

agent()

Definition at line 21 of file agent.cpp.

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

5.1.2.3 ~agent()

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

agent destructor

invokes when instance is killed

Definition at line 62 of file agent.cpp.

```
63 {
64
65 }
```

5.1.3 Member Function Documentation

5.1.3.1 setFeatures()

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

used to initialize the agent

setting parameters

Parameters

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

Definition at line 54 of file agent.cpp.

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

5.1.3.2 updatePosition()

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

calculates next position in each update using force applied

position update is invoked in periodically in a loop

Parameters

arrive	agent has arriving behavior or not

See also

agent()

Definition at line 33 of file agent.cpp.

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

Here is the call graph for this function:

5.1.4 Member Data Documentation

5.1.4.1 acceleration

```
pvector agent::acceleration
```

added to velocity in each update

acceleration to apply

Definition at line 120 of file agent.h.

5.1.4.2 arrive

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

defines if agent will have arriving behavior

arriving behavior

Definition at line 150 of file agent.h.

5.1.4.3 desiredVelocity

```
pvector agent::desiredVelocity
```

get using target point and used to get steering force

desired velocity to reach the target point

Definition at line 126 of file agent.h.

5.1.4.4 fillColor

```
color agent::fillColor
```

color of the agent

color information passed to graphics

Definition at line 72 of file agent.h.

5.1.4.5 force

```
pvector agent::force
```

total force to apply

force to apply to agent instance

Definition at line 114 of file agent.h.

5.1.4.6 id

```
int agent::id
```

used to distinguish specific agent

identification number of the agent

Definition at line 144 of file agent.h.

5.1.4.7 mass

```
float agent::mass
```

used to get acceleration from force

mass of the agent

Definition at line 138 of file agent.h.

5.1.4.8 maxForce

```
float agent::maxForce
```

maximum force of the agent

if force of the agent is more than this value, limit function restricts force

Definition at line 102 of file agent.h.

5.1.4.9 maxSpeed

```
float agent::maxSpeed
```

maximum speed of the agent

if velocity of the agent is more than this value, limit function restricts velocity

Definition at line 96 of file agent.h.

5.1.4.10 name

```
string agent::name
```

name of the agent

used to distinguish specific agent

Definition at line 66 of file agent.h.

5.1.4.11 position

```
point agent::position
```

x and y coordinates of the agent

position information

Definition at line 78 of file agent.h.

5.1.4.12 r

```
float agent::r
```

agent slows down as target point gets smaller than radius

radius for arrivin behavior

Definition at line 132 of file agent.h.

5.1.4.13 steering

```
pvector agent::steering
```

steering force to apply

steering force to change direction

Definition at line 108 of file agent.h.

5.1.4.14 targetPoint

```
point agent::targetPoint
```

target of the agent

calculated target point of the agent

Definition at line 90 of file agent.h.

5.1.4.15 velocity

```
pvector agent::velocity
```

velocity of the agent

velocity vector

Definition at line 84 of file agent.h.

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

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

5.2 color Class Reference

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

Collaboration diagram for color:

Public Member Functions

- color ()
  default constructor.
- color (float r, float g, float b)
  Constructor.
- void createColors ()
  fills colors vector with 8 main colors in color bar
- color getColor (int i)
  Constructor.

Public Attributes

- float R
  red condiment
- float G
  green condiment
- float B
  blue condiment
- vector< color > colors
  stores main colors

5.2.1 Detailed Description

Definition at line 20 of file color.h.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 color() [1/2]

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

default constructor.

Create a new color object.

See also

color(float r, float g, float b)

Definition at line 25 of file color.cpp.

```
26 {
27
28 }
```

5.2.2.2 color() [2/2]

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

Constructor.

Create a new color object.

Parameters

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

See also

path()

Definition at line 13 of file color.cpp.

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

5.2.3 Member Function Documentation

5.2.3.1 createColors()

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

fills colors vector with 8 main colors in color bar

creates main colors for future use

Definition at line 30 of file color.cpp.

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

5.2.3.2 getColor()

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

Constructor.

returns specified color from colors vector

Parameters

i	gets specified color

Returns

requested pre-created color instance

Definition at line 20 of file color.cpp.

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

Here is the caller graph for this function:

5.2.4 Member Data Documentation

5.2.4.1 B

`float color::B`

blue condiment

blue color ratio

Definition at line 69 of file color.h.

5.2.4.2 colors

`vector<color> color::colors`

stores main colors

vector of fundamental colors

Definition at line 75 of file color.h.

5.2.4.3 G

`float color::G`

green condiment

green color ratio

Definition at line 63 of file color.h.

5.2.4.4 R

```
float color::R
```

red condiment

red color ratio

Definition at line 57 of file color.h.

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

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

5.3 evade Class Reference

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

Inheritance diagram for evade:

Collaboration diagram for evade:

Public Member Functions

- evade ()

Static Public Member Functions

- static void loop ()

Additional Inherited Members

5.3.1 Detailed Description

Definition at line 8 of file evade.h.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 evade()

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

Definition at line 24 of file evade.cpp.

```
25 {
26     name = "evading";
27     createAgent(STATIC, nullptr, nullptr, nullptr);
28     callback = reinterpret_cast <void(*)()> ( (void *)(&loop) );
29 }
```

5.3.3 Member Function Documentation

5.3.3.1 loop()

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

Definition at line 8 of file evade.cpp.

```
9 {
10     for(auto it = agents.begin(); it < agents.end(); it++){
11          if((*it).name == "lion"){
12              (*it).targetPoint = view.getMousePosition();
13              (*it).force  = behavior.seek(*it);
14              (*it).arrive = true;
15          }
16          else{//gazelle
17              (*it).force  = behavior.evade(agents, *it, view);
18          }
19     }
20
21     refresh();
22 }
```

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

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

5.4 flee Class Reference

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

Inheritance diagram for flee:

Collaboration diagram for flee:

Public Member Functions

- flee ()

Static Public Member Functions

- static void loop ()

Additional Inherited Members

5.4.1 Detailed Description

Definition at line 8 of file flee.h.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 flee()

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

Definition at line 17 of file flee.cpp.

```
18 {
19     int agentCount = 196;
20     name = "fleeing troop";
21     createAgent(TROOP, &agentCount, nullptr, nullptr);
22     callback = reinterpret_cast <void(*)()> ( (void *)(&loop) );
23 }
```

5.4.3 Member Function Documentation

5.4.3.1 loop()

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

Definition at line 8 of file flee.cpp.

```
9 {
10     for(auto it = agents.begin(); it < agents.end(); it++){
11         (*it).force = behavior.flee((*it), view, view.getMousePosition());
12     }
13
14     refresh();
15 }
```

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

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

5.5 flock Class Reference

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

Inheritance diagram for flock:

Collaboration diagram for flock:

Public Member Functions

- flock ()

Static Public Member Functions

- static void loop ()

Additional Inherited Members

5.5.1 Detailed Description

Definition at line 8 of file flock.h.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 flock()

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

Definition at line 29 of file flock.cpp.

```
30 {
31     int agentCount = 50;
32     float maxForce = 0.3;
33     float maxSpeed = 0.8;
34     name = "flocking agents";
35     createAgent(RANDOM, &agentCount, &maxForce, &maxSpeed);
36     callback = reinterpret_cast <void(*)()> ( (void *)(&loop) );
37 }
```

5.5.3 Member Function Documentation

5.5.3.1 loop()

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

Definition at line 8 of file flock.cpp.

```
9 {
10     for(auto it = agents.begin(); it < agents.end(); it++){
11         view.forceInScreen((*it));
12
13         pvector sep = behavior.separation(agents, *it);
14         sep.mul(1.5);
15         pvector ali = behavior.align(agents, *it);
16         ali.mul(4);
17         pvector coh = behavior.cohesion(agents, *it);
18         coh.mul(0.1);
19
20         (*it).force = sep + ali + coh;
21         (*it).desiredVelocity = (*it).force + (*it).velocity;
22         (*it).targetPoint = (*it).position + (*it).desiredVelocity;
23         (*it).arrive = true;
24     }
25
26     refresh();
27 }
```

Here is the call graph for this function:

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

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

5.6 flowField Class Reference

`#include <flowField.h>`

Collaboration diagram for flowField:

Public Member Functions

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

5.6.1 Detailed Description

Definition at line 18 of file flowField.h.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 flowField() [1/2]

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

default constructor.

Create a new flowField object.

See also

flowField(pvector p)

Definition at line 15 of file flowField.cpp.

```
16 {
17
18 }
```

5.6.2.2 flowField() [2/2]

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

constructor.

Create a new flowField object.

Parameters

p	force vector

See also

flowField()

Definition at line 10 of file flowField.cpp.

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

5.6.3 Member Function Documentation

5.6.3.1 getField()

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

get force for individual pixel

get force for a specific position

Parameters

x	x cprovidesoordinate
y	y coordinate

Returns

returns force at specified position

Definition at line 39 of file flowField.cpp.

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

Here is the caller graph for this function:

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

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

5.7 graphics Class Reference

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

Collaboration diagram for graphics:

Public Member Functions

- void drawWall (float border, color color)
  draws wall
- void drawAgent (agent &agent, color &color)
  drawing agent
- void drawLine (point p1, point p2, color cl)
  drawing line
- void drawPath (path &path, color color)
  draws path that consists of points
- void drawPoint (point p)
  drawing point
- void drawCircle (point p, float radius)
  drawing circle
- void drawText (string text, point p)
  drawing text on screen
- void forceInScreen (agent &agent)
  changes agent position if it is out of screen
- void refreshScene ()
  position updates for all agents
- 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 function
- static void handleKeypress (unsigned char key, int x, int y)
  key press event of the openGL
- static void mouseButton (int button, int state, int x, int y)
  mouse press event of the openGL
- static void handleResize (int w, int h)
  event triggered after resizing
- static void mouseMove (int x, int y)
  event triggered after moving mouse

Static Public Attributes

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

5.7.1 Detailed Description

Definition at line 22 of file graphics.h.

5.7.2 Member Function Documentation

5.7.2.1 drawAgent()

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

drawing agent

draws agent and rotates it with its velocity

Parameters

agent	agent to draw
color	color of the agent

Definition at line 180 of file graphics.cpp.

```
181 {
182     glPushMatrix();
183     glTranslatef(agent.position.x, agent.position.y, 0.0f);
184     glRotatef(agent.velocity.getAngle(), 0.0f, 0.0f, 1.0f);
185     glBegin(GL_TRIANGLES);
186     glColor3f( color.R, color.G, color.B);
187     glVertex3f( 1.0f,  0.0f, 0.0f);
188     glVertex3f(-1.0f,  0.5f, 0.0f);
189     glVertex3f(-1.0f, -0.5f, 0.0f);
190     glEnd();
191     glPopMatrix();
192 }
```

Here is the call graph for this function:

5.7.2.2 drawCircle()

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

drawing circle

draws circle using openGL

Parameters

p	center of the circle
radius	radius of the circle

Definition at line 139 of file graphics.cpp.

```
140 {
141    glBegin(GL_LINE_STRIP);
142    glLineWidth(2);
143    for (int i = 0; i <= 300; i++) {
```

```
144       float angle = 2 * PI * i / 300;
145       float x = cos(angle) * radius;
146       float y = sin(angle) * radius;
147       glVertex2d(p.x + x, p.y + y);
148     }
149     glEnd();
150 }
```

5.7.2.3 drawLine()

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

drawing line

draws line with specified color

Parameters

p1	start point of the line
p2	end point of the line
color	color of the line

Definition at line 129 of file graphics.cpp.

```
130 {
131     glColor3f( cl.R, cl.G, cl.B);
132     glLineWidth(2);
133     glBegin(GL_LINES);
134     glVertex2f(p1.x, p1.y);
135     glVertex2f(p2.x, p2.y);
136     glEnd();
137 }
```

5.7.2.4 drawPath()

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

draws path that consists of points

draws path using lines

Parameters

path	path to draw
color	color of the path

Definition at line 115 of file graphics.cpp.

```
116 {
```

```
117     point p1, p2;
118     for(auto it = path.points.begin(); it < path.points.end()-1; it++){
119         p1 = point((*it).x, (*it).y - path.width/2) ;
120         p2 = point((*(it+1)).x, (*(it+1)).y - path.width/2);
121         drawLine(p1, p2, color.getColor(BLUE));
122 
123         p1 = point((*it).x, (*it).y + path.width/2) ;
124         p2 = point((*(it+1)).x, (*(it+1)).y + path.width/2);
125         drawLine(p1, p2, color.getColor(BLUE));
126     }
127 }
```

Here is the call graph for this function:

5.7.2.5 drawPoint()

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

drawing point

draws point using openGL

Parameters

p	point to draw

Definition at line 152 of file graphics.cpp.

```
153 {
154     glColor3f(1,1,1);
155     glPointSize(4.0);
156     glBegin(GL_POINTS);
157     glVertex2f(p.x, p.y);
158     glEnd();
159 }
```

Here is the caller graph for this function:

5.7.2.6 drawText()

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

drawing text on screen

draws text using openGL

Parameters

p	position of the text
text	text to display

Definition at line 22 of file graphics.cpp.

```
23 {
24     glColor3f (0.0, 0.0, 1.0);
25     //glRasterPos2f(-34, 32.5);
```

```
26     glRasterPos2f(p.x, p.y);
27     for ( string::iterator it=text.begin(); it!=text.end(); ++it){
28        glutBitmapCharacter(GLUT_BITMAP_9_BY_15, *it);
29     }
30 }
```

Here is the caller graph for this function:

5.7.2.7 drawWall()

```
void graphics::drawWall (
              float border,
              color color )
```

draws wall

draws square that consists of 4 lines

Parameters

border	position of the wall
color	color of the wall

Definition at line 161 of file graphics.cpp.

```
162 {
163     point p1 {-border,  border};
164     point p2 { border,  border};
165     drawLine(p1, p2, color.getColor(BLUE));
166
167     p1 = point ( border,  border);
168     p2 = point ( border, -border);
169     drawLine(p1, p2, color.getColor(BLUE));
170
171     p1 = point (  border, -border);
172     p2 = point ( -border, -border);
173     drawLine(p1, p2, color.getColor(BLUE));
174
175     p1 = point (-border,  border);
176     p2 = point (-border, -border);
177     drawLine(p1, p2, color.getColor(BLUE));
178 }
```

Here is the call graph for this function:

5.7.2.8 forceInScreen()

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

changes agent position if it is out of screen

makes the agent stay in screen

Parameters

agent	agent to be in screen

Definition at line 64 of file graphics.cpp.

```
65 {
66     if(agent.position.x > WIDTH)
67        agent.position.x -= 2 * WIDTH;
68     if(agent.position.x < -WIDTH)
69        agent.position.x += 2 * WIDTH;
70     if(agent.position.y > HEIGHT)
71        agent.position.y -= 2 * HEIGHT;
72     if(agent.position.y < -HEIGHT)
73        agent.position.y += 2 * HEIGHT;
74 }
```

5.7.2.9 getMousePosition()

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

gets mouse position

used to get mouse position

Definition at line 59 of file graphics.cpp.

```
60 {
61     return point (graphics::target_x, graphics::target_y);
62 }
```

Here is the call graph for this function:

5.7.2.10 handleKeypress()

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

key press event of the openGL

openGL key press event

Parameters

key	key
x	unused but required for openGL
y	unused but required for openGL

Definition at line 108 of file graphics.cpp.

```
109 {
110     if (key == ESC){
111         exit(0);
112     }
113 }
```

Here is the caller graph for this function:

5.7.2.11 handleResize()

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

event triggered after resizing

openGL screeen resize event

Parameters

w	width of the screen
h	height of the screen

Definition at line 84 of file graphics.cpp.

```
85 {
86     glViewport(0, 0, w, h);  //Tell OpenGL how to convert from coordinates to pixel values
87     glMatrixMode(GL_PROJECTION); //Switch to setting the camera perspective
88     glLoadIdentity(); //Reset the camera
89     //Set the camera perspective
90     gluPerspective(45.0,                   //The camera angle
91                    (double)w / (double)h, //The width-to-height ratio
92                    1.0,                   //The near z clipping coordinate
93                    200.0);                //The far z clipping coordinate
94 }
```

Here is the caller graph for this function:

5.7.2.12 initGraphics()

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

initialization of graphics

used to init graphics

Parameters

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

Definition at line 42 of file graphics.cpp.

```
43 {
44     glutInit(argv, argc);
45     glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
46     glutInitWindowSize(400, 400);
47     glutCreateWindow("Autonomous Steering Agents");
48     glClearColor(0.7f, 0.7f, 0.7f, 1.0f); //set background color
49     glEnable(GL_DEPTH_TEST);
50     glutDisplayFunc(*callback);
51     glutMouseFunc(graphics::mouseButton);
52     glutPassiveMotionFunc(graphics::mouseMove);
53     glutKeyboardFunc(graphics::handleKeypress);
54     glutReshapeFunc(graphics::handleResize);
55     glutTimerFunc(20, graphics::timerEvent, 0);
56     glutMainLoop();
57 }
```

Here is the call graph for this function:

5.7.2.13 mouseButton()

```
void graphics::mouseButton (
```

```
            int button,
            int state,
            int x,
            int y )  [static]
```

mouse press event of the openGL

openGL key mouss press event

Parameters

button	mouse button
x	unused but required for openGL
y	unused but required for openGL

Definition at line 102 of file graphics.cpp.

```
103 {
104     if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN){
105     }
106 }
```

Here is the caller graph for this function:

5.7.2.14 mouseMove()

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

event triggered after moving mouse

openGL mouse move event

Parameters

x	x position of the mouse
y	y position of the mouse

Definition at line 76 of file graphics.cpp.

```
77 {
78     //TODO: mouse position to glut
79     //TODO: magic numbers
80     graphics::target_x = x / 5.88 - 34;
81     graphics::target_y = 34 - y / 5.88;
82 }
```

Here is the caller graph for this function:

5.7.2.15 refreshScene()

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

position updates for all agents

refresh screen for every existing object

Definition at line 33 of file graphics.cpp.

```
34 {
35     glutSwapBuffers();
36     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
37     glMatrixMode(GL_MODELVIEW); //Switch to the drawing perspective
38     glLoadIdentity(); //Reset the drawing perspective
39     glTranslatef(0.0f, 0.0f, -85.0f); //Move to the center of the triangle
40 }
```

5.7.2.16 timerEvent()

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

periodic timer event function

openGL timer event callback

Parameters

value	period as ms

Definition at line 96 of file graphics.cpp.

```
97 {
98     glutPostRedisplay(); //Tell GLUT that the display has changed
99     glutTimerFunc(value, timerEvent, 20);
100 }
```

Here is the caller graph for this function:

5.7.3 Member Data Documentation

5.7.3.1 target_x

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

mouse position x

holds mouse y position

Definition at line 153 of file graphics.h.

5.7.3.2 target_y

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

mouse position y

holds mouse x position

Definition at line 159 of file graphics.h.

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

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

5.8 mouseFollower Class Reference

`#include <mouseFollower.h>`

Inheritance diagram for mouseFollower:

Collaboration diagram for mouseFollower:

Public Member Functions

- mouseFollower ()

Static Public Member Functions

- static void loop ()

Additional Inherited Members

5.8.1 Detailed Description

Definition at line 8 of file mouseFollower.h.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 mouseFollower()

`mouseFollower::mouseFollower ( )`

Definition at line 18 of file mouseFollower.cpp.

```
19 {
20     int agentCount = 30;
21     float maxForce = 0.3;
22     float maxSpeed = 0.6;
23     name = "mouse following";
24     createAgent(RANDOM, &agentCount, &maxForce, &maxSpeed);
25     callback = reinterpret_cast <void(*)()> ( (void *)(&loop) );
26 }
```

5.8.3 Member Function Documentation

5.8.3.1 loop()

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

Definition at line 8 of file mouseFollower.cpp.

```
9 {
10     for(auto it = agents.begin(); it < agents.end(); it++){
11         (*it).targetPoint = view.getMousePosition();
12         (*it).force  = behavior.seek(*it);
13         (*it).arrive = true;
14     }
15     refresh();
16 }
```

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

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

5.9 obstacle Class Reference

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

Collaboration diagram for obstacle:

Public Member Functions

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

Public Attributes

- point p
  x and y coordinates
- float r
  the bigger radius the bigger the obstacle

5.9.1 Detailed Description

Definition at line 12 of file obstacle.h.

5.9.2 Constructor & Destructor Documentation

5.9.2.1 obstacle() [1/2]

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

default constructor.

create a new obstacle object.

See also

obstacle(point p, float r

Definition at line 15 of file obstacle.cpp.

```
16 {
17
18 }
```

5.9.2.2 obstacle() [2/2]

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

constructor

create a new obstacle object.

Parameters

p	center of the circular obstacle
r	radius of the obstacle

See also

obstacle(point p, float r);

Definition at line 20 of file obstacle.cpp.

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

5.9.3 Member Data Documentation

5.9.3.1 p

```
point obstacle::p
```

x and y coordinates

center point of the obstacle

Definition at line 34 of file obstacle.h.

5.9.3.2 r

```
float obstacle::r
```

the bigger radius the bigger the obstacle

radius of the obstacle

Definition at line 40 of file obstacle.h.

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

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

5.10 obstacleAvoidance Class Reference

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

Inheritance diagram for obstacleAvoidance:

Collaboration diagram for obstacleAvoidance:

Public Member Functions

- obstacleAvoidance ()

Static Public Member Functions

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

Static Public Attributes

- static vector< obstacle > obstacles

Additional Inherited Members

5.10.1 Detailed Description

Definition at line 9 of file obstacleAvoidance.h.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 obstacleAvoidance()

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

Definition at line 36 of file obstacleAvoidance.cpp.

```
37 {
38     name = "avoid obstacles";
39     createAgent(STATIC, nullptr, nullptr, nullptr);
40     createObstacle(obstacles);
41     callback = reinterpret_cast <void(*)()> ( (void *)(&loop) );
42 }
```

5.10.3 Member Function Documentation

5.10.3.1 createObstacle()

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

Definition at line 29 of file obstacleAvoidance.cpp.

```
30 {
31     obstacles.push_back(obstacle(point(0,0), 8));
32     obstacles.push_back(obstacle(point(-20,0), 3));
33     obstacles.push_back(obstacle(point(20,-10), 4));
34 }
```

Here is the call graph for this function:

5.10.3.2 loop()

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

Definition at line 10 of file obstacleAvoidance.cpp.

```
11 {
12     for(auto it = agents.begin(); it < agents.end(); it++){
13         for(auto it = obstacles.begin(); it < obstacles.end(); it++){
14             point p = (*it).p;
15             view.drawCircle(p, (*it).r);
16         }
17
18         (*it).targetPoint = view.getMousePosition();
19         pvector seek  = behavior.seek(*it);
20         seek.mul(0.5);
21
22         pvector avoid = behavior.avoid(obstacles, *it);
23         (*it).force = avoid + seek;
24         (*it).arrive = true;
25     }
26     refresh();
27 }
```

Here is the call graph for this function:

5.10.4 Member Data Documentation

5.10.4.1 obstacles

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

Definition at line 13 of file obstacleAvoidance.h.

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

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

5.11 path Class Reference

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

Collaboration diagram for path:

Public Member Functions

- path ()
  Default constructor.
- path (float width)
  Constructor.
- void addPoint (point p)
  adds a new point to the path

Public Attributes

- vector< point > points
  points added to the path
- int width
  defines width of the path

5.11.1 Detailed Description

Definition at line 15 of file path.h.

5.11.2 Constructor & Destructor Documentation

5.11.2.1 path() [1/2]

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

Default constructor.

Create a new path object.

See also

path(float width)

Definition at line 16 of file path.cpp.

```
17 {
18
19 }
```

5.11.2.2 path() [2/2]

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

Constructor.

Create a new path object.

Parameters

width	The width of the path.

See also

path()

Definition at line 21 of file path.cpp.

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

5.11.3 Member Function Documentation

5.11.3.1 addPoint()

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

adds a new point to the path

Used when customizing path

Parameters

point	new point to add to the path

Definition at line 11 of file path.cpp.

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

Here is the caller graph for this function:

5.11.4 Member Data Documentation

5.11.4.1 points

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

points added to the path

path is created from these points

Definition at line 43 of file path.h.

5.11.4.2 width

```
int path::width
```

defines width of the path

path width

Definition at line 49 of file path.h.

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

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

5.12 pathFollower Class Reference

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

Inheritance diagram for pathFollower:

Collaboration diagram for pathFollower:

Public Member Functions

- pathFollower ()

Static Public Member Functions

- static void loop ()
- static void createPath (path &p)

Static Public Attributes

- static path myPath

Additional Inherited Members

5.12.1 Detailed Description

Definition at line 8 of file pathFollower.h.

5.12.2 Constructor & Destructor Documentation

5.12.2.1 pathFollower()

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

Definition at line 30 of file pathFollower.cpp.

```
31 {
32     int agentCount = 40;
33     float maxForce = 0.2;
34     float maxSpeed = 0.4;
35     myPath = path(8);
36     createPath(myPath);
37     name = "path following";
38     createAgent(RANDOM, &agentCount, &maxForce, &maxSpeed);
39     callback = reinterpret_cast <void(*)()> ( (void *)(&loop) );
40 }
```

5.12.3 Member Function Documentation

5.12.3.1 createPath()

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

Definition at line 22 of file pathFollower.cpp.

```
23 {
24     p.addPoint(point(-40,  5));
25     p.addPoint(point(-14, 15));
26     p.addPoint(point( 10,  7));
27     p.addPoint(point( 40, 12));
28 }
```

Here is the call graph for this function:

5.12.3.2 loop()

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

Definition at line 10 of file pathFollower.cpp.

```
11 {
12     for(auto it = agents.begin(); it < agents.end(); it++){
13         view.drawPath(myPath, myColor);
14         pvector seek = behavior.stayInPath(*it, myPath, view);
15         pvector sep  = behavior.separation(agents, *it);
16         sep.mul(5);
17         (*it).force = sep + seek;
18     }
19     refresh();
20 }
```

Here is the call graph for this function:

5.12.4 Member Data Documentation

5.12.4.1 myPath

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

Definition at line 12 of file pathFollower.h.

