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 ()
  default constructor.

Static Public Member Functions

- static void loop ()
  loop function for evading scenario

Additional Inherited Members

5.3.1 Detailed Description

Definition at line 15 of file evade.h.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 evade()

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

default constructor.

Creates scenario

Definition at line 31 of file evade.cpp.

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

5.3.3 Member Function Documentation

5.3.3.1 loop()

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

loop function for evading scenario

evading loop function

Note

opengl callback forces that function to be static

Definition at line 15 of file evade.cpp.

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

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

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

5.4 flee Class Reference

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

Inheritance diagram for flee:

Collaboration diagram for flee:

Public Member Functions

- flee ()

  default constructor.

Static Public Member Functions

- static void loop ()

  loop function for evading scenario

Additional Inherited Members

5.4.1 Detailed Description

Definition at line 14 of file flee.h.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 flee()

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

default constructor.

Creates scenario

Definition at line 24 of file flee.cpp.

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

5.4.3 Member Function Documentation

5.4.3.1 loop()

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

loop function for evading scenario

fleeing loop function

Note

opengl callback forces that function to be static

Definition at line 15 of file flee.cpp.

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

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

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

5.5 flock Class Reference

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

Inheritance diagram for flock:

Collaboration diagram for flock:

Public Member Functions

- flock ()
  default constructor.

Static Public Member Functions

- static void loop ()
  loop function for evading scenario

Additional Inherited Members

5.5.1 Detailed Description

Definition at line 15 of file flock.h.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 flock()

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

default constructor.

Creates scenario

Definition at line 36 of file flock.cpp.

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

5.5.3 Member Function Documentation

5.5.3.1 loop()

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

loop function for evading scenario

flocking loop function

Note

opengl callback forces that function to be static

Definition at line 15 of file flock.cpp.

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

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 ()
  default constructor.

Static Public Member Functions

- static void loop ()
  loop function for evading scenario

Additional Inherited Members

5.8.1 Detailed Description

Definition at line 14 of file mouseFollower.h.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 mouseFollower()

`mouseFollower::mouseFollower ( )`

default constructor.

Creates scenario

Definition at line 25 of file mouseFollower.cpp.

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

5.8.3 Member Function Documentation

5.8.3.1 loop()

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

loop function for evading scenario

mouse following loop function

Note

opengl callback forces that function to be static

Definition at line 15 of file mouseFollower.cpp.

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

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

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

5.9 obstacle Class Reference

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

Collaboration diagram for obstacle:

Public Member Functions

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

Public Attributes

- point p
  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 ()
  default constructor.

Static Public Member Functions

- static void loop ()
  loop function for evading scenario
- static void createObstacle (vector< obstacle > &obstacles)
  obstacle creation

Static Public Attributes

- static vector< obstacle > obstacles

  list of obstacles

Additional Inherited Members

5.10.1 Detailed Description

Definition at line 15 of file obstacleAvoidance.h.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 obstacleAvoidance()

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

default constructor.

Creates scenario

Definition at line 43 of file obstacleAvoidance.cpp.

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

5.10.3 Member Function Documentation

5.10.3.1 createObstacle()

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

obstacle creation

Parameters

obstacles	obstacle list to be created

Note

opengl callback forces that function to be static

Definition at line 36 of file obstacleAvoidance.cpp.

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

Here is the call graph for this function:

5.10.3.2 loop()

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

loop function for evading scenario

obstacle avoidance loop function

Note

opengl callback forces that function to be static

Definition at line 17 of file obstacleAvoidance.cpp.

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

Here is the call graph for this function:

5.10.4 Member Data Documentation

5.10.4.1 obstacles

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

list of obstacles

Note

opengl callback forces that function to be static

Definition at line 34 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 ()
  default constructor.

Static Public Member Functions

- static void loop ()
  loop function for evading scenario
- static void createPath (path &p)
  creates path

Static Public Attributes

- static path myPath
  used to access path class behaviors

Additional Inherited Members

5.12.1 Detailed Description

Definition at line 14 of file pathFollower.h.

5.12.2 Constructor & Destructor Documentation

5.12.2.1 pathFollower()

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

default constructor.

Creates scenario

Definition at line 37 of file pathFollower.cpp.

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

5.12.3 Member Function Documentation

5.12.3.1 createPath()

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

creates path

path creating

Parameters

path	to create

Note

opengl callback forces that function to be static

Definition at line 29 of file pathFollower.cpp.

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

Here is the call graph for this function:

5.12.3.2 loop()

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

loop function for evading scenario

path follower avoidance loop function

Note

opengl callback forces that function to be static

Definition at line 17 of file pathFollower.cpp.

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

Here is the call graph for this function:

5.12.4 Member Data Documentation

5.12.4.1 myPath

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

used to access path class behaviors

path instance

Note

opengl callback forces that function to be static

Definition at line 42 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 ()

  default constructor.

Static Public Member Functions

- static void loop ()

  loop function for evading scenario

Additional Inherited Members

5.14.1 Detailed Description

Definition at line 15 of file prison.h.

5.14.2 Constructor & Destructor Documentation

5.14.2.1 prison()

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

default constructor.

Creates scenario

Definition at line 28 of file prison.cpp.

```
29 {
30     int agentCount = 30;
31     float maxForce = 0.6;
32     float maxSpeed = 0.6;
33
34     name = "stay in prison";
35     createAgent(RANDOM, &agentCount, &maxForce, &maxSpeed);
36     callback = reinterpret_cast <void(*)()> ( (void *)(&loop) );
37 }
```

5.14.3 Member Function Documentation

5.14.3.1 loop()

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

loop function for evading scenario

prison loop function

Note

opengl callback forces that function to be static

Definition at line 18 of file prison.cpp.

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

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

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

5.15 pursuit Class Reference

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

Inheritance diagram for pursuit:

Collaboration diagram for pursuit:

Public Member Functions

- pursuit ()
  default constructor.

Static Public Member Functions

- static void loop ()
  loop function for evading scenario

Additional Inherited Members

5.15.1 Detailed Description

Definition at line 14 of file pursuit.h.

5.15.2 Constructor & Destructor Documentation

5.15.2.1 pursuit()

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

default constructor.

Creates scenario

Definition at line 31 of file pursuit.cpp.

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

5.15.3 Member Function Documentation

5.15.3.1 loop()

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

loop function for evading scenario

pursuing loop function

Note

opengl callback forces that function to be static

Definition at line 15 of file pursuit.cpp.

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

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

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

5.16 pvector Class Reference

`#include <pvector.h>`

Collaboration diagram for pvector:

Public Member Functions

- pvector ()
  default constructor
- pvector (float x, float y)
  constructor
- float magnitude ()
  calculates magnitude of the vector
- pvector & normalize ()
  normalize 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 ()

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

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

  initializing of openGL

Static Public Member Functions

- static void refresh ()

  refresh behavior for all items

Public Attributes

- void(∗ callback)()

  openGL screen refresh callback function

Static Public Attributes

- static vector< agent > agents
  all the agents
- static graphics view
  graphics instance
- static steeringBehavior behavior
  behavior instance
- static color myColor
  color instance
- static string name
  name of the scenario

5.18.1 Detailed Description

Definition at line 19 of file scenario.h.

5.18.2 Constructor & Destructor Documentation

5.18.2.1 scenario()

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

default constructor.

Creates scenario

Definition at line 28 of file scenario.cpp.

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

5.18.3 Member Function Documentation

5.18.3.1 createAgent()

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

agent creation

creates agents

Parameters

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

Definition at line 106 of file scenario.cpp.

```
107 {
108     if(type == TROOP){
109         createTroop(*count);
110     }
111     else if(type == RANDOM){
112         createRandomAgents(*count, *force, *speed);
113     }
114     else if(type == STATIC){
115         createStaticAgents();
116     }
117     else{
118         //error message
119     }
120 }
```

5.18.3.2 initGL()

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

initializing of openGL

graphics initialization

Parameters

argv	list of user arguments
argc	number of user arguments

Definition at line 22 of file scenario.cpp.

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

Here is the caller graph for this function:

5.18.3.3 refresh()

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

refresh behavior for all items

applying force, upodating position etc for all items

Note

opengl callback forces that function to be static

Definition at line 35 of file scenario.cpp.

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

Here is the call graph for this function:

5.18.4 Member Data Documentation

5.18.4.1 agents

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

all the agents

existing agents stored in that variable

Note

opengl callback forces that function to be static

Definition at line 57 of file scenario.h.

5.18.4.2 behavior

`steeringBehavior scenario::behavior [static]`

behavior instance

used to apply steering behaviors

Note

opengl callback forces that function to be static

Definition at line 71 of file scenario.h.

5.18.4.3 callback

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

openGL screen refresh callback function

used as main loop in derived classes

Definition at line 91 of file scenario.h.

5.18.4.4 myColor

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

color instance

used to apply color behaviors

Note

opengl callback forces that function to be static

Definition at line 78 of file scenario.h.

5.18.4.5 name

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

name of the scenario

used to display scenario on screen

Note

opengl callback forces that function to be static

Definition at line 85 of file scenario.h.

5.18.4.6 view

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

graphics instance

used to apply graphics operations

Note

opengl callback forces that function to be static

Definition at line 64 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)
  returns force to apply if it is near the specified border
- pvector inFlowField (agent &agent, flowField &flow)
  applies flow field at agents position
- pvector stayInPath (agent &agent, path &path, graphics view)
  agent follows given path
- pvector seek (agent &agent)
  agent goes to specified point
- pvector separation (vector< agent > agents, agent &agent)
  agent stays away from other agents, with specified distance
- pvector cohesion (vector< agent > boids, agent &agent)
  agent goes at the center of other agents positions
- pvector align (vector< agent > boids, agent &agent)
  agent velocity aligned with other agents, with specified distance
- pvector wander (agent &agent)
  agent that will wander
- pvector pursuit (vector< agent > boids, agent &pursuer, graphics view)
  agent pursuits other agent in all agents
- pvector evade (vector< agent > boids, agent &evader, graphics view)
  agent escapes other agent in all agents
- pvector flee (agent &agent, graphics &view, point p)
  agent flees from mouse
- pvector avoid (vector< obstacle > obstacles, agent &agent)
  agent escapes other agent in all agents
- void setAngle (pvector &p, float angle)
  applies angle on vector

5.19.1 Detailed Description

Definition at line 35 of file steeringBehavior.h.

5.19.2 Member Function Documentation

5.19.2.1 align()

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

agent velocity aligned with other agents, with specified distance

align behavior

Parameters

agent	agent to be aligned
boids	list of all the agents

Returns

force to be applied

Definition at line 117 of file steeringBehavior.cpp.

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

Here is the call graph for this function:

5.19.2.2 avoid()

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

agent escapes other agent in all agents

avoidin behavior

Parameters

agent	agent that will avoid from obstacles
obstacles	list of all existing objects

Returns

force to be applied

Definition at line 181 of file steeringBehavior.cpp.

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

Here is the call graph for this function:

5.19.2.3 cohesion()

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

agent goes at the center of other agents positions

cohesion behavior

Parameters

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

Returns

force to be applied

Definition at line 138 of file steeringBehavior.cpp.

```
139 {
140    float neighborDist = 20; //TODO: magic numer
141    point sum {0,0};
142    int count = 0;
```

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

Here is the call graph for this function:

5.19.2.4 evade()

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

agent escapes other agent in all agents

evading behavior

Parameters

evader	agent that will escape
view	used for debugging
boids	list of all the agents

Returns

force to be applied

Definition at line 45 of file steeringBehavior.cpp.

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

Here is the call graph for this function:

5.19.2.5 flee()

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

agent flees from mouse

fleeing behavior

Parameters

agent	agent that will flee
view	used for debugging
p	point that agent flees

Returns

force to be applied

Definition at line 28 of file steeringBehavior.cpp.

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

Here is the call graph for this function:

5.19.2.6 inFlowField()

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

applies flow field at agents position

flow field behavior

Parameters

agent	unit to apply flow field
flow	flow field

Returns

force to be applied

Definition at line 236 of file steeringBehavior.cpp.

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

Here is the call graph for this function:

5.19.2.7 pursuit()

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

agent pursuits other agent in all agents

pursuing behavior

Parameters

pursuer	agent that will follow specified agent
view	used for debugging
boids	list of all the agents

Returns

force to be applied

Definition at line 72 of file steeringBehavior.cpp.

```
73 {
74    agent target;
75    for(auto it = boids.begin(); it < boids.end(); it++){
76       if((*it).name == "gazelle"){
77          target = *it;
78       }
79    }
80
81    point p = point(target.position.x + 2, target.position.y - 2);
82    view.drawText(target.name, p);
83    p = point(pursuer.position.x + 2, pursuer.position.y - 2);
84    view.drawText(pursuer.name, p);
85
86    float dist = (target.position - pursuer.position).magnitude();
87    float t = dist / target.maxSpeed;
88
89    pvector targetVel = target.velocity;
90    targetVel.mul(t);
91    point futurePos = target.position + targetVel;
92    pursuer.targetPoint = futurePos;
93    return seek(pursuer);
94 }
```

Here is the call graph for this function:

5.19.2.8 seek()

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

agent goes to specified point

seek behavior

Parameters

agent	agent that will go to specific target point

Returns

force to be applied

Definition at line 206 of file steeringBehavior.cpp.

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

5.19.2.9 separation()

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

agent stays away from other agents, with specified distance

separation behavior

Parameters

agent	agent to be stayed away
agents	list of all the agents

Returns

force to be applied

Definition at line 158 of file steeringBehavior.cpp.

```
159 {
160     float desiredSeparation = 5; //TODO: magic number
161     pvector sum = pvector(0,0);
162     int count = 0;
163     for(auto it = agents.begin(); it < agents.end(); it++){
164        float d = (agent.position - (*it).position).magnitude();
165        if( (d > 0) && (d < desiredSeparation) ){
166           pvector diff = agent.position - (*it).position;
167           diff.normalize().div(d);
168           sum = sum + diff;
```

```
169           count++;
170        }
171     }
172     if(count > 0){
173        sum.div(count);
174        sum.normalize().mul(agent.maxSpeed);
175        agent.steering = sum - agent.velocity;
176        return agent.steering;
177     }
178     return pvector(0,0);
179 }
```

Here is the call graph for this function:

5.19.2.10 setAngle()

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

applies angle on vector

rotates vector with angle

Parameters

angle	angle that will be set
p	vector that angle will be applied

Definition at line 22 of file steeringBehavior.cpp.

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

5.19.2.11 stayInArea()

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

returns force to apply if it is near the specified border

reflection behavior

Parameters

agent	unit to check
turnpoint	defines border to apply force

Returns

force to be applied

Definition at line 245 of file steeringBehavior.cpp.

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

5.19.2.12 stayInPath()

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

agent follows given path

multi segment path following behavior

Parameters

agent	agent to follow the pathk
path	path to follow
view	used for debugging

Returns

force to be applied

Definition at line 213 of file steeringBehavior.cpp.

```
214 {
215    float worldRecord = 1000000; //TODO: magic number
216    point normalPoint, predictedPos, start, end;
217    pvector distance;
218    for(auto it = path.points.begin(); it < path.points.end()-1; it++){
219       start = point((*it).x, (*it).y);
220       end   = point((*(it+1)).x, (*(it+1)).y);
221       predictedPos = agent.position + agent.velocity;
222       normalPoint.getNormalPoint(predictedPos, start, end);
223       if (normalPoint.x < start.x || normalPoint.x > end.x){
224          normalPoint = end;
225       }
226       distance = predictedPos - normalPoint;
227       if (distance.magnitude() < worldRecord){
228          worldRecord = distance.magnitude();
229          agent.targetPoint = end;
230       }
231       view.drawPoint(agent.targetPoint);
232    }
```

```
233     return seek(agent);
234 }
```

Here is the call graph for this function:

5.19.2.13 wander()

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

agent that will wander

wandering behavior

Parameters

agent	agent to be stayed away

Returns

force to be applied

Definition at line 96 of file steeringBehavior.cpp.

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

Here is the call graph for this function:

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

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

5.20 wander Class Reference

`#include <wander.h>`

Inheritance diagram for wander:

Collaboration diagram for wander:

Public Member Functions

- wander ()

  default constructor.

Static Public Member Functions

- static void loop ()

  loop function for evading scenario

Additional Inherited Members

5.20.1 Detailed Description

Definition at line 14 of file wander.h.

5.20.2 Constructor & Destructor Documentation

5.20.2.1 wander()

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

default constructor.

Creates scenario

Definition at line 24 of file wander.cpp.

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

5.20.3 Member Function Documentation

5.20.3.1 loop()

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

loop function for evading scenario

wandering loop function

Note

opengl callback forces that function to be static

Definition at line 15 of file wander.cpp.

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

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

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

5.21 windy Class Reference

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

Inheritance diagram for windy:

Collaboration diagram for windy:

Public Member Functions

- windy ()
  default constructor.

Static Public Member Functions

- static void loop ()
  loop function for evading scenario

Static Public Attributes

- static flowField flow
  flow field instance

Additional Inherited Members

5.21.1 Detailed Description

Definition at line 15 of file windy.h.

5.21.2 Constructor & Destructor Documentation

5.21.2.1 windy()

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

default constructor.

Creates scenario

Definition at line 29 of file windy.cpp.

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

5.21.3 Member Function Documentation

5.21.3.1 loop()

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

loop function for evading scenario

windy loop function

Note

opengl callback forces that function to be static

Definition at line 17 of file windy.cpp.

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

5.21.4 Member Data Documentation

5.21.4.1 flow

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

flow field instance

Note

opengl callback forces that function to be static

Definition at line 34 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

evade class inherited from scenario class

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

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

Classes

- class evade

6.3.1 Detailed Description

evade class inherited from scenario class

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.4 include/flee.h File Reference

agents flee from mouse scenario

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

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

Classes

- class flee

6.4.1 Detailed Description

agents flee from mouse scenario

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.5 include/flock.h File Reference

flocking agents scenario

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

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

Classes

- class flock

6.5.1 Detailed Description

flocking agents scenario

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.6 include/flowField.h File Reference

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

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

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

Classes

- class flowField

Macros

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

6.6.1 Detailed Description

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

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

13.05.2021

6.6.2 Macro Definition Documentation

6.6.2.1 FIELD_HEIGHT

```
#define FIELD_HEIGHT 34
```

Definition at line 13 of file flowField.h.

6.6.2.2 FIELD_WIDTH

```
#define FIELD_WIDTH 34
```

Definition at line 12 of file flowField.h.

6.6.2.3 GRAVITY

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

Definition at line 16 of file flowField.h.

6.6.2.4 WIND_WEST

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

Definition at line 15 of file flowField.h.

6.7 include/graphics.h File Reference

graphics class, drives openGL

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

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

Classes

- class graphics

Macros

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

6.7.1 Detailed Description

graphics class, drives openGL

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.7.2 Macro Definition Documentation

6.7.2.1 ESC

```
#define ESC 27
```

Definition at line 16 of file graphics.h.

6.7.2.2 HEIGHT

```
#define HEIGHT 34
```

Definition at line 14 of file graphics.h.

6.7.2.3 PI

```
#define PI 3.14159265
```

Definition at line 17 of file graphics.h.

6.7.2.4 WIDTH

```
#define WIDTH 34
```

Definition at line 13 of file graphics.h.

6.8 include/mouseFollower.h File Reference

agents follow mouse scenario

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

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

Classes

- class mouseFollower

6.8.1 Detailed Description

agents follow mouse scenario

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.9 include/obstacle.h File Reference

circular obstacles for agent avoidance behaviors

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

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

Classes

- class obstacle

6.9.1 Detailed Description

circular obstacles for agent avoidance behaviors

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

12.05.2021

6.10 include/obstacleAvoidance.h File Reference

agents avoid from obstacles scenario

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

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

Classes

- class obstacleAvoidance

6.10.1 Detailed Description

agents avoid from obstacles scenario

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.11 include/path.h File Reference

path class used for path following steering behaviors.

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

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

Classes

- class path

6.11.1 Detailed Description

path class used for path following steering behaviors.

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

12.05.2021

6.12 include/pathFollower.h File Reference

path following scenario

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

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

Classes

- class pathFollower

6.12.1 Detailed Description

path following scenario

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.13 include/point.h File Reference

point class used for point operations

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

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

Classes

- class point

6.13.1 Detailed Description

point class used for point operations

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.14 include/prison.h File Reference

agents cant escape from field scenario

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

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

Classes

- class prison

6.14.1 Detailed Description

agents cant escape from field scenario

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.15 include/pursuit.h File Reference

one agent pursue other one scenario

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

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

Classes

- class pursuit

6.15.1 Detailed Description

one agent pursue other one scenario

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.16 include/pvector.h File Reference

pvector class used for 2D vector operations

```
#include <string>
```

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

Classes

- class pvector

Macros

- #define PI 3.14159265

6.16.1 Detailed Description

pvector class used for 2D vector operations

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.16.2 Macro Definition Documentation

6.16.2.1 PI

```
#define PI 3.14159265
```

Definition at line 11 of file pvector.h.

6.17 include/random.h File Reference

utility class for random operations

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

Classes

- class random

6.17.1 Detailed Description

utility class for random operations

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.18 include/scenario.h File Reference

base class for all scenarios

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

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

Classes

- class scenario

Enumerations

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

6.18.1 Detailed Description

base class for all scenarios

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.18.2 Enumeration Type Documentation

6.18.2.1 types

```
enum types
```

Enumerator

RANDOM	
STATIC	
TROOP	

Definition at line 17 of file scenario.h.

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

6.19 include/steeringBehavior.h File Reference

functions for autonomous steering behaviors

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

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

Classes

- class steeringBehavior

Macros

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

6.19.1 Detailed Description

functions for autonomous steering behaviors

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.19.2 Macro Definition Documentation

6.19.2.1 AVOID_OBSTACLE

```
#define AVOID_OBSTACLE 4
```

Definition at line 21 of file steeringBehavior.h.

6.19.2.2 CIRCLE_DISTANCE

```
#define CIRCLE_DISTANCE 0.1
```

Definition at line 15 of file steeringBehavior.h.

6.19.2.3 CIRCLE_RADIUS

```
#define CIRCLE_RADIUS 0.4
```

Definition at line 16 of file steeringBehavior.h.

6.19.2.4 EVADE

```
#define EVADE 10
```

Definition at line 27 of file steeringBehavior.h.

6.19.2.5 FLEE

```
#define FLEE 8
```

Definition at line 25 of file steeringBehavior.h.

6.19.2.6 FLOCK

```
#define FLOCK 6
```

Definition at line 23 of file steeringBehavior.h.

6.19.2.7 FOLLOW_MOUSE

```
#define FOLLOW_MOUSE 1
```

Definition at line 18 of file steeringBehavior.h.

6.19.2.8 IN_FLOW_FIELD

```
#define IN_FLOW_FIELD 3
```

Definition at line 20 of file steeringBehavior.h.

6.19.2.9 PURSUIT

```
#define PURSUIT 9
```

Definition at line 26 of file steeringBehavior.h.

6.19.2.10 STAY_IN_FIELD

```
#define STAY_IN_FIELD 2
```

Definition at line 19 of file steeringBehavior.h.

6.19.2.11 STAY_IN_PATH

```
#define STAY_IN_PATH 5
```

Definition at line 22 of file steeringBehavior.h.

6.19.2.12 WANDER

```
#define WANDER 7
```

Definition at line 24 of file steeringBehavior.h.

6.20 include/wander.h File Reference

random wandering agents scenario

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

Classes

- class wander

6.20.1 Detailed Description

random wandering agents scenario

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.21 include/windy.h File Reference

windy air scenario

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

Classes

- class windy

6.21.1 Detailed Description

windy air scenario

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.22 main.cpp File Reference

client code

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

Include dependency graph for main.cpp:

Functions

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

Variables

- int mode
  specifies user selected scenario

6.22.1 Detailed Description

client code

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.22.2 Function Documentation

6.22.2.1 main()

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

main routine

Definition at line 48 of file main.cpp.

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

Here is the call graph for this function:

6.22.2.2 menu()

```
void menu ( )
```

displays menu

Definition at line 31 of file main.cpp.

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

Here is the caller graph for this function:

6.22.3 Variable Documentation

6.22.3.1 mode

```
int mode
```

specifies user selected scenario

Definition at line 26 of file main.cpp.

6.23 README.md File Reference

6.24 src/agent.cpp File Reference

implementation of the agent class

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

Include dependency graph for agent.cpp:

6.24.1 Detailed Description

implementation of the agent class

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

14.05.2021

6.25 src/color.cpp File Reference

color class implementation

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

Include dependency graph for color.cpp:

6.25.1 Detailed Description

color class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

13.05.2021

6.26 src/evade.cpp File Reference

evade class implementation

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

Include dependency graph for evade.cpp:

6.26.1 Detailed Description

evade class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.27 src/flee.cpp File Reference

flee class implementation

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

Include dependency graph for flee.cpp:

6.27.1 Detailed Description

flee class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.28 src/flock.cpp File Reference

flock class implementation

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

Include dependency graph for flock.cpp:

6.28.1 Detailed Description

flock class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.29 src/flowField.cpp File Reference

flowField class implementation

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

Include dependency graph for flowField.cpp:

6.29.1 Detailed Description

flowField class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

13.05.2021

6.30 src/graphics.cpp File Reference

graphics class implementation

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

Include dependency graph for graphics.cpp:

6.30.1 Detailed Description

graphics class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.31 src/mouseFollower.cpp File Reference

mouseFollower class implementation

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

Include dependency graph for mouseFollower.cpp:

6.31.1 Detailed Description

mouseFollower class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.32 src/obstacle.cpp File Reference

obstacle class implementation

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

Include dependency graph for obstacle.cpp:

6.32.1 Detailed Description

obstacle class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

12.05.2021

6.33 src/obstacleAvoidance.cpp File Reference

obstacleAvoidance class implementation

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

Include dependency graph for obstacleAvoidance.cpp:

6.33.1 Detailed Description

obstacleAvoidance class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.34 src/path.cpp File Reference

path class implementation

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

Include dependency graph for path.cpp:

6.34.1 Detailed Description

path class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

12.05.2021

6.35 src/pathFollower.cpp File Reference

pathFollower class implementation

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

Include dependency graph for pathFollower.cpp:

6.35.1 Detailed Description

pathFollower class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.36 src/point.cpp File Reference

point class implementation file

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

Include dependency graph for point.cpp:

6.36.1 Detailed Description

point class implementation file

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.37 src/prison.cpp File Reference

prison class implementation

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

Include dependency graph for prison.cpp:

Macros

- #define WALL 30
- #define DISTANCE 2

6.37.1 Detailed Description

prison class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.37.2 Macro Definition Documentation

6.37.2.1 DISTANCE

```
#define DISTANCE 2
```

Definition at line 14 of file prison.cpp.

6.37.2.2 WALL

```
#define WALL 30
```

Definition at line 13 of file prison.cpp.

6.38 src/pursuit.cpp File Reference

prison class implementation

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

Include dependency graph for pursuit.cpp:

6.38.1 Detailed Description

prison class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.39 src/pvector.cpp File Reference

pvector class implementation

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

Include dependency graph for pvector.cpp:

6.39.1 Detailed Description

pvector class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.40 src/random.cpp File Reference

utility class for random operations

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

Include dependency graph for random.cpp:

6.40.1 Detailed Description

utility class for random operations

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.41 src/scenario.cpp File Reference

scenario base class implementation

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

Include dependency graph for scenario.cpp:

Macros

- #define MAX_NUMBER_OF_AGENTS 50

6.41.1 Detailed Description

scenario base class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.41.2 Macro Definition Documentation

6.41.2.1 MAX_NUMBER_OF_AGENTS

```
#define MAX_NUMBER_OF_AGENTS 50
```

Definition at line 12 of file scenario.cpp.

6.42 src/steeringBehavior.cpp File Reference

implementation of autonomous steering behaviors

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

Include dependency graph for steeringBehavior.cpp:

6.42.1 Detailed Description

implementation of autonomous steering behaviors

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.43 src/wander.cpp File Reference

wander class implementation

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

Include dependency graph for wander.cpp:

6.43.1 Detailed Description

wander class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.44 src/windy.cpp File Reference

windy class implementation

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

Include dependency graph for windy.cpp:

6.44.1 Detailed Description

windy class implementation

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.45 test/test_suites.cpp File Reference

unit test suites

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

Include dependency graph for test_suites.cpp:

Macros

- #define BOOST_TEST_MODULE test_suites

Functions

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

6.45.1 Detailed Description

unit test suites

Author

Mehmet Rıza Öz - mehmetrizaoz@gmail.com

Date

15.05.2021

6.45.2 Macro Definition Documentation

6.45.2.1 BOOST_TEST_MODULE

```
#define BOOST_TEST_MODULE test_suites
```

Definition at line 8 of file test_suites.cpp.

6.45.3 Function Documentation

6.45.3.1 BOOST_AUTO_TEST_CASE() [1/12]

```
BOOST_AUTO_TEST_CASE (
              s1t1  )
```

pvector magnitude test case

Definition at line 22 of file test_suites.cpp.

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

Here is the call graph for this function:

6.45.3.2 BOOST_AUTO_TEST_CASE() [2/12]

```
BOOST_AUTO_TEST_CASE (
              s1t2  )
```

pvector mul test case

Definition at line 33 of file test_suites.cpp.

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

Here is the call graph for this function:

6.45.3.3 BOOST_AUTO_TEST_CASE() [3/12]

```
BOOST_AUTO_TEST_CASE (
              s1t3  )
```

pvector div test case

Definition at line 44 of file test_suites.cpp.

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

Here is the call graph for this function:

6.45.3.4 BOOST_AUTO_TEST_CASE() [4/12]

```
BOOST_AUTO_TEST_CASE (
              s1t4  )
```

pvector dotproduct test case

Definition at line 55 of file test_suites.cpp.

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

Here is the call graph for this function:

6.45.3.5 BOOST_AUTO_TEST_CASE() [5/12]

```
BOOST_AUTO_TEST_CASE (
              s1t5  )
```

pvector angle between vectors test case

Definition at line 66 of file test_suites.cpp.

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

Here is the call graph for this function:

6.45.3.6 BOOST_AUTO_TEST_CASE() [6/12]

```
BOOST_AUTO_TEST_CASE (
              s1t6  )
```

pvector get vector angle test case

Definition at line 77 of file test_suites.cpp.

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

Here is the call graph for this function:

6.45.3.7 BOOST_AUTO_TEST_CASE() [7/12]

```
BOOST_AUTO_TEST_CASE (
              s1t7  )
```

pvector normalize test case

Definition at line 87 of file test_suites.cpp.

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

Here is the call graph for this function:

6.45.3.8 BOOST_AUTO_TEST_CASE() [8/12]

```
BOOST_AUTO_TEST_CASE (
              s1t8  )
```

pvector limit test case

Definition at line 99 of file test_suites.cpp.

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

Here is the call graph for this function:

6.45.3.9 BOOST_AUTO_TEST_CASE() [9/12]

```
BOOST_AUTO_TEST_CASE (
              s1t9  )
```

pvector overloaded operators test case

Definition at line 111 of file test_suites.cpp.

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

Here is the call graph for this function:

6.45.3.10 BOOST_AUTO_TEST_CASE() [10/12]

```
BOOST_AUTO_TEST_CASE (
              s2t1  )
```

point multiplication test case

Definition at line 133 of file test_suites.cpp.

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

Here is the call graph for this function:

6.45.3.11 BOOST_AUTO_TEST_CASE() [11/12]

```
BOOST_AUTO_TEST_CASE (
              s2t2  )
```

point division test case

Definition at line 144 of file test_suites.cpp.

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

Here is the call graph for this function:

6.45.3.12 BOOST_AUTO_TEST_CASE() [12/12]

```
BOOST_AUTO_TEST_CASE (
              s2t3  )
```

point overloaded operators test case

Definition at line 155 of file test_suites.cpp.

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

Here is the call graph for this function:

Index