

Fisher sim

0.0.1

Technical Documentation



Software Engineering Project
Group 12

Generated by Doxygen 1.8.9.1

Sun Mar 29 2015 22:19:34

Contents

1	Algorithms & Data Structures	3
2	Namespace Index	5
2.1	Namespace List	5
3	Hierarchical Index	7
3.1	Class Hierarchy	7
4	Class Index	9
4.1	Class List	9
5	File Index	11
5.1	File List	11
6	Namespace Documentation	13
6.1	Ui Namespace Reference	13
7	Class Documentation	15
7.1	Agent Class Reference	15
7.1.1	Detailed Description	17
7.1.2	Constructor & Destructor Documentation	17
7.1.2.1	Agent	17
7.1.3	Member Function Documentation	17
7.1.3.1	calcThreshold	17
7.1.3.2	getCommunication	17
7.1.3.3	getDecision	18
7.1.3.4	getEarlyDecision	18
7.1.3.5	getFishDuration	18
7.1.3.6	getHistory	19
7.1.3.7	getSkill	19
7.1.3.8	getStrat	19

7.1.3.9	getTemp	19
7.1.3.10	getThreshold	20
7.1.3.11	makeDecision	20
7.1.3.12	makeEarlyDecision	20
7.1.3.13	setCommunication	21
7.1.3.14	setFishduration	21
7.1.3.15	setSkill	21
7.1.3.16	setTemp	22
7.1.3.17	updateHistory	22
7.1.3.18	updateStrategyScore	23
7.1.4	Member Data Documentation	23
7.1.4.1	communication	23
7.1.4.2	decision	23
7.1.4.3	earlydecison	23
7.1.4.4	fishduration	23
7.1.4.5	history	23
7.1.4.6	skill	24
7.1.4.7	strat	24
7.1.4.8	temp	24
7.1.4.9	threshold	24
7.2	Graphview Class Reference	24
7.2.1	Detailed Description	25
7.2.2	Constructor & Destructor Documentation	25
7.2.2.1	Graphview	25
7.2.2.2	~Graphview	26
7.2.3	Member Function Documentation	26
7.2.3.1	on_actionInsert_Plot_triggered	26
7.2.3.2	on_actionSave_Document_triggered	26
7.2.3.3	setupPlot	26
7.2.4	Member Data Documentation	26
7.2.4.1	ui	26
7.3	MainWindow Class Reference	27
7.3.1	Detailed Description	28
7.3.2	Constructor & Destructor Documentation	28
7.3.2.1	MainWindow	28
7.3.2.2	~MainWindow	28
7.3.3	Member Function Documentation	28

7.3.3.1	log	29
7.3.3.2	on_fishers_valueChanged	30
7.3.3.3	on_fishpop_valueChanged	30
7.3.3.4	on_fishtypes_valueChanged	30
7.3.3.5	on_lineEdit_0_textEdited	30
7.3.3.6	on_lineEdit_1_textEdited	31
7.3.3.7	on_lineEdit_2_textEdited	31
7.3.3.8	on_lineEdit_3_textEdited	31
7.3.3.9	on_lineEdit_4_textEdited	31
7.3.3.10	on_locations_valueChanged	31
7.3.3.11	on_reportButton_clicked	31
7.3.3.12	on_runtime_valueChanged	31
7.3.3.13	on_simulateButton_clicked	31
7.3.3.14	on_weather_clicked	32
7.3.4	Member Data Documentation	32
7.3.4.1	graphview	32
7.3.4.2	ui	32
7.4	Spot Class Reference	32
7.4.1	Detailed Description	33
7.4.2	Constructor & Destructor Documentation	33
7.4.2.1	Spot	33
7.4.3	Member Function Documentation	33
7.4.3.1	crowdness	33
7.4.3.2	getAgentNum	34
7.4.3.3	getSpotCapacity	34
7.4.3.4	setAgentNum	35
7.4.3.5	setCap	35
7.4.4	Member Data Documentation	35
7.4.4.1	maxcapacity	35
7.4.4.2	numAgent	36
7.5	Strategy Class Reference	36
7.5.1	Detailed Description	37
7.5.2	Constructor & Destructor Documentation	37
7.5.2.1	Strategy	37
7.5.3	Member Function Documentation	37
7.5.3.1	getDecisionPattern	37
7.5.3.2	getScore	37

7.5.3.3	updateScore	38
7.5.4	Member Data Documentation	38
7.5.4.1	decisionPattern	38
7.5.4.2	score	38
7.6	UserSettings Class Reference	38
7.6.1	Detailed Description	39
7.6.2	Constructor & Destructor Documentation	39
7.6.2.1	UserSettings	39
7.6.3	Member Function Documentation	39
7.6.3.1	getfisherNum	39
7.6.3.2	getfishLoc	40
7.6.3.3	getfishPop	40
7.6.3.4	getfishTemp	40
7.6.3.5	getfishType	41
7.6.3.6	getRuntime	41
7.6.4	Member Data Documentation	41
7.6.4.1	fisherNum	41
7.6.4.2	fishLoc	42
7.6.4.3	fishPop	42
7.6.4.4	fishTemp	42
7.6.4.5	fishType	42
7.6.4.6	runtime	42
8	File Documentation	43
8.1	agent.cpp File Reference	43
8.1.1	Function Documentation	44
8.1.1.1	initAgent	44
8.1.2	Variable Documentation	44
8.1.2.1	all_agent	44
8.2	agent.h File Reference	44
8.2.1	Detailed Description	45
8.2.2	Function Documentation	46
8.2.2.1	initAgent	46
8.2.3	Variable Documentation	46
8.2.3.1	numOfAgent	46
8.3	Algorithms.dox File Reference	46
8.4	graphview.cpp File Reference	46

8.5	graphview.h File Reference	47
8.5.1	Detailed Description	48
8.6	main.cpp File Reference	48
8.6.1	Function Documentation	49
8.6.1.1	main	49
8.7	mainpage.dox File Reference	49
8.8	mainwindow.cpp File Reference	49
8.9	mainwindow.h File Reference	50
8.9.1	Detailed Description	51
8.10	randomgenerator.cpp File Reference	51
8.10.1	Function Documentation	52
8.10.1.1	generateRandomNumber	52
8.11	randomgenerator.h File Reference	53
8.11.1	Detailed Description	54
8.11.2	Function Documentation	54
8.11.2.1	generateRandomNumber	54
8.12	spot.cpp File Reference	54
8.13	spot.h File Reference	55
8.13.1	Detailed Description	56
8.14	spottest.cpp File Reference	56
8.14.1	Function Documentation	57
8.14.1.1	main	58
8.15	strategy.cpp File Reference	59
8.15.1	Function Documentation	59
8.15.1.1	initStrategy	59
8.16	strategy.h File Reference	59
8.16.1	Detailed Description	60
8.16.2	Function Documentation	61
8.16.2.1	initStrategy	61
8.17	unit_spot_test.cpp File Reference	61
8.17.1	Function Documentation	61
8.17.1.1	main	62
8.18	unit_test_agent.cpp File Reference	62
8.18.1	Function Documentation	63
8.18.1.1	main	64
8.19	unit_test_strategy.cpp File Reference	64
8.19.1	Function Documentation	65

8.19.1.1	main	66
8.20	UserSettings.cpp File Reference	67
8.21	UserSettings.h File Reference	67
8.21.1	Detailed Description	68
Index		69

Introduction

Fisher Sim is being developed as part of a Software Engineering project at Rutgers University for the spring semester of 2015.

Group 12

Team members:

- Matthew Chatten
- Ameer Fiqri Barahim
- Vicent Vindel Dura
- Alexander Hill
- David Lazaar
- Orielle Joy Yu

Project Goals

The Fisher Sim project seeks to build off of the classic El Farol Bar problem in game theory. In the El Farol Bar problem models for decisions that a based on others are examined. In the original formulation, the question is whether or not to go to a bar. Going to the bar is a good decision only if most people decide it is a bad decision, and vice versa.

Fisher Sim adds additional metrics to this problem in an attempt to better understand and predict people's disision to go fishing.

Compiling the software

Fisher sim currently consists of two separate programs. The primary component is located under the CrowdAnalysys folder in in the project root directory. This folder contains the main project as a QT application along with the technical documentation (this file). The other components of the Fisher sim program are located under the /spot and /Agent folders. These folders contain work on the simulation engine and contain basic console c++ applications. They are currently separated from the primary GUI application in order to simplify debugging.

To build the primary application you will need a working installation of the QT creator framework. The community edition obtained for free from their website located here: <https://www.qt.io/download/> In addition to QT creator, you will need a c++ compiler for your system. If you do not already have a compiler installed and are on a Windows system then a suitable compiler can be obtained by installing a version of Microsoft's visual studio express. On Debian Linux systems, a c++ compiler can be installed by installing the buildutils package from your package manager.

Updating Documentation

Technical documentation is maintained through the Doxygen tool by loading the Doxyfile located under /Crowd←Analysys/docs. Using Doxygen allows for the documentation to be included along with the code which can assist in keeping things up to date. When changes to the code / documentation are made the Doxygen tool must be run to rebuild the Technical Documentation. This will create an additional 2 folders in the docs folder each one containing an html edition and the other containing a Latex / pdf version.

If you wish to build the pdf version you will need an installation of latex on your system and to have its binaries in your system path. Linux editions of latex can be installed through the package manager and a windows edition can be obtained from the Miktex project located at <http://miktex.org/>. In order to generate class relation images your system will need GraphViz installed.

Tools needed summery

Software Build

- MSVS or GNU Build system
- Qt Creator

Documentation Build

- Doxygen
- Latex
- GraphViz

Adding Documentation

Documentation can be added in two general styles. Most documentation will mostly be general explanations for programming constructs which can be added as explained <http://www.stack.nl/~dimitri/doxygen/manual/docblocks.↵html>.

More extensive comments can take advantage of Markdown formatting and Latex style mathematical expressions. Supported markdown formatting can be seen here: <http://www.stack.nl/~dimitri/doxygen/manual/markdown.↵html>.

Chapter 1

Algorithms & Data Structures

Algorithms

Decision Making

The algorithm is made to compute a unique decision for every agent. The decision is either to go fishing (denoted as 1) or stay at home (denoted as -1). At first every decision of an agent is randomly chosen from a random strategy. Then, every decision may change by the percentage of influence threshold, p . The decision is determined using the logic below:

```
if p < 70
    decision that is made by the strategy is kept.
else if p > 70
    decision will be change to 1-go to fishing.
```

The value of influence threshold depends on the factors below:

- Skill and experience rank
- Frequency of communication
- Amount of each type of fish
- Fishing duration
- Weather pattern

Since some of the factors above are unique for each agents, it will be able to preserve the uniqueness of every decision. Every factors will contribute 20% to the influence threshold.

Strategy

Every agent will have a short-term memory and a long-term memory. Short-term memory is limited to 3 previous outcomes of the agent winning and losing. Long-term memory is the strategy that is used by the agent to make the initial decision before taking into account of influence threshold.

Since there are 8 possible outcomes from the short-term memory, the strategy that can be generated from these outcome is 256. Every agent is allow to have 3 strategies, this will result in 2,763,520 different combinations of strategies. Every agent will get a random combination of 3 strategies and it will be likely that every combination is unique.

The process to make the early decision is shown below: strategy=choose the strategy that has a higher score

```

for i=1 →8
    if history==strategy history [i]
        early decision=strategy action [i]
    return early decision

```

At the beginning of every simulation, all the strategies' score are zero. So, it can be conclude that the initial strategy of every agent is random. If the agent won the round the strategy score will increase by one point. Conversely, every losing round the strategy score is lowered by one. The early decision will be passed to the decision making where the influence threshold of the agent will be calculated and the early decision may be changed.

Overall process

Below is the overall process of how every decision of an agent being made:

```

strategy=choose the strategy that has a higher score
for i=1 →8
    if history==strategy history [i]
        early decision=strategy action [i]
if p<70
    decision=early decision
else if p>70
    decision=1

```

Strategy score will be calculated when all the decisions have been made. Plus for a strategy to earn the score the decision must not be changed by the influence threshold. The logic is shown below:

```

if p<70
    if majority go to fishing and decision == -1
        strategy score increase by one point
    else strategy score lower by one point
    if majority stay at home and decision==1
        strategy score increase by one point
    else strategy score lower by one point

```

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Ui	13
----------	----

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Agent	15
QMainWindow	
Graphview	24
MainWindow	27
Spot	32
Strategy	36
UserSettings	38
MainWindow	27

Chapter 4

Class Index

4.1 Class List

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

Agent	Agent represents 1 agent's experience and decisions	15
Graphview	View that shows the collected graphs and allows them to be inserted into a report	24
MainWindow	The MainWindow class Provides the Main windows for the Fisher sim project	27
Spot	Spot is used to create a location and calculate how crowded it is	32
Strategy	Strategy for determining the conditions of going fishing. since each startegy depends on 3 previous outcomes, so posiible output for one strategy is 8. the sequence for the 3 previous outcomes would be: 000,001,010,...,111 special case for starategy: 0->stay at home, 1->go fishing	36
UserSettings	Users global simulation parameters	38

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

agent.cpp	43
agent.h	
Agent represents 1 agent's experience and decisions	44
graphview.cpp	46
graphview.h	
View that shows the collected graphs and allows them to be inserted into a report	47
main.cpp	48
mainwindow.cpp	49
mainwindow.h	
Mainwindow creates the primary GUI display	50
randomgenerator.cpp	51
randomgenerator.h	
Construct a trivial random generator engine from a time-based seed:	53
spot.cpp	54
spot.h	
Used to create a spot and calculate how crowded a spot is	55
spottest.cpp	56
strategy.cpp	59
strategy.h	
Strategy for determining the conditions of going fishing	59
unit_spot_test.cpp	61
unit_test_agent.cpp	62
unit_test_strategy.cpp	64
UserSettings.cpp	67
UserSettings.h	
Users global simulation parameters	67

Chapter 6

Namespace Documentation

6.1 Ui Namespace Reference

Chapter 7

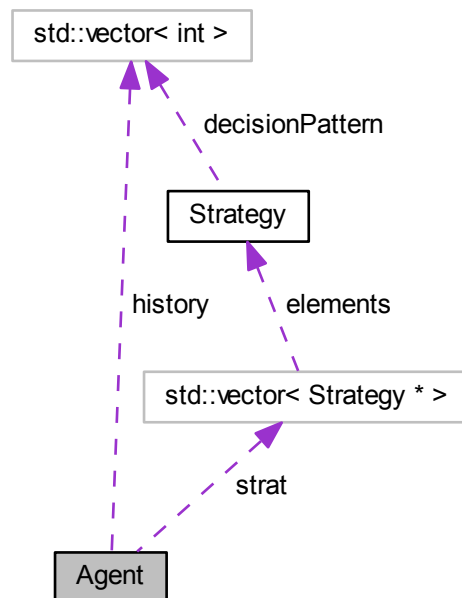
Class Documentation

7.1 Agent Class Reference

agent represents 1 agent's experience and decisions

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

Collaboration diagram for Agent:



Public Member Functions

- [Agent](#) (vector< [Strategy](#) * > strat)
default constructor
- void [updateStrategyScore](#) (int winnigScore)
- void [calcThreshold](#) ()
- void [makeEarlyDecision](#) ()
- void [makeDecision](#) ()
will be based on earlydecision and threshold
- void [updateHistory](#) ()
push new decision on
- void [setTemp](#) (float newTemp)
- void [setSkill](#) (int newskill)
can be randomize
- void [setFishduration](#) (float newFishDuration)
can be randomize
- void [setCommunication](#) (int newCommunication)
- vector< int > [getHistory](#) ()
- int [getDecision](#) ()
- int [getCommunication](#) ()
Returns the amount the agent communicates with other agents.
- int [getSkill](#) ()
Returns the current skill of the agent.
- float [getTemp](#) ()
- float [getFishDuration](#) ()
- int [getEarlyDecision](#) ()
- float [getThreshold](#) ()
- vector< [Strategy](#) * > [getStrat](#) ()

Private Attributes

- vector< [Strategy](#) * > [strat](#)
holds the 3 unique strategies
- vector< int > [history](#)
holds the last 3 outcomes
- int [decision](#)
*new decision *have to calculate, use makeDecision*
- int [earlydecison](#)
based on history sequence and strategy
- double [skill](#)
*will be 1-5 *from input*
- float [fishduration](#)
**from input*
- float [temp](#)
**from input*
- int [communication](#)
**from input*
- float [threshold](#)

7.1.1 Detailed Description

agent represents 1 agent's experience and decisions

records the total number of agents created.

influence threshold,

```
if based on report > 70
    will make agent's decision change to 1
if < 70
    agent's decision remain the same

new rule: p => 85 change decision to 1
         40 < p < 85 decision remain
         p <= 40 decision change to -1
```

7.1.2 Constructor & Destructor Documentation

7.1.2.1 Agent::Agent (vector< Strategy * > strat)

default constructor

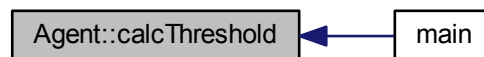
Here is the call graph for this function:



7.1.3 Member Function Documentation

7.1.3.1 void Agent::calcThreshold ()

Here is the caller graph for this function:



7.1.3.2 int Agent::getCommunication ()

Returns the amount the agent communicates with other agents.

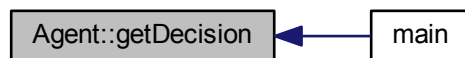
7.1.3.3 int Agent::getDecision ()

Returns the Decision of the [Agent](#)

Returns

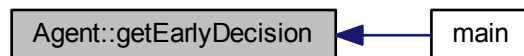
the decision of the [Agent](#)

Here is the caller graph for this function:



7.1.3.4 int Agent::getEarlyDecision ()

Here is the caller graph for this function:



7.1.3.5 float Agent::getFishDuration ()

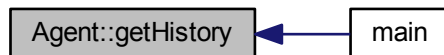
Returns the decay of the fish population

Returns

the decay value used

7.1.3.6 `vector< int > Agent::getHistory ()`

Here is the caller graph for this function:

**7.1.3.7** `int Agent::getSkill ()`

Returns the current skill of the agent.

7.1.3.8 `vector< Strategy * > Agent::getStrat ()`

Returns the statagies used by this agent Agents can change stratagies as they learn

Returns

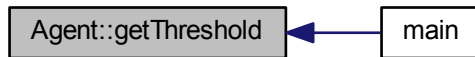
a vector containing the stratagies. This will normally return 3 items.

Here is the caller graph for this function:

**7.1.3.9** `float Agent::getTemp ()`

7.1.3.10 float Agent::getThreshold ()

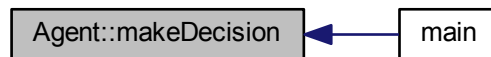
Here is the caller graph for this function:



7.1.3.11 void Agent::makeDecision ()

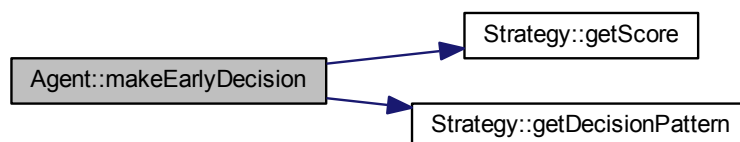
will be based on earlydecision and threshold

Here is the caller graph for this function:

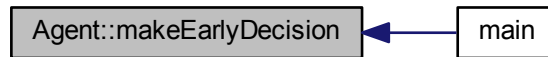


7.1.3.12 void Agent::makeEarlyDecision ()

Here is the call graph for this function:

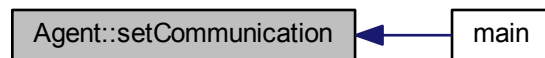


Here is the caller graph for this function:



7.1.3.13 void Agent::setCommunication (int *newCommunication*)

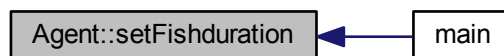
Here is the caller graph for this function:



7.1.3.14 void Agent::setFishduration (float *newFishDuration*)

can be randomize

Here is the caller graph for this function:



7.1.3.15 void Agent::setSkill (int *newskill*)

can be randomize

Here is the caller graph for this function:



7.1.3.16 void Agent::setTemp (float *newTemp*)

Sets the temperature of the water

Parameters

<i>newTemp</i>	the new temperature in degrees Celsius
----------------	--

Sets the temperature of the water

Parameters

<i>newTemp</i>	the new temperature in degrees celsius
----------------	--

Here is the caller graph for this function:



7.1.3.17 void Agent::updateHistory ()

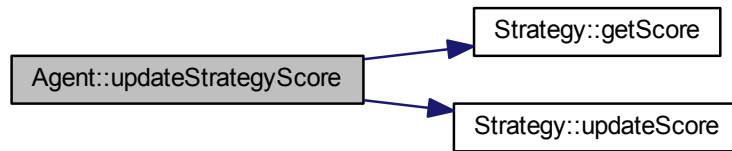
push new decision on

Here is the caller graph for this function:

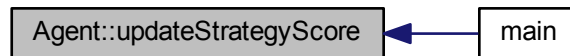


7.1.3.18 void Agent::updateStrategyScore (int *winnigScore*)

Here is the call graph for this function:



Here is the caller graph for this function:



7.1.4 Member Data Documentation

7.1.4.1 int Agent::communication [private]

*from input

7.1.4.2 int Agent::decision [private]

new decision *have to calculate, use makeDecision

7.1.4.3 int Agent::earlydecison [private]

based on history sequence and strategy

7.1.4.4 float Agent::fishduration [private]

*from input

7.1.4.5 vector<int> Agent::history [private]

holds the last 3 outcomes

7.1.4.6 `double Agent::skill` `[private]`

will be 1-5 *from input

7.1.4.7 `vector<Strategy*> Agent::strat` `[private]`

holds the 3 unique strategies

7.1.4.8 `float Agent::temp` `[private]`

*from input

7.1.4.9 `float Agent::threshold` `[private]`

influence threshold, if based on report > 70 will make agent's decision change to 1 < 70 agent's decision remain the same new rule: $p \Rightarrow 85$ change decision to 1 $40 < p < 85$ decision remain $p \leq 40$ decision change to -1

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

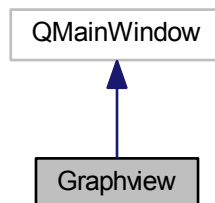
- [agent.h](#)
- [agent.cpp](#)

7.2 Graphview Class Reference

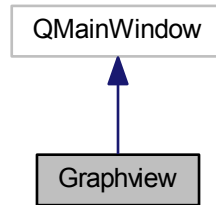
provides a view that shows the collected graphs and allows them to be inserted into a report.

```
#include <graphview.h>
```

Inheritance diagram for Graphview:



Collaboration diagram for Graphview:



Public Member Functions

- [Graphview](#) (`QWidget *parent=0`)
constructor for the [Graphview](#) class
- [~Graphview](#) ()
- void [setupPlot](#) ()
setupPlot

Private Slots

- void [on_actionInsert_Plot_triggered](#) ()
[Graphview::on_actionInsert_Plot_triggered](#).
- void [on_actionSave_Document_triggered](#) ()

Private Attributes

- `Ui::Graphview * ui`

7.2.1 Detailed Description

provides a view that shows the collected graphs and allows them to be inserted into a report.

[Graphview](#) is intended to be used after the simulation has finished. It will accept data from the simulation module defining plots and display them to the users. There is also a report view on the left side that allows users to insert selected graphs to compile a final report.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 `Graphview::Graphview (QWidget * parent = 0) [explicit]`

constructor for the [Graphview](#) class

Here is the call graph for this function:



7.2.2.2 `Graphview::~~Graphview ()`

7.2.3 Member Function Documentation

7.2.3.1 `void Graphview::on_actionInsert_Plot_triggered () [private],[slot]`

[Graphview::on_actionInsert_Plot_triggered.](#)

7.2.3.2 `void Graphview::on_actionSave_Document_triggered () [private],[slot]`

7.2.3.3 `void Graphview::setupPlot ()`

`setupPlot`

configures the plots

Here is the caller graph for this function:



7.2.4 Member Data Documentation

7.2.4.1 `Ui::Graphview* Graphview::ui [private]`

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

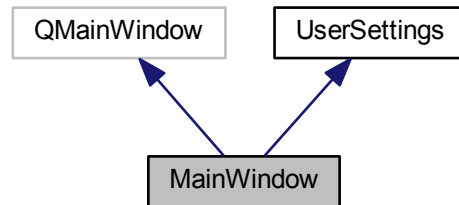
- [graphview.h](#)
- [graphview.cpp](#)

7.3 MainWindow Class Reference

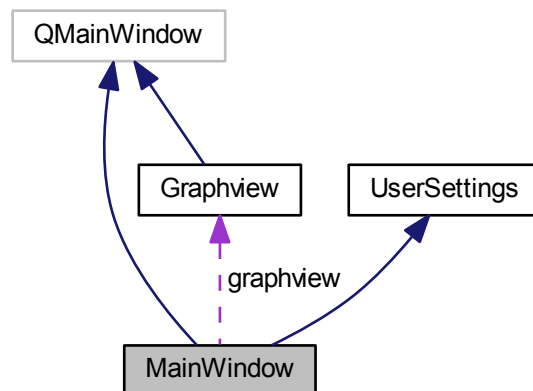
The [MainWindow](#) class Provides the Main windows for the Fisher sim project.

```
#include <mainwindow.h>
```

Inheritance diagram for MainWindow:



Collaboration diagram for MainWindow:



Public Member Functions

- [MainWindow](#) (QWidget *parent=0)
- [~MainWindow](#) ()
- void [log](#) (const QString &text)

Sends a string to the simulation log.

Private Slots

- void [on_fishers_valueChanged](#) (int value)
- void [on_locations_valueChanged](#) (int value)
- void [on_fishtypes_valueChanged](#) (int value)
- void [on_fishpop_valueChanged](#) (int value)
- void [on_runtime_valueChanged](#) (int value)
- void [on_lineEdit_0_textEdited](#) (const QString &arg1)
- void [on_lineEdit_1_textEdited](#) (const QString &arg1)
- void [on_lineEdit_2_textEdited](#) (const QString &arg1)
- void [on_lineEdit_3_textEdited](#) (const QString &arg1)
- void [on_lineEdit_4_textEdited](#) (const QString &arg1)
- void [on_weather_clicked](#) ()
- void [on_reportButton_clicked](#) ()
- void [on_simulateButton_clicked](#) ()

Private Attributes

- Ui::MainWindow * [ui](#)
- [Graphview](#) [graphview](#)

Additional Inherited Members

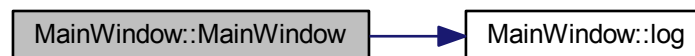
7.3.1 Detailed Description

The [MainWindow](#) class Provides the Main windows for the Fisher sim project.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 MainWindow::MainWindow (QWidget * *parent* = 0) [explicit]

Here is the call graph for this function:



7.3.2.2 MainWindow::~~MainWindow ()

7.3.3 Member Function Documentation

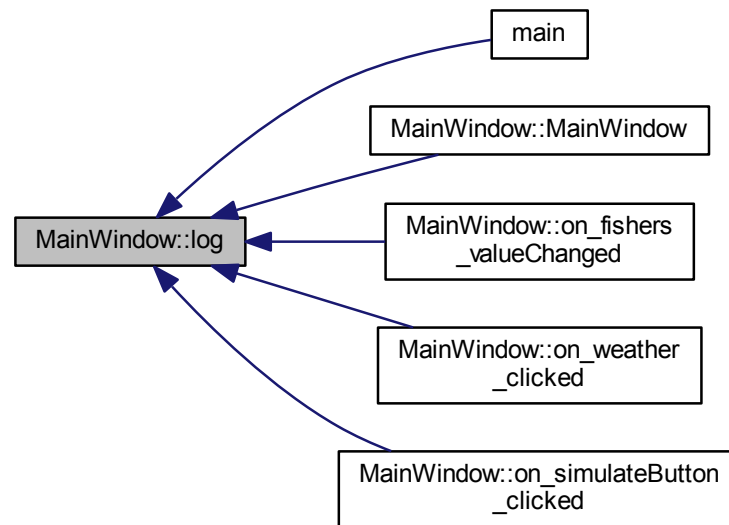
7.3.3.1 void MainWindow::log (const QString & *text*)

Sends a string to the simulation log.

Parameters

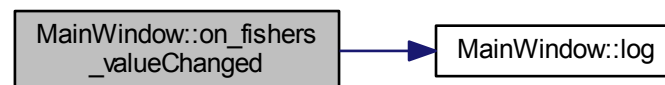
<i>text</i>	to display in the log.
-------------	------------------------

Here is the caller graph for this function:



7.3.3.2 `void MainWindow::on_fishers_valueChanged (int value)` [private],[slot]

Here is the call graph for this function:



7.3.3.3 `void MainWindow::on_fishpop_valueChanged (int value)` [private],[slot]

7.3.3.4 `void MainWindow::on_fishtypes_valueChanged (int value)` [private],[slot]

7.3.3.5 `void MainWindow::on_lineEdit_0_textEdited (const QString & arg1)` [private],[slot]

7.3.3.6 void MainWindow::on_lineEdit_1_textEdited (const QString & *arg1*) [private],[slot]

7.3.3.7 void MainWindow::on_lineEdit_2_textEdited (const QString & *arg1*) [private],[slot]

7.3.3.8 void MainWindow::on_lineEdit_3_textEdited (const QString & *arg1*) [private],[slot]

7.3.3.9 void MainWindow::on_lineEdit_4_textEdited (const QString & *arg1*) [private],[slot]

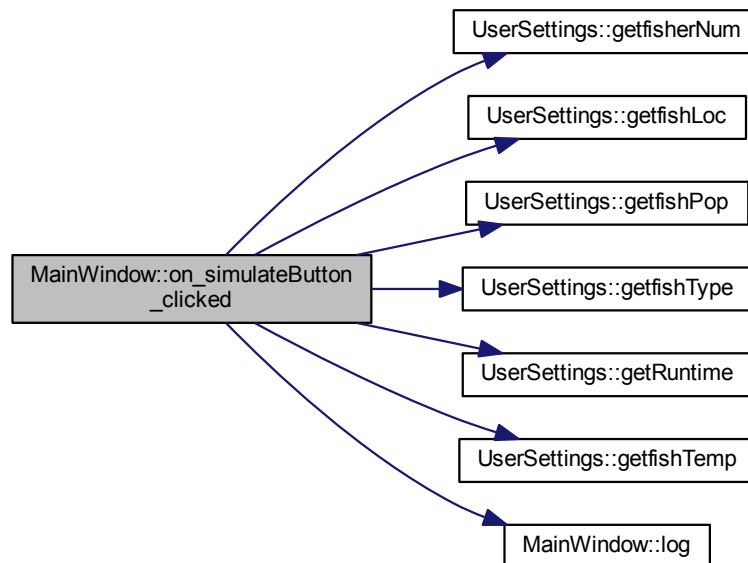
7.3.3.10 void MainWindow::on_locations_valueChanged (int *value*) [private],[slot]

7.3.3.11 void MainWindow::on_reportButton_clicked () [private],[slot]

7.3.3.12 void MainWindow::on_runtime_valueChanged (int *value*) [private],[slot]

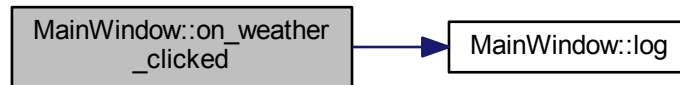
7.3.3.13 void MainWindow::on_simulateButton_clicked () [private],[slot]

Here is the call graph for this function:



7.3.3.14 void MainWindow::on_weather_clicked () [private],[slot]

Here is the call graph for this function:



7.3.4 Member Data Documentation

7.3.4.1 Graphview MainWindow::graphview [private]

7.3.4.2 Ui::MainWindow* MainWindow::ui [private]

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

- [mainwindow.h](#)
- [mainwindow.cpp](#)

7.4 Spot Class Reference

[Spot](#) is used to create a location and calculate how crowded it is.

```
#include <spot.h>
```

Public Member Functions

- [Spot](#) ()
- void [setCap](#) (double cap)
- double [getSpotCapacity](#) ()
- void [setAgentNum](#) (int fisherNum)
- int [getAgentNum](#) ()
- double [crowdness](#) (double goFish)

Private Attributes

- double [maxcapacity](#)
max agents per spot
- int [numAgent](#)
number of agents possibly going fishing per spot

7.4.1 Detailed Description

[Spot](#) is used to create a location and calculate how crowded it is.

7.4.2 Constructor & Destructor Documentation

7.4.2.1 Spot::Spot ()

Constructor for a spot

Precondition

none

Postcondition

numAgent and maxcapacity is initialized to zero

Returns

none

7.4.3 Member Function Documentation

7.4.3.1 double Spot::crowdness (double *goFish*)

Calculate how crowded a spot is

Precondition

Give number of agents that decided to go fishing

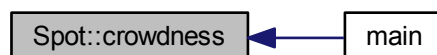
Postcondition

Percentage of crowd is calculated

Returns

Percentage of fisherman going fishing

Here is the caller graph for this function:



7.4.3.2 int Spot::getAgentNum ()

Get back total number agents

Precondition

Number of agents is already set

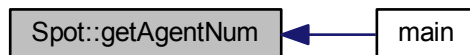
Postcondition

none

Returns

Integer number of total agents (numAgent)

Here is the caller graph for this function:



7.4.3.3 double Spot::getSpotCapacity ()

Get the maxcapacity

Precondition

A capacity is already set

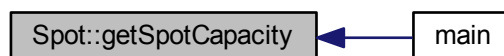
Postcondition

none

Returns

The number of maxcapacity

Here is the caller graph for this function:



7.4.3.4 void Spot::setAgentNum (int *fisherNum*)

Set the number of agents possibly going to spot

Precondition

Give the number of agents called fisherNum

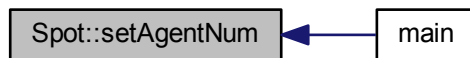
Postcondition

The numAgent is equal to given fisherNum

Returns

none

Here is the caller graph for this function:



7.4.3.5 void Spot::setCap (double *cap*)

Here is the caller graph for this function:



7.4.4 Member Data Documentation

7.4.4.1 double Spot::maxcapacity [private]

max agents per spot

7.4.4.2 int Spot::numAgent [private]

number of agents possibly going fishing per spot

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

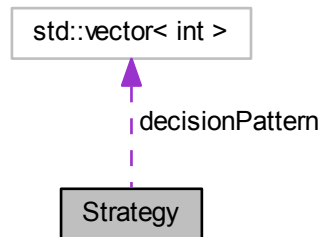
- [spot.h](#)
- [spot.cpp](#)

7.5 Strategy Class Reference

represents a strategy for determining the conditions of going fishing. since each startegy depends on 3 previous outcomes, so posibile output for one strategy is 8. the sequence for the 3 previous outcomes would be: 000,001,010,...,111 special case for starategy: 0->stay at home, 1->go fishing

```
#include <strategy.h>
```

Collaboration diagram for Strategy:



Public Member Functions

- [Strategy](#) (vector< int > randDecision)
Strategy constructor.
- vector< int > [getDecisionPattern](#) ()
- int [getScore](#) ()
returns the score This value represents the number of wins that an agent has made using this strategy
- void [updateScore](#) (int point)
records the secess of this strategy

Private Attributes

- int [score](#)
hold score for strategy
- vector< int > [decisionPattern](#)
contains the record of past decisions

7.5.1 Detailed Description

represents a strategy for determining the conditions of going fishing. since each startegy depends on 3 previous outcomes, so posiible output for one strategy is 8. the sequence for the 3 previous outcomes would be: 000,001,010,...,111
special case for starategy: 0->stay at home, 1->go fishing

7.5.2 Constructor & Destructor Documentation

7.5.2.1 Strategy::Strategy (vector< int > randDecision)

Strategy constructor.

7.5.3 Member Function Documentation

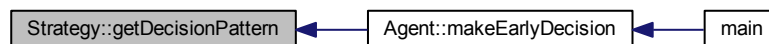
7.5.3.1 vector< int > Strategy::getDecisionPattern ()

Returns the decision pattern used by this strategy

Returns

a vector containing the decision pattern used. 0 represents staying home and 1 going fishing.

Here is the caller graph for this function:



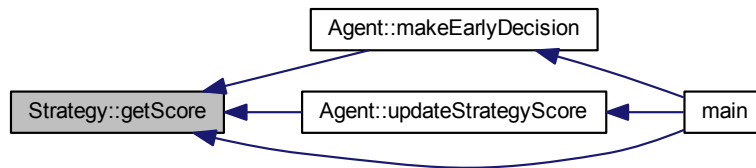
7.5.3.2 int Strategy::getScore ()

returns the score This value represents the number of wins that an agent has made using this strategy

Returns

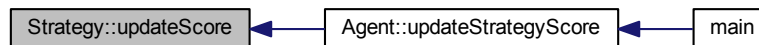
the strategy score

Here is the caller graph for this function:

**7.5.3.3 void Strategy::updateScore (int *point*)**

records the seccess of this strategy

Here is the caller graph for this function:

**7.5.4 Member Data Documentation****7.5.4.1 vector<int> Strategy::decisionPattern [private]**

contains the record of past decisions

7.5.4.2 int Strategy::score [private]

hold score for strategy

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

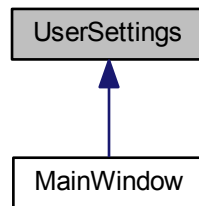
- [strategy.h](#)
- [strategy.cpp](#)

7.6 UserSettings Class Reference

contains the users global simulation parameters.

```
#include <UserSettings.h>
```

Inheritance diagram for UserSettings:



Public Member Functions

- [UserSettings \(\)](#)
- [int getfisherNum \(\)](#)
- [int getfishLoc \(\)](#)
- [int getfishType \(\)](#)
- [int getfishPop \(\)](#)
- [int getfishTemp \(\)](#)
- [int getRuntime \(\)](#)

Protected Attributes

- [int fisherNum](#)
- [int fishLoc](#)
- [int fishType](#)
- [int fishPop](#)
- [int fishTemp](#)
- [int runtime](#)

7.6.1 Detailed Description

contains the users global simulation parameters.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 UserSettings::UserSettings ()

7.6.3 Member Function Documentation

7.6.3.1 int UserSettings::getfisherNum ()

Returns the number of Fishers to use in the simulation

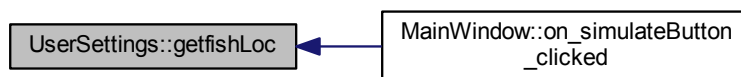
Here is the caller graph for this function:



7.6.3.2 `int UserSettings::getfishLoc ()`

Returns the number of different locations

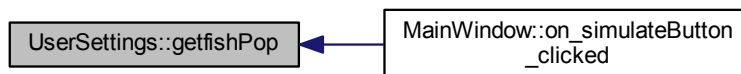
Here is the caller graph for this function:



7.6.3.3 `int UserSettings::getfishPop ()`

Returns the initial population of fish when the simulation starts.

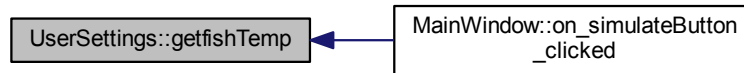
Here is the caller graph for this function:



7.6.3.4 `int UserSettings::getfishTemp ()`

Returns the conditions: overcast, snow, rain.

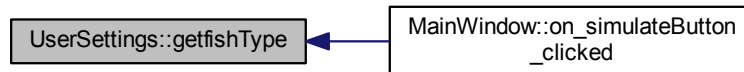
Here is the caller graph for this function:



7.6.3.5 `int UserSettings::getfishType ()`

Returns the number of fish types.

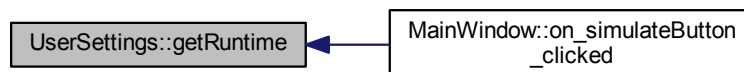
Here is the caller graph for this function:



7.6.3.6 `int UserSettings::getRuntime ()`

Returns the number of days to run the simulation.

Here is the caller graph for this function:



7.6.4 Member Data Documentation

7.6.4.1 `int UserSettings::fisherNum` `[protected]`

The number of Fishers to use in the simulation

7.6.4.2 `int UserSettings::fishLoc` [protected]

The number of different locations

7.6.4.3 `int UserSettings::fishPop` [protected]

The initial population of fish when the simulation starts.

7.6.4.4 `int UserSettings::fishTemp` [protected]

The conditions: overcast, snow, rain

7.6.4.5 `int UserSettings::fishType` [protected]

The number of fish types.

7.6.4.6 `int UserSettings::runtime` [protected]

The number of days to run the simulation

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

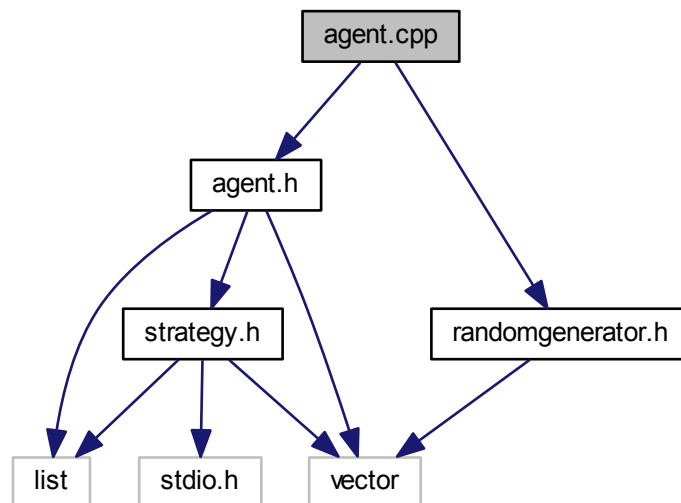
- [UserSettings.h](#)
- [UserSettings.cpp](#)

Chapter 8

File Documentation

8.1 agent.cpp File Reference

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



Functions

- void `initAgent` (list< `Agent` * > *allAgent, int numAgent, list< `Strategy` * > stratlist)

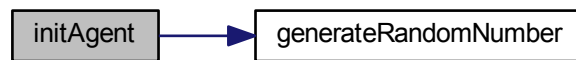
Variables

- static list< Agent > all_agent

8.1.1 Function Documentation

8.1.1.1 void initAgent (list< Agent * > * allAgent, int numAgent, list< Strategy * > stratlist)

Here is the call graph for this function:



Here is the caller graph for this function:



8.1.2 Variable Documentation

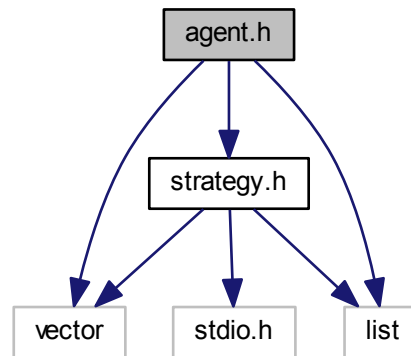
8.1.2.1 list<Agent> all_agent [static]

8.2 agent.h File Reference

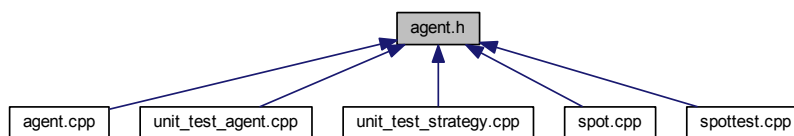
agent represents 1 agent's experience and decisions

```
#include "strategy.h"
#include <list>
#include <vector>
```

Include dependency graph for agent.h:



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



Classes

- class [Agent](#)
agent represents 1 agent's experience and decisions

Functions

- void [initAgent](#) (list< [Agent](#) * > *allAgent, int numAgent, list< [Strategy](#) * > stratlist)

Variables

- static int [numOfAgent](#)

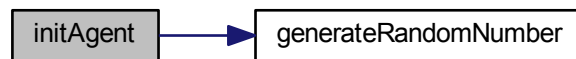
8.2.1 Detailed Description

agent represents 1 agent's experience and decisions

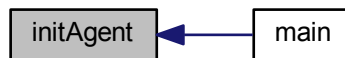
8.2.2 Function Documentation

8.2.2.1 `void initAgent (list< Agent * > * allAgent, int numAgent, list< Strategy * > stratlist)`

Here is the call graph for this function:



Here is the caller graph for this function:



8.2.3 Variable Documentation

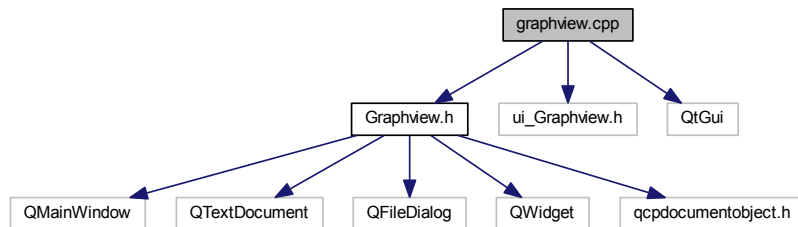
8.2.3.1 `int numOfAgent [static]`

8.3 Algorithms.dox File Reference

8.4 graphview.cpp File Reference

```
#include "Graphview.h"  
#include "ui_Graphview.h"  
#include <QtGui>
```

Include dependency graph for graphview.cpp:

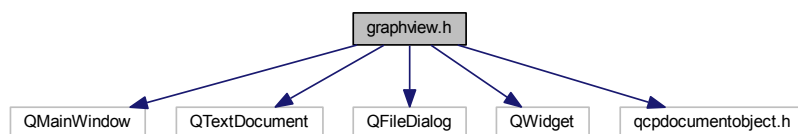


8.5 graphview.h File Reference

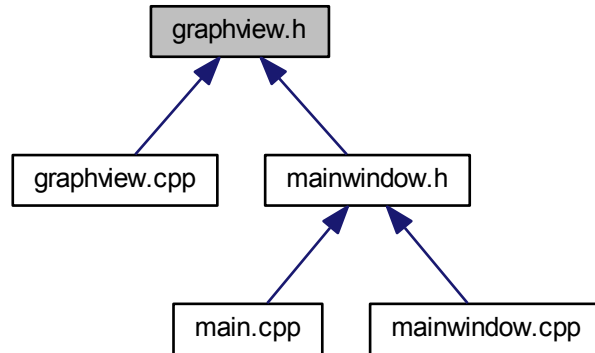
provides a view that shows the collected graphs and allows them to be inserted into a report.

```
#include <QMainWindow>
#include <QTextDocument>
#include <QFileDialog>
#include <QWidget>
#include "qcpdocumentobject.h"
```

Include dependency graph for graphview.h:



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



Classes

- class [Graphview](#)

provides a view that shows the collected graphs and allows them to be inserted into a report.

Namespaces

- [Ui](#)

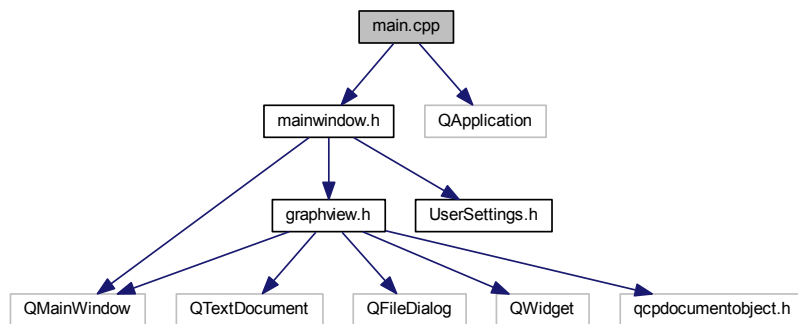
8.5.1 Detailed Description

provides a view that shows the collected graphs and allows them to be inserted into a report.

8.6 main.cpp File Reference

```
#include "mainwindow.h"  
#include <QApplication>
```


Include dependency graph for main.cpp:



Functions

- int `main` (int argc, char *argv[])

8.6.1 Function Documentation

8.6.1.1 int main (int argc, char * argv[])

Here is the call graph for this function:



8.7 mainpage.dox File Reference

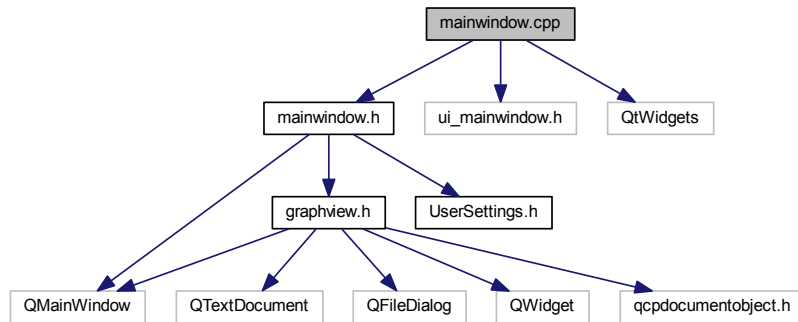
8.8 mainwindow.cpp File Reference

```

#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QtWidgets>

```

Include dependency graph for mainwindow.cpp:

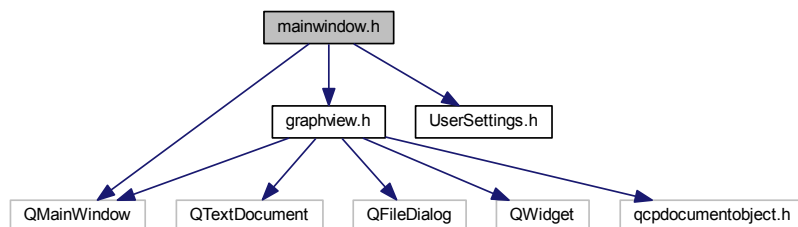


8.9 mainwindow.h File Reference

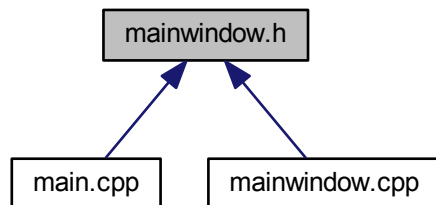
mainwindow creates the primary GUI display

```
#include <QMainWindow>
#include "graphview.h"
#include "UserSettings.h"
```

Include dependency graph for mainwindow.h:



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



Classes

- class [MainWindow](#)

The [MainWindow](#) class Provides the Main windows for the Fisher sim project.

Namespaces

- [Ui](#)

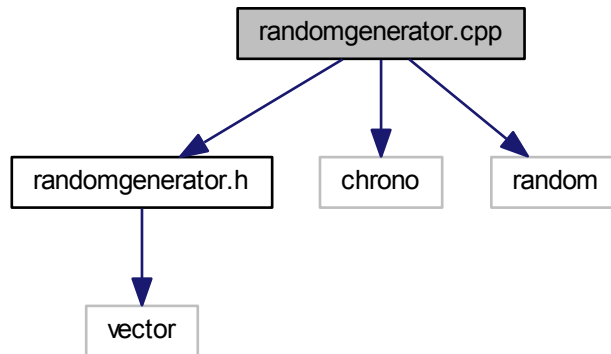
8.9.1 Detailed Description

mainwindow creates the primary GUI display

8.10 randomgenerator.cpp File Reference

```
#include "randomgenerator.h"
#include <chrono>
#include <random>
```

Include dependency graph for randomgenerator.cpp:



Functions

- `vector< int > generateRandomNumber (int lowerBound, int upperBound, int length)`

Generates a vector of random numbers from some minimum bound to an upper bound.

8.10.1 Function Documentation

8.10.1.1 `vector< int > generateRandomNumber (int lowerBound, int upperBound, int length)`

Generates a vector of random numbers from some minimum bound to an upper bound.

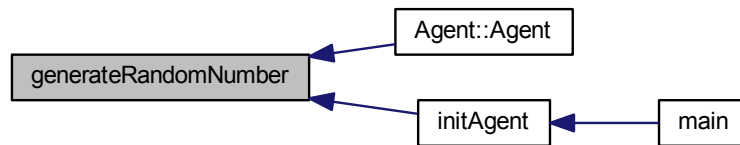
Parameters

<i>lowerBound</i>	the lower bound that the random numbers can have.
<i>upperBound</i>	the upper bound of the random numbers
<i>length</i>	the number of random numbers to generate.

Returns

a vector containing length number of elements from 0 - (length-1)

Here is the caller graph for this function:

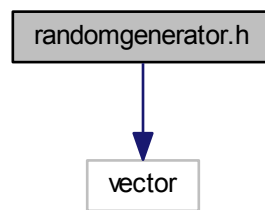


8.11 randomgenerator.h File Reference

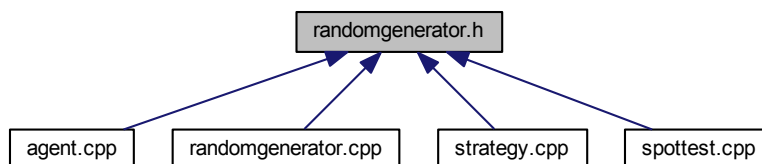
construct a trivial random generator engine from a time-based seed:

```
#include <vector>
```

Include dependency graph for `randomgenerator.h`:



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



Functions

- `vector< int > generateRandomNumber (int lowerBound, int upperBound, int length)`

Generates a vector of random numbers from some minimum bound to an upper bound.

8.11.1 Detailed Description

construct a trivial random generator engine from a time-based seed:

8.11.2 Function Documentation

8.11.2.1 `vector<int> generateRandomNumber (int lowerBound, int upperBound, int length)`

Generates a vector of random numbers from some minimum bound to an upper bound.

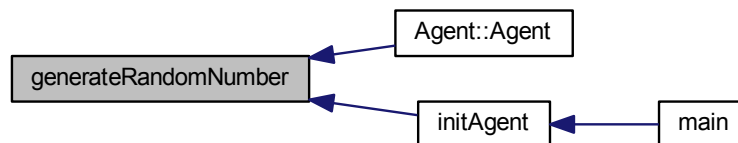
Parameters

<i>lowerBound</i>	the lower bound that the random numbers can have.
<i>upperBound</i>	the upper bound of the random numbers
<i>length</i>	the number of random numbers to generate.

Returns

a vector containing length number of elements from 0 - (length-1)

Here is the caller graph for this function:



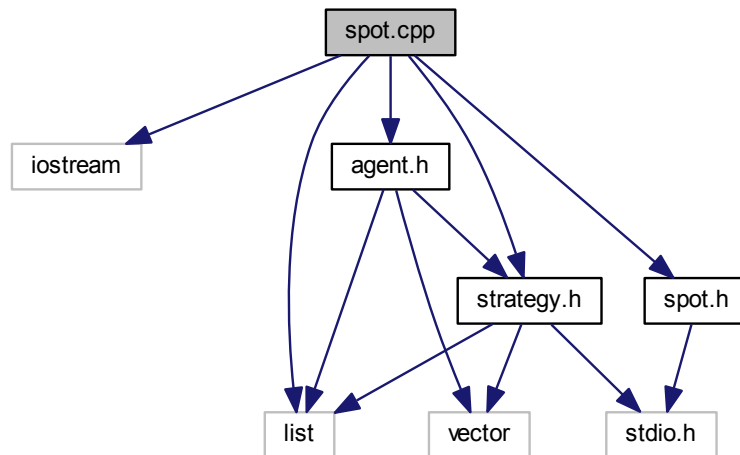
8.12 spot.cpp File Reference

```

#include <iostream>
#include <list>
#include "agent.h"
#include "strategy.h"
#include "spot.h"

```

Include dependency graph for spot.cpp:

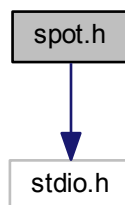


8.13 spot.h File Reference

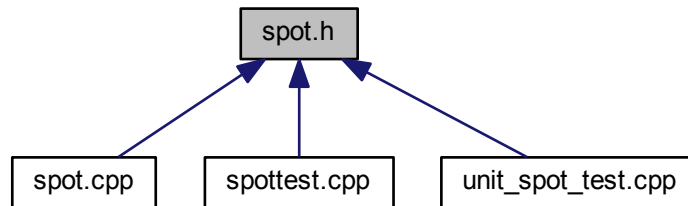
Used to create a spot and calculate how crowded a spot is.

```
#include <stdio.h>
```

Include dependency graph for spot.h:



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



Classes

- class [Spot](#)

[Spot](#) is used to create a location and calculate how crowded it is.

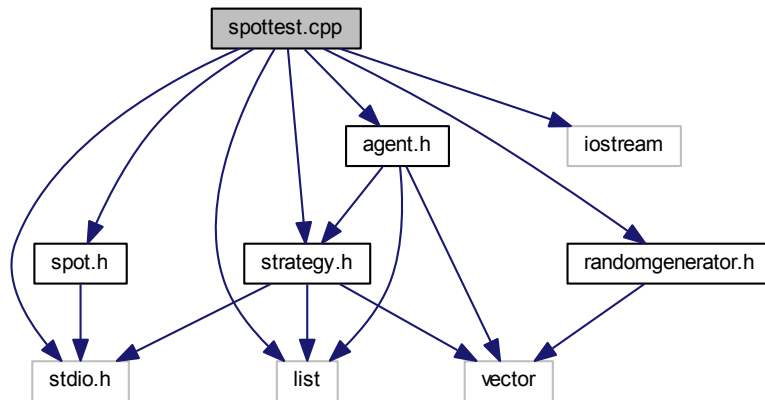
8.13.1 Detailed Description

Used to create a spot and calculate how crowded a spot is.

8.14 spottest.cpp File Reference

```
#include <stdio.h>
#include <iostream>
#include <list>
#include "randomgenerator.h"
#include "agent.h"
#include "strategy.h"
#include "spot.h"
```


Include dependency graph for spottest.cpp:



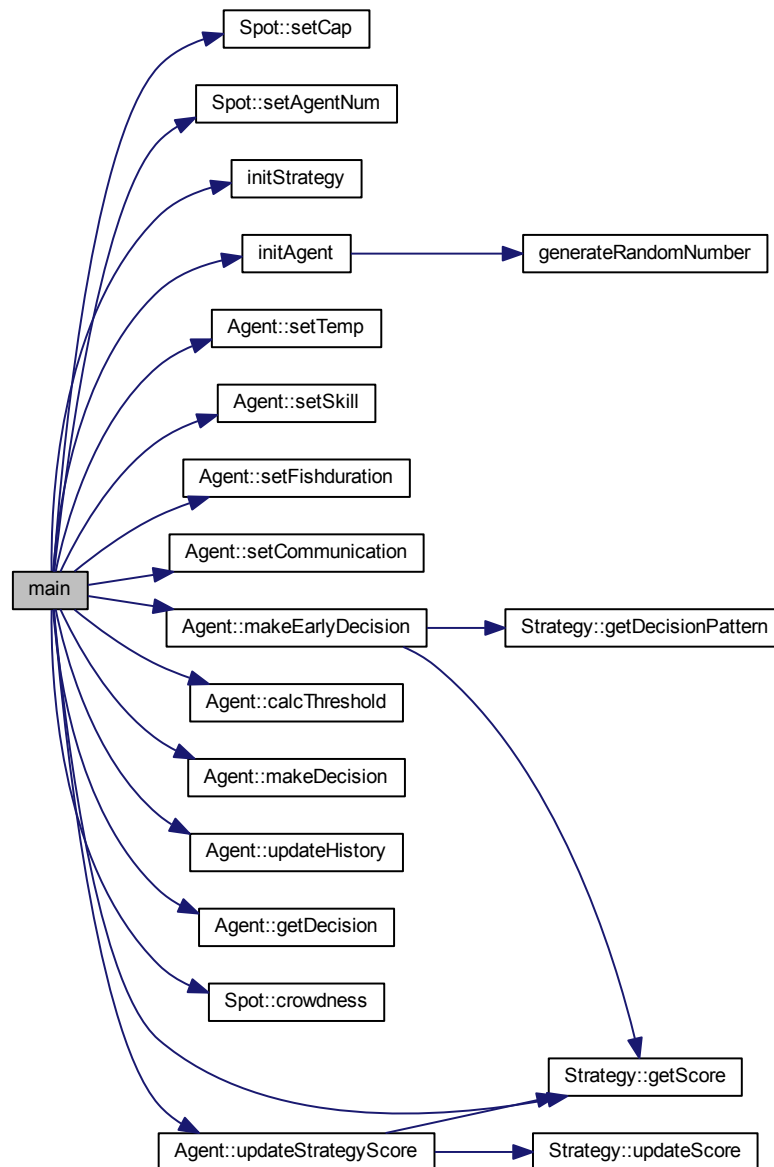
Functions

- `int main ()`

8.14.1 Function Documentation

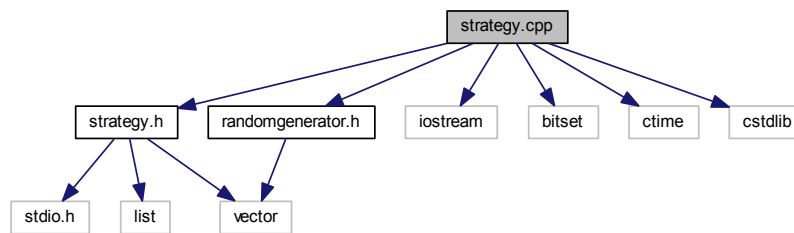
8.14.1.1 `int main ()`

Here is the call graph for this function:



8.15 strategy.cpp File Reference

```
#include "strategy.h"
#include "randomgenerator.h"
#include <iostream>
#include <bitset>
#include <ctime>
#include <cstdlib>
Include dependency graph for strategy.cpp:
```



Functions

- void `initStrategy` (list< `Strategy` * > *allStrategy)
initializes all of the strategies

8.15.1 Function Documentation

8.15.1.1 void `initStrategy` (list< `Strategy` * > * *allStrategy*)

initializes all of the strategies

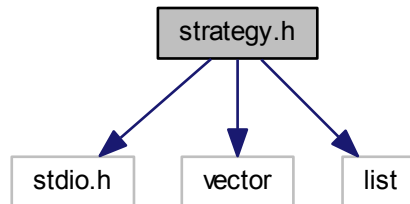
Here is the caller graph for this function:



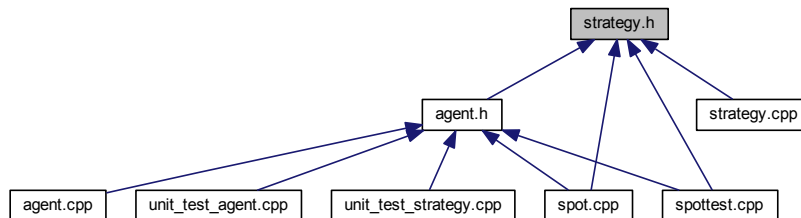
8.16 strategy.h File Reference

represents a strategy for determining the conditions of going fishing.

```
#include <stdio.h>
#include <vector>
#include <list>
Include dependency graph for strategy.h:
```



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



Classes

- class [Strategy](#)

represents a strategy for determining the conditions of going fishing. since each startegy depends on 3 previous outcomes, so possiible output for one strategy is 8. the sequence for the 3 previous outcomes would be: 000,001,010,...,111 special case for starategy: 0-> stay at home, 1-> go fishing

Functions

- void [initStrategy](#) (list< [Strategy](#) * > *allStrategy)
initializes all of the strategies

8.16.1 Detailed Description

represents a strategy for determining the conditions of going fishing.

8.16.2 Function Documentation

8.16.2.1 void initStrategy (list< Strategy * > * allStrategy)

initializes all of the strategies