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

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

5.13 point Class Reference

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

Collaboration diagram for point:

Public Member Functions

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

Public Attributes

- float x
  x position of the point
- float y
  y position of the point

5.13.1 Detailed Description

Definition at line 15 of file point.h.

5.13.2 Constructor & Destructor Documentation

5.13.2.1 point() [1/2]

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

default constructor

create a new point instance

See also

point(float x, float y)

Definition at line 21 of file point.cpp.

```
21 {}
```

Here is the caller graph for this function:

5.13.2.2 point() [2/2]

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

constructor

create a new point instance

Parameters

x	position x of the point
y	position y of the point

See also

point()

Definition at line 15 of file point.cpp.

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

5.13.3 Member Function Documentation

5.13.3.1 div()

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

divide point

helper function to divide point position

Parameters

d	scalar to divide position of the point

Definition at line 38 of file point.cpp.

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

Here is the caller graph for this function:

5.13.3.2 getNormalPoint()

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

gets a points normal point on a vector

provides normal point on a vector of a point

Parameters

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

Definition at line 67 of file point.cpp.

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

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

5.13.3.3 mul()

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

multiply point

helper function to multiply point position

Parameters

d	scalar to multiply position of the point

Definition at line 44 of file point.cpp.

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

Here is the caller graph for this function:

5.13.3.4 operator+() [1/2]

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

used between point and point

overloaded + operator

Parameters

obj	point to add

Returns

substracted result

Definition at line 51 of file point.cpp.

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

5.13.3.5 operator+() [2/2]

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

used between vector and point

overloaded + operator

Parameters

obj	vector to add

Returns

substracted result

Definition at line 23 of file point.cpp.

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

5.13.3.6 operator-()

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

used between point and point

overloaded - operator

Parameters

obj	point to substract

Returns

substracted result

Definition at line 59 of file point.cpp.

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

5.13.3.7 operator==()

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

used between point and point

overloaded == operator

Parameters

obj	point to compare

Returns

true or false

Definition at line 31 of file point.cpp.

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

5.13.3.8 print()

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

debug function

prints position of the point

Parameters

s	explanation string of the log

Definition at line 79 of file point.cpp.

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

5.13.4 Member Data Documentation

5.13.4.1 x

```
float point::x
```

x position of the point

x coordinate

Definition at line 99 of file point.h.

5.13.4.2 y

```
float point::y
```

y position of the point

y coordinate

Definition at line 105 of file point.h.

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

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

5.14 prison Class Reference

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

Inheritance diagram for prison:

Collaboration diagram for prison:

Public Member Functions

- prison ()

Static Public Member Functions

- static void loop ()

Additional Inherited Members

5.14.1 Detailed Description

Definition at line 8 of file prison.h.

5.14.2 Constructor & Destructor Documentation

5.14.2.1 prison()

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

Definition at line 21 of file prison.cpp.

```
22 {
23     int agentCount = 30;
24     float maxForce = 0.5;
25     float maxSpeed = 0.6;
26 
27     name = "stay in prison";
28     createAgent(RANDOM, &agentCount, &maxForce, &maxSpeed);
29     callback = reinterpret_cast <void(*)()> ( (void *)(&loop) );
30 }
```

5.14.3 Member Function Documentation

5.14.3.1 loop()

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

Definition at line 11 of file prison.cpp.

```
12 {
13     for(auto it = agents.begin(); it < agents.end(); it++){
14          view.drawWall(WALL, myColor);
15          (*it).force  = behavior.stayInArea(*it, WALL - DISTANCE);
16          (*it).force += behavior.separation(agents, *it);
17     }
18     refresh();
19 }
```

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

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

5.15 pursuit Class Reference

`#include <pursuit.h>`

Inheritance diagram for pursuit:

Collaboration diagram for pursuit:

Public Member Functions

- pursuit ()

Static Public Member Functions

- static void loop ()

Additional Inherited Members

5.15.1 Detailed Description

Definition at line 8 of file pursuit.h.

5.15.2 Constructor & Destructor Documentation

5.15.2.1 pursuit()

`pursuit::pursuit ( )`

Definition at line 24 of file pursuit.cpp.

```
25 {
26     name = "pursuit";
27     createAgent(STATIC, nullptr, nullptr, nullptr);
28     callback = reinterpret_cast <void(*)()> ( (void *)(&loop) );
29 }
```

5.15.3 Member Function Documentation

5.15.3.1 loop()

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

Definition at line 8 of file pursuit.cpp.

```
9 {
10     for(auto it = agents.begin(); it < agents.end(); it++){
11          if((*it).name == "gazelle"){
12             (*it).targetPoint = view.getMousePosition();
13             (*it).force  = behavior.seek(*it);
14          }
15          else{//lion
16             (*it).force  = behavior.pursuit(agents, *it, view);
17          }
18          (*it).arrive = true;
19     }
20
21     refresh();
22 }
```

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

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

5.16 pvector Class Reference

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

Collaboration diagram for pvector:

Public Member Functions

- pvector ()
  default constructor
- pvector (float x, float y)
  constructor
- float magnitude ()
  calculates magnitude of the vector
- pvector & normalize ()
  normalize vector
- void div (float i)
  divides vector by given scalar value
- void mul (float i)
  multiplies vector by given scalar value
- void add (pvector p)
  addition of vectors
- void limit (float limit)
  limits vector with the given parameter
- float getAngle ()
  get angle using its x and y magnitudes
- float dotProduct (pvector v)
  dot product of two vectors
- float angleBetween (pvector v)
  angle is calculated using dot product

- void print (const string &s)
  debug function
- pvector operator+= (pvector const &obj)
  used between vectors
- pvector operator+ (pvector const &obj)
  used between vectors
- pvector operator- (pvector const &obj)
  used between vectors
- pvector operator- (point const &obj)
  used between vector and point
- pvector operator+ (point const &obj)
  used between vector and point
- bool operator== (pvector const &obj)
  used between vectors

Public Attributes

- float x
  used between vector and point
- float y
  used between vector and point

5.16.1 Detailed Description

Definition at line 17 of file pvector.h.

5.16.2 Constructor & Destructor Documentation

5.16.2.1 pvector() [1/2]

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

default constructor

create a new pvector instance

See also

pvector(float x, float y)

Definition at line 35 of file pvector.cpp.

```
36 {
37
38 }
```

5.16.2.2 pvector() [2/2]

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

constructor

create a new pvector instance

Parameters

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

See also

pvector()

Definition at line 40 of file pvector.cpp.

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

5.16.3 Member Function Documentation

5.16.3.1 add()

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

addition of vectors

vector addition

Parameters

p	vector to add

Definition at line 58 of file pvector.cpp.

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

5.16.3.2 angleBetween()

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

angle is calculated using dot product

angle calculation between two vectors

Parameters

v	vector to calculate angle

Returns

angle value between two vectors

Definition at line 23 of file pvector.cpp.

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

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

5.16.3.3 div()

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

divides vector by given scalar value

vector division

Parameters

i	scalar value to divide

Definition at line 46 of file pvector.cpp.

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

Here is the caller graph for this function:

5.16.3.4 dotProduct()

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

dot product of two vectors

dot product calculation

Parameters

v	vector to calculate dot product

Returns

returns scalar dot product value

Definition at line 30 of file pvector.cpp.

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

Here is the caller graph for this function:

5.16.3.5 getAngle()

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

get angle using its x and y magnitudes

calculates vector angle

Returns

angle of the vector

Definition at line 16 of file pvector.cpp.

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

Here is the caller graph for this function:

5.16.3.6 limit()

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

limits vector with the given parameter

vector limitation

Parameters

limit	upper limit to restrict vector

Definition at line 83 of file pvector.cpp.

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

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

5.16.3.7 magnitude()

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

calculates magnitude of the vector

uses pisagor theorem for magnitude calculation

Returns

magnitude of the vector

Definition at line 64 of file pvector.cpp.

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

Here is the caller graph for this function:

5.16.3.8 mul()

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

multiplies vector by given scalar value

vector multiplication

Parameters

i	scalar value to multiply

Definition at line 52 of file pvector.cpp.

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

Here is the caller graph for this function:

5.16.3.9 normalize()

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

normalize vector

divides vector by magnitude

Returns

normalized vector

Definition at line 69 of file pvector.cpp.

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

Here is the caller graph for this function:

5.16.3.10 operator+() [1/2]

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

used between vector and point

overloaded + operator

Parameters

obj	point to add

Returns

sum

Definition at line 111 of file pvector.cpp.

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

5.16.3.11 operator+() [2/2]

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

used between vectors

overloaded + operator

Parameters

obj	vector to add

Returns

sum of vectors

Definition at line 89 of file pvector.cpp.

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

5.16.3.12 operator+=()

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

used between vectors

overloaded += operator

Parameters

obj	vector to add

Returns

sum of vectors

Definition at line 97 of file pvector.cpp.

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

5.16.3.13 operator-() [1/2]

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

used between vector and point

overloaded - operator

Parameters

obj	point to substract

Returns

difference

Definition at line 119 of file pvector.cpp.

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

5.16.3.14 operator-() [2/2]

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

used between vectors

overloaded - operator

Parameters

obj	vector to substract

Returns

difference of vectors

Definition at line 132 of file pvector.cpp.

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

5.16.3.15 operator==()

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

used between vectors

overloaded == operator

Parameters

obj	vector to check if equal

Returns

true or false

Definition at line 104 of file pvector.cpp.

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

5.16.3.16 print()

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

debug function

prints position of the vector

Parameters

s	explanation string of the log

Definition at line 127 of file pvector.cpp.

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

5.16.4 Member Data Documentation

5.16.4.1 x

```
float pvector::x
```

used between vector and point

x magnitude of the vector

Definition at line 159 of file pvector.h.

5.16.4.2 y

```
float pvector::y
```

used between vector and point

y magnitude of the vector

Definition at line 165 of file pvector.h.

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

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

5.17 random Class Reference

`#include <random.h>`

Collaboration diagram for random:

Static Public Member Functions

- static void createRandomArray (int *arr, int size)
  generates random array usin swap between its elements

5.17.1 Detailed Description

Definition at line 9 of file random.h.

5.17.2 Member Function Documentation

5.17.2.1 createRandomArray()

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

generates random array usin swap between its elements

random array generation

Parameters

arr	int array that will include random values
size	size of the array

Definition at line 14 of file random.cpp.

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

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

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

5.18 scenario Class Reference

`#include <scenario.h>`

Inheritance diagram for scenario:

Collaboration diagram for scenario:

Public Member Functions

- scenario ()
- void createAgent (int type, int *count, float *force, float *speed)
- void initGL (int *argv, char **argc)

Static Public Member Functions

- static void refresh ()

Public Attributes

- void(* callback)()

Static Public Attributes

- static vector< agent > agents
- static graphics view
- static steeringBehavior behavior
- static color myColor
- static string name

5.18.1 Detailed Description

Definition at line 12 of file scenario.h.

5.18.2 Constructor & Destructor Documentation

5.18.2.1 scenario()

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

Definition at line 21 of file scenario.cpp.

```
22 {
23     srand(time(NULL));
24     myColor.createColors();
25     view = graphics();
26 }
```

5.18.3 Member Function Documentation

5.18.3.1 createAgent()

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

Definition at line 98 of file scenario.cpp.

```
99 {
100     if(type == TROOP){
101         createTroop(*count);
102     }
103     else if(type == RANDOM){
104         createRandomAgents(*count, *force, *speed);
105     }
106     else if(type == STATIC){
107         createStaticAgents();
108     }
109     else{
110         //error message
111     }
112 }
```

5.18.3.2 initGL()

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

Definition at line 15 of file scenario.cpp.

```
16 {
17     view.initGraphics(argc, argv, callback);
18 }
```

Here is the caller graph for this function:

5.18.3.3 refresh()

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

Definition at line 28 of file scenario.cpp.

```
28                      {
29     for(auto it = agents.begin(); it < agents.end(); it++){
30        (*it).updatePosition((*it).arrive);
31        view.drawAgent(*it, (*it).fillColor);
32     }
33
34     view.drawText(name, point(-34, 32.25)); //TODO: magic numbers, define left corner
35     view.refreshScene();
36 }
```

Here is the call graph for this function:

5.18.4 Member Data Documentation

5.18.4.1 agents

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

Definition at line 18 of file scenario.h.

5.18.4.2 behavior

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

Definition at line 20 of file scenario.h.

5.18.4.3 callback

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

Definition at line 23 of file scenario.h.

5.18.4.4 myColor

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

Definition at line 21 of file scenario.h.

5.18.4.5 name

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

Definition at line 22 of file scenario.h.

5.18.4.6 view

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

Definition at line 19 of file scenario.h.

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

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

5.19 steeringBehavior Class Reference

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

Collaboration diagram for steeringBehavior:

Public Member Functions

- pvector stayInArea (agent &agent, int turnPoint)
- pvector inFlowField (agent &agent, flowField &flow)
- pvector stayInPath (agent &agent, path &path, graphics view)
- pvector seek (agent &agent)
- pvector separation (vector< agent > agents, agent &agent)
- pvector cohesion (vector< agent > boids, agent &agent)
- pvector align (vector< agent > boids, agent &agent)
- pvector wander (agent &agent)
- pvector pursuit (vector< agent > boids, agent &pursuer, graphics view)
- pvector evade (vector< agent > boids, agent &evader, graphics view)
- pvector flee (agent &agent, graphics &view, point p)
- pvector avoid (vector< obstacle > obstacles, agent &agent)
- void setAngle (pvector &p, float angle)

5.19.1 Detailed Description

Definition at line 28 of file steeringBehavior.h.

5.19.2 Member Function Documentation

5.19.2.1 align()

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

Definition at line 110 of file steeringBehavior.cpp.

```
111 {
112    float neighborDist = 30; //TODO: magic numer
113    pvector sum {0,0};
114    int count = 0;
115    for(auto it = boids.begin(); it < boids.end(); it++){
116       float d = (agent.position - (*it).position).magnitude();
117       if( (d >0) && (d < neighborDist) ){
118          sum += (*it).velocity;
119          count++;
120       }
121    }
122    if(count>0){
123       sum.div(count);
124       sum.normalize().mul(agent.maxSpeed);
125       agent.steering = sum - agent.velocity;
126       return agent.steering;
127    }
128    return pvector(0,0);
129 }
```

Here is the call graph for this function:

5.19.2.2 avoid()

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

Definition at line 174 of file steeringBehavior.cpp.

```
175 {
176    float dynamic_length = agent.velocity.magnitude() / agent.maxSpeed;
177    pvector vel = agent.velocity;
178    vel.normalize().mul(dynamic_length);
179    pvector ahead  = vel + agent.position;
180    vel.mul(6);
181    pvector ahead2 = vel + agent.position;
182    //view.drawPoint(point(ahead.x, ahead.y));
183    //view.drawPoint(point(ahead2.x, ahead2.y));
184
185    for(auto it = obstacles.begin(); it < obstacles.end(); it++){
186       float dist  = (ahead  - (*it).p).magnitude();
187       float dist2 = (ahead2 - (*it).p).magnitude();
188       if(dist < (*it).r + 2 || dist2 < (*it).r + 2){
189          pvector avoidance = ahead - (*it).p;
190          avoidance.normalize().mul(20);
191          /*a = point(avoidance.x, avoidance.y);
192          view.drawLine(agent.position, agent.position + a, color(0,1,0));*/
193          return avoidance;
194       }
195    }
196    return pvector(0,0);
197 }
```

Here is the call graph for this function:

5.19.2.3 cohesion()

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

Definition at line 131 of file steeringBehavior.cpp.

```
132 {
133    float neighborDist = 20; //TODO: magic numer
134    point sum {0,0};
135    int count = 0;
136    for(auto it = boids.begin(); it < boids.end(); it++){
137       float d = (agent.position - (*it).position).magnitude();
138       if( (d >0) && (d < neighborDist) ){
139          sum = sum + (*it).position;
140          count++;
141       }
142    }
143    if(count>0){
144       sum.div(count);
145       agent.targetPoint = sum;
146       return seek(agent);
147    }
148    return pvector(0,0);
149 }
```

Here is the call graph for this function:

5.19.2.4 evade()

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

Definition at line 38 of file steeringBehavior.cpp.

```
39 {
40    agent target;
41    for(auto it = boids.begin(); it < boids.end(); it++){
42       if((*it).name == "lion"){
43          target = *it;
44       }
45    }
46
47    point p = point(evader.position.x + 2, evader.position.y - 2);
48    view.drawText(evader.name, p);
49    p = point(target.position.x + 2, target.position.y - 2);
50    view.drawText(target.name, p);
51
52    pvector targetVel = target.velocity;
53    targetVel.mul(5);//TODO: magic number
54
55    point futurePos = target.position + targetVel;
56    view.drawPoint(futurePos);
57
58    pvector dist = evader.position - futurePos;
59    dist.normalize().mul( 1 / dist.magnitude() );
60
61    evader.targetPoint = evader.position + dist;
62    return flee(evader, view, futurePos);
63 }
```

Here is the call graph for this function:

5.19.2.5 flee()

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

Definition at line 21 of file steeringBehavior.cpp.

```
22 {
23    pvector dist = agent.targetPoint - p;
24    view.drawPoint(agent.targetPoint);
25
26    if(dist.magnitude() < 15){  //TODO: magic number
```

```
27       agent.arrive = false;
28       agent.desiredVelocity = agent.position - p;
29    }
30    else{
31       agent.arrive = true;
32       agent.desiredVelocity = agent.targetPoint - agent.position;
33    }
34    agent.steering = agent.desiredVelocity - agent.velocity;
35    return agent.steering;
36 }
```

Here is the call graph for this function:

5.19.2.6 inFlowField()

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

Definition at line 229 of file steeringBehavior.cpp.

```
230 {
231     //pos_x, pos_y must be non negative integer
232     int pos_x = abs((int)agent.position.x) % WIDTH;
233     int pos_y = abs((int)agent.position.y) % HEIGHT;
234     //TODO: modification required for non uniform fields
235     return flow.getField(pos_x, pos_y);
236 }
```

Here is the call graph for this function:

5.19.2.7 pursuit()

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

Definition at line 65 of file steeringBehavior.cpp.

```
66 {
67    agent target;
68    for(auto it = boids.begin(); it < boids.end(); it++){
69       if((*it).name == "gazelle"){
70          target = *it;
71       }
72    }
73
74    point p = point(target.position.x + 2, target.position.y - 2);
75    view.drawText(target.name, p);
76    p = point(pursuer.position.x + 2, pursuer.position.y - 2);
77    view.drawText(pursuer.name, p);
78
79    float dist = (target.position - pursuer.position).magnitude();
80    float t = dist / target.maxSpeed;
81
82    pvector targetVel = target.velocity;
83    targetVel.mul(t);
84    point futurePos = target.position + targetVel;
85    pursuer.targetPoint = futurePos;
86    return seek(pursuer);
87 }
```

Here is the call graph for this function:

5.19.2.8 seek()

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

Definition at line 199 of file steeringBehavior.cpp.

```
200 {
201    agent.desiredVelocity = agent.targetPoint - agent.position;
202    agent.steering = agent.desiredVelocity - agent.velocity;
203    return agent.steering;
204 }
```

5.19.2.9 separation()

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

Definition at line 151 of file steeringBehavior.cpp.

```
152 {
153    float desiredSeparation = 5; //TODO: magic number
154    pvector sum = pvector(0,0);
155    int count = 0;
156    for(auto it = agents.begin(); it < agents.end(); it++){
157       float d = (agent.position - (*it).position).magnitude();
158       if( (d > 0) && (d < desiredSeparation) ){
159          pvector diff = agent.position - (*it).position;
160          diff.normalize().div(d);
161          sum = sum + diff;
162          count++;
163       }
164    }
165    if(count > 0){
166       sum.div(count);
167       sum.normalize().mul(agent.maxSpeed);
168       agent.steering = sum - agent.velocity;
169       return agent.steering;
170    }
171    return pvector(0,0);
172 }
```

Here is the call graph for this function:

5.19.2.10 setAngle()

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

Definition at line 15 of file steeringBehavior.cpp.

```
16 {
17    p.x = cos ( angle * PI / 180.0 );
18    p.y = sin ( angle * PI / 180.0 );
19 }
```

5.19.2.11 stayInArea()

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

Definition at line 238 of file steeringBehavior.cpp.

```
239 {
240    if(agent.position.x >= turnPoint){
241       agent.desiredVelocity = pvector( -agent.maxSpeed, agent.velocity.y );
242       agent.steering = agent.desiredVelocity - agent.velocity;
243       return agent.steering;
244    }
245    else if(agent.position.x <= -turnPoint){
246       agent.desiredVelocity = pvector( agent.maxSpeed, agent.velocity.y );
247       agent.steering = agent.desiredVelocity - agent.velocity;
248       return agent.steering;
249    }
250    else if(agent.position.y >= turnPoint){
251       agent.desiredVelocity = pvector( agent.velocity.x, -agent.maxSpeed );
252       agent.steering = agent.desiredVelocity - agent.velocity;
253       return agent.steering;
254    }
255    else if(agent.position.y <= -turnPoint){
256       agent.desiredVelocity = pvector( agent.velocity.x, agent.maxSpeed );
257       agent.steering = agent.desiredVelocity - agent.velocity;
258       return agent.steering;
259    }
260    return pvector(0,0);
261 }
```

5.19.2.12 stayInPath()

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

Definition at line 206 of file steeringBehavior.cpp.

```
207 {
208    float worldRecord = 1000000; //TODO: magic number
209    point normalPoint, predictedPos, start, end;
210    pvector distance;
211    for(auto it = path.points.begin(); it < path.points.end()-1; it++){
212       start = point((*it).x, (*it).y);
213       end   = point((*(it+1)).x, (*(it+1)).y);
214       predictedPos = agent.position + agent.velocity;
215       normalPoint.getNormalPoint(predictedPos, start, end);
216       if (normalPoint.x < start.x || normalPoint.x > end.x){
217          normalPoint = end;
218       }
219       distance = predictedPos - normalPoint;
220       if (distance.magnitude() < worldRecord){
221          worldRecord = distance.magnitude();
222          agent.targetPoint = end;
223       }
224       view.drawPoint(agent.targetPoint);
225    }
226    return seek(agent);
227 }
```

Here is the call graph for this function:

5.19.2.13 wander()

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

Definition at line 89 of file steeringBehavior.cpp.

```
90 {
91    pvector circleCenter = agent.velocity;
92    circleCenter.normalize().mul(CIRCLE_DISTANCE + CIRCLE_RADIUS);
93
94    int wanderAngle = (rand() % 360);
95    pvector displacement {0, 1};
96    setAngle(displacement, wanderAngle);
97    displacement.mul(CIRCLE_RADIUS);
98
99    agent.desiredVelocity = displacement + circleCenter;
100    agent.steering = agent.desiredVelocity - agent.velocity;
101
102    //move it to the center when it is out of screen
103    if(agent.position.x > WIDTH || agent.position.x < -WIDTH ||
104       agent.position.y > HEIGHT || agent.position.y < -HEIGHT)
105       agent.position = point(0,0);
106
107    return agent.steering;
108 }
```

Here is the call graph for this function:

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

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

5.20 wander Class Reference

`#include <wander.h>`

Inheritance diagram for wander:

Collaboration diagram for wander:

Public Member Functions

- wander ()

Static Public Member Functions

- static void loop ()

Additional Inherited Members

5.20.1 Detailed Description

Definition at line 8 of file wander.h.

5.20.2 Constructor & Destructor Documentation

5.20.2.1 wander()

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

Definition at line 17 of file wander.cpp.

```
18 {
19     int agentCount = 30;
20     float maxForce = 0.3;
21     float maxSpeed = 0.6;
22 
23     name = "wandering objects";
24     createAgent(RANDOM, &agentCount, &maxForce, &maxSpeed);
25     callback = reinterpret_cast <void(*)()> ( (void *)(&loop) );
26 }
```

5.20.3 Member Function Documentation

5.20.3.1 loop()

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

Definition at line 8 of file wander.cpp.

```
9 {
10     for(auto it = agents.begin(); it < agents.end(); it++){
11        (*it).force = behavior.wander(*it);
12     }
13 
14     refresh();
15 }
```

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

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

5.21 windy Class Reference

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

Inheritance diagram for windy:

Collaboration diagram for windy:

Public Member Functions

- windy ()

Static Public Member Functions

- static void loop ()

Static Public Attributes

- static flowField flow

Additional Inherited Members

5.21.1 Detailed Description

Definition at line 9 of file windy.h.

5.21.2 Constructor & Destructor Documentation

5.21.2.1 windy()

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

Definition at line 22 of file windy.cpp.

```
23 {
24     int agentCount = 30;
25     float maxForce = 0.3;
26     float maxSpeed = 0.6;
27 
28     name = "flow field";
29     createAgent(RANDOM, &agentCount, &maxForce, &maxSpeed);
30     callback = reinterpret_cast <void(*)()> ( (void *)(&loop) );
31 }
```

5.21.3 Member Function Documentation

5.21.3.1 loop()

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

Definition at line 10 of file windy.cpp.

```
11 {
12     for(auto it = agents.begin(); it < agents.end(); it++){
13          flow = flowField(pvector(GRAVITY));
14          (*it).force  = behavior.inFlowField(*it, flow);
15 
16          flow = flowField(pvector(WIND_WEST));
17          (*it).force += behavior.inFlowField(*it, flow);
18     }
19     refresh();
20 }
```

5.21.4 Member Data Documentation

5.21.4.1 flow

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

Definition at line 13 of file windy.h.

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

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

Chapter 6

File Documentation

6.1 include/agent.h File Reference

agent class defines all agent specifications

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

Include dependency graph for agent.h:

6.2 include/color.h File Reference

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

```
#include <vector>
```

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

Classes

- class color

Enumerations

- enum num {
  BLACK =0, BLUE, GREEN, CYAN,
  RED, MAGENDA, YELLOW, WHITE }

  used to get color from colors vector

6.2.1 Detailed Description

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

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

13.05.2021

6.2.2 Enumeration Type Documentation

6.2.2.1 num

```
enum num
```

used to get color from colors vector

color names for fundamental colors

Enumerator

BLACK	
BLUE	
GREEN	
CYAN	
RED	
MAGENDA	
YELLOW	
WHITE	

Definition at line 18 of file color.h.

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

6.3 include/evade.h File Reference

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

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

Classes

- class evade

6.4 include/flee.h File Reference

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

Classes

- class flee

6.5 include/flock.h File Reference

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

Classes

- class flock

6.6 include/flowField.h File Reference

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

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

Classes

- class flowField

Macros

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

6.6.1 Detailed Description

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

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

13.05.2021

6.6.2 Macro Definition Documentation

6.6.2.1 FIELD_HEIGHT

```
#define FIELD_HEIGHT 34
```

Definition at line 13 of file flowField.h.

6.6.2.2 FIELD_WIDTH

```
#define FIELD_WIDTH 34
```

Definition at line 12 of file flowField.h.

6.6.2.3 GRAVITY

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

Definition at line 16 of file flowField.h.

6.6.2.4 WIND_WEST

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

Definition at line 15 of file flowField.h.

6.7 include/graphics.h File Reference

graphics class, drives openGL

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

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

Classes

- class graphics

Macros

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

6.7.1 Detailed Description

graphics class, drives openGL

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.7.2 Macro Definition Documentation

6.7.2.1 ESC

```
#define ESC 27
```

Definition at line 16 of file graphics.h.

6.7.2.2 HEIGHT

```
#define HEIGHT 34
```

Definition at line 14 of file graphics.h.

6.7.2.3 PI

```
#define PI 3.14159265
```

Definition at line 17 of file graphics.h.

6.7.2.4 WIDTH

```
#define WIDTH 34
```

Definition at line 13 of file graphics.h.

6.8 include/mouseFollower.h File Reference

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

Classes

- class mouseFollower

6.9 include/obstacle.h File Reference

circular obstacles for agent avoidance behaviors

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

Classes

- class obstacle

6.9.1 Detailed Description

circular obstacles for agent avoidance behaviors

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

12.05.2021

6.10 include/obstacleAvoidance.h File Reference

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

Classes

- class obstacleAvoidance

6.11 include/path.h File Reference

path class used for path following steering behaviors.

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

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

Classes

- class path

6.11.1 Detailed Description

path class used for path following steering behaviors.

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

12.05.2021

6.12 include/pathFollower.h File Reference

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

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

Classes

- class pathFollower

6.13 include/point.h File Reference

point class used for point operations

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

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

Classes

- class point

6.13.1 Detailed Description

point class used for point operations

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.14 include/prison.h File Reference

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

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

Classes

- class prison

6.15 include/pursuit.h File Reference

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

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

Classes

- class pursuit

6.16 include/pvector.h File Reference

pvector class used for 2D vector operations

```
#include <string>
```

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

Classes

- class pvector

Macros

- #define PI 3.14159265

6.16.1 Detailed Description

pvector class used for 2D vector operations

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.16.2 Macro Definition Documentation

6.16.2.1 PI

```
#define PI 3.14159265
```

Definition at line 11 of file pvector.h.

6.17 include/random.h File Reference

utility class for random operations

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

Classes

- class random

6.17.1 Detailed Description

utility class for random operations

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.18 include/scenario.h File Reference

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

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

Classes

- class scenario

Enumerations

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

6.18.1 Enumeration Type Documentation

6.18.1.1 types

```
enum types
```

Enumerator

RANDOM	
STATIC	
TROOP	

Definition at line 10 of file scenario.h.

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

6.19 include/steeringBehavior.h File Reference

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

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

Classes

- class steeringBehavior

Macros

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

6.19.1 Macro Definition Documentation

6.19.1.1 AVOID_OBSTACLE

```
#define AVOID_OBSTACLE 4
```

Definition at line 14 of file steeringBehavior.h.

6.19.1.2 CIRCLE_DISTANCE

```
#define CIRCLE_DISTANCE 0.1
```

Definition at line 8 of file steeringBehavior.h.

6.19.1.3 CIRCLE_RADIUS

```
#define CIRCLE_RADIUS 0.4
```

Definition at line 9 of file steeringBehavior.h.

6.19.1.4 EVADE

```
#define EVADE 10
```

Definition at line 20 of file steeringBehavior.h.

6.19.1.5 FLEE

```
#define FLEE 8
```

Definition at line 18 of file steeringBehavior.h.

6.19.1.6 FLOCK

```
#define FLOCK 6
```

Definition at line 16 of file steeringBehavior.h.

6.19.1.7 FOLLOW_MOUSE

```
#define FOLLOW_MOUSE 1
```

Definition at line 11 of file steeringBehavior.h.

6.19.1.8 IN_FLOW_FIELD

```
#define IN_FLOW_FIELD 3
```

Definition at line 13 of file steeringBehavior.h.

6.19.1.9 PURSUIT

```
#define PURSUIT 9
```

Definition at line 19 of file steeringBehavior.h.

6.19.1.10 STAY_IN_FIELD

```
#define STAY_IN_FIELD 2
```

Definition at line 12 of file steeringBehavior.h.

6.19.1.11 STAY_IN_PATH

```
#define STAY_IN_PATH 5
```

Definition at line 15 of file steeringBehavior.h.

6.19.1.12 WANDER

```
#define WANDER 7
```

Definition at line 17 of file steeringBehavior.h.

6.20 include/wander.h File Reference

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

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

Classes

- class wander

6.21 include/windy.h File Reference

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

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

Classes

- class windy

6.22 main.cpp File Reference

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

Include dependency graph for main.cpp:

Functions

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

Variables

- int mode

6.22.1 Function Documentation

6.22.1.1 main()

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

Definition at line 32 of file main.cpp.

```
32                                  {
33     menu();
34
35     scenario* sc;
36
37     if(mode == FOLLOW_MOUSE){
38        *sc = mouseFollower();
39     }
40     else if(mode == STAY_IN_FIELD){
41        *sc = prison();
42     }
43     else if(mode == IN_FLOW_FIELD){
44        *sc = windy();
45     }
```

```
46     else if(mode == WANDER){
47        *sc = wander();
48     }
49     else if(mode == PURSUIT){
50        *sc = pursuit();
51     }
52     else if(mode == FLEE){
53        *sc = flee();
54     }
55     else if(mode == EVADE){
56        *sc = evade();
57     }
58     else if(mode == FLOCK){
59        *sc = flock();
60     }
61     else if(mode == STAY_IN_PATH){
62        *sc = pathFollower();
63     }
64     else if(mode == AVOID_OBSTACLE){
65        *sc = obstacleAvoidance();
66     }
67
68     sc->initGL(&argc, argv);
69
70     return 0;
71 }
```

Here is the call graph for this function:

6.22.1.2 menu()

```
void menu ( )
```

Definition at line 18 of file main.cpp.

```
18            {
19     cout « "Follow Mouse       : 1" « endl;
20     cout « "Stay in Field      : 2" « endl;
21     cout « "In Flow Field      : 3" « endl;
22     cout « "OBSTACLE AVOIDANCE : 4" « endl;
23     cout « "Stay in Path       : 5" « endl;
24     cout « "FLOCK              : 6" « endl;
25     cout « "WANDER             : 7" « endl;
26     cout « "FLEE               : 8" « endl;
27     cout « "PURSUIT            : 9" « endl;
28     cout « "EVADE              : 10" « endl;
29     cin » mode;
30 }
```

Here is the caller graph for this function:

6.22.2 Variable Documentation

6.22.2.1 mode

```
int mode
```

Definition at line 16 of file main.cpp.

6.23 README.md File Reference

6.24 src/agent.cpp File Reference

implementation of the agent class

```
#include "agent.h"
#include "pvector.h"
#include "graphics.h"
#include "random.h"
#include <iostream>
```

Include dependency graph for agent.cpp:

6.24.1 Detailed Description

implementation of the agent class

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

14.05.2021

6.25 src/color.cpp File Reference

color class implementation

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

Include dependency graph for color.cpp:

6.25.1 Detailed Description

color class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

13.05.2021

6.26 src/evade.cpp File Reference

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

Include dependency graph for evade.cpp:

6.27 src/flee.cpp File Reference

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

Include dependency graph for flee.cpp:

6.28 src/flock.cpp File Reference

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

Include dependency graph for flock.cpp:

6.29 src/flowField.cpp File Reference

flowField class implementation

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

Include dependency graph for flowField.cpp:

6.29.1 Detailed Description

flowField class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

13.05.2021

6.30 src/graphics.cpp File Reference

graphics class implementation

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

Include dependency graph for graphics.cpp:

6.30.1 Detailed Description

graphics class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.31 src/mouseFollower.cpp File Reference

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

Include dependency graph for mouseFollower.cpp:

6.32 src/obstacle.cpp File Reference

obstacle class implementation

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

Include dependency graph for obstacle.cpp:

6.32.1 Detailed Description

obstacle class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

12.05.2021

6.33 src/obstacleAvoidance.cpp File Reference

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

6.34 src/path.cpp File Reference

path class implementation

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

6.34.1 Detailed Description

path class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

12.05.2021

6.35 src/pathFollower.cpp File Reference

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

6.36 src/point.cpp File Reference

point class implementation file

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

6.36.1 Detailed Description

point class implementation file

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.37 src/prison.cpp File Reference

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

Include dependency graph for prison.cpp:

Macros

- #define WALL 30
- #define DISTANCE 2

6.37.1 Macro Definition Documentation

6.37.1.1 DISTANCE

```
#define DISTANCE 2
```

Definition at line 7 of file prison.cpp.

6.37.1.2 WALL

```
#define WALL 30
```

Definition at line 6 of file prison.cpp.

6.38 src/pursuit.cpp File Reference

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

Include dependency graph for pursuit.cpp:

6.39 src/pvector.cpp File Reference

pvector class implementation

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

Include dependency graph for pvector.cpp:

6.39.1 Detailed Description

pvector class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.40 src/random.cpp File Reference

utility class for random operations

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

Include dependency graph for random.cpp:

6.40.1 Detailed Description

utility class for random operations

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.41 src/scenario.cpp File Reference

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

Include dependency graph for scenario.cpp:

Macros

- #define MAX_NUMBER_OF_AGENTS 50

6.41.1 Macro Definition Documentation

6.41.1.1 MAX_NUMBER_OF_AGENTS

```
#define MAX_NUMBER_OF_AGENTS 50
```

Definition at line 5 of file scenario.cpp.

6.42 src/steeringBehavior.cpp File Reference

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

Include dependency graph for steeringBehavior.cpp:

6.43 src/wander.cpp File Reference

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

Include dependency graph for wander.cpp:

6.44 src/windy.cpp File Reference

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

Include dependency graph for windy.cpp:

6.45 test/test_suites.cpp File Reference

```
#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)
- BOOST_AUTO_TEST_CASE (s1t2)
- BOOST_AUTO_TEST_CASE (s1t3)
- BOOST_AUTO_TEST_CASE (s1t4)
- BOOST_AUTO_TEST_CASE (s1t5)
- BOOST_AUTO_TEST_CASE (s1t6)
- BOOST_AUTO_TEST_CASE (s1t7)
- BOOST_AUTO_TEST_CASE (s1t8)
- BOOST_AUTO_TEST_CASE (s1t9)
- BOOST_AUTO_TEST_CASE (s2t1)
- BOOST_AUTO_TEST_CASE (s2t2)
- BOOST_AUTO_TEST_CASE (s2t3)

6.45.1 Macro Definition Documentation

6.45.1.1 BOOST_TEST_MODULE

```
#define BOOST_TEST_MODULE test_suites
```

Definition at line 1 of file test_suites.cpp.

6.45.2 Function Documentation

6.45.2.1 BOOST_AUTO_TEST_CASE() [1/12]

```
BOOST_AUTO_TEST_CASE (
              s1t1  )
```

Definition at line 12 of file test_suites.cpp.

```
13   {
14     pvector p1 = pvector(0, 4);
15     pvector p2 = pvector(3, 0);
16     pvector p3 = p1 + p2;
17     BOOST_CHECK(p3.magnitude() == 5);
18   }
```

Here is the call graph for this function:

6.45.2.2 BOOST_AUTO_TEST_CASE() [2/12]

```
BOOST_AUTO_TEST_CASE (
              s1t2  )
```

Definition at line 20 of file test_suites.cpp.

```
21   {
22     pvector p1 = pvector(1, 1);
23     p1.mul(3);
24     pvector p2 = pvector(3, 3);
25     BOOST_CHECK(p1 == p2);
26   }
```

Here is the call graph for this function:

6.45.2.3 BOOST_AUTO_TEST_CASE() [3/12]

```
BOOST_AUTO_TEST_CASE (
              s1t3  )
```

Definition at line 28 of file test_suites.cpp.

```
29   {
30     pvector p1 = pvector(5, 5);
31     p1.div(5);
32     pvector p2 = pvector(1, 1);
33     BOOST_CHECK(p1 == p2);
34   }
```

Here is the call graph for this function:

6.45.2.4 BOOST_AUTO_TEST_CASE() [4/12]

```
BOOST_AUTO_TEST_CASE (
              s1t4  )
```

Definition at line 36 of file test_suites.cpp.

```
37   {
38     pvector p1 = pvector(1, 4);
39     pvector p2 = pvector(3, 2);
40     float dotProduct = p1.dotProduct(p2);
41     BOOST_CHECK(dotProduct == 11);
42   }
```

Here is the call graph for this function:

6.45.2.5 BOOST_AUTO_TEST_CASE() [5/12]

```
BOOST_AUTO_TEST_CASE (
              s1t5  )
```

Definition at line 44 of file test_suites.cpp.

```
45   {
46     pvector p1 = pvector(10, 10);
47     pvector p2 = pvector(0, 10);
48     float angle = p1.angleBetween(p2);
49     BOOST_CHECK(angle == 45);
50   }
```

Here is the call graph for this function:

6.45.2.6 BOOST_AUTO_TEST_CASE() [6/12]

```
BOOST_AUTO_TEST_CASE (
              s1t6  )
```

Definition at line 52 of file test_suites.cpp.

```
53   {
54     pvector p1 = pvector(3, 4);
55     float angle = p1.getAngle();
56     BOOST_CHECK(angle < 53.2 && angle > 52.8);
57   }
```

Here is the call graph for this function:

6.45.2.7 BOOST_AUTO_TEST_CASE() [7/12]

```
BOOST_AUTO_TEST_CASE (
              s1t7  )
```

Definition at line 59 of file test_suites.cpp.

```
60   {
61     pvector p1 = pvector(2, 2);
62     p1.normalize();
63     float range = 0.01;
64     BOOST_CHECK_CLOSE_FRACTION(0.707, p1.x, range);
65     BOOST_CHECK_CLOSE_FRACTION(0.707, p1.y, range);
66   }
```

Here is the call graph for this function:

6.45.2.8 BOOST_AUTO_TEST_CASE() [8/12]

```
BOOST_AUTO_TEST_CASE (
              s1t8  )
```

Definition at line 68 of file test_suites.cpp.

```
69   {
70     pvector p1 = pvector(2, 2);
71     p1.limit(3);
72     float range = 0.01;
73     BOOST_CHECK_CLOSE_FRACTION(2.12, p1.x, range);
74     BOOST_CHECK_CLOSE_FRACTION(2.12, p1.y, range);
75   }
```

Here is the call graph for this function:

6.45.2.9 BOOST_AUTO_TEST_CASE() [9/12]

```
BOOST_AUTO_TEST_CASE (
              s1t9  )
```

Definition at line 77 of file test_suites.cpp.

```
78   {
79     pvector p1 = pvector(1, 1);
80     p1 += pvector(1,1);
81     BOOST_CHECK(p1 == pvector(2,2));
82     p1 = pvector(1,1) + pvector(3,3);
83     BOOST_CHECK(p1 == pvector(4,4));
84     p1 = pvector(4,1) - pvector(3,3);
85     BOOST_CHECK(p1 == pvector(1,-2));
86     p1 = pvector(4,1) - point(3,3);
87     BOOST_CHECK(p1 == pvector(1,-2));
88     p1 = pvector(4,1) + point(3,3);
89     BOOST_CHECK(p1 == pvector(7,4));
90   }
```

Here is the call graph for this function:

6.45.2.10 BOOST_AUTO_TEST_CASE() [10/12]

```
BOOST_AUTO_TEST_CASE (
              s2t1  )
```

Definition at line 96 of file test_suites.cpp.

```
97   {
98     point p1 = point(1, 1);
99     p1.mul(3);
100     point p2 = point(3, 3);
101     BOOST_CHECK(p1 == p2);
102   }
```

Here is the call graph for this function:

6.45.2.11 BOOST_AUTO_TEST_CASE() [11/12]

```
BOOST_AUTO_TEST_CASE (
              s2t2  )
```

Definition at line 104 of file test_suites.cpp.

```
105   {
106     point p1 = point(4, 4);
107     p1.div(4);
108     point p2 = point(1, 1);
109     BOOST_CHECK(p1 == p2);
110   }
```

Here is the call graph for this function:

6.45.2.12 BOOST_AUTO_TEST_CASE() [12/12]

```
BOOST_AUTO_TEST_CASE (
              s2t3  )
```

Definition at line 112 of file test_suites.cpp.

```
113   {
114     point p1 = point(1,1) + point(3,3);
115     BOOST_CHECK(p1 == point(4,4));
116     p1 = point(1,1) + pvector(3,3);
117     BOOST_CHECK(p1 == point(4,4));
118     pvector p2 = point(1,1) - point(3,3);
119     BOOST_CHECK(p2 == pvector(-2,-2));
120   }
```

Here is the call graph for this function:

Index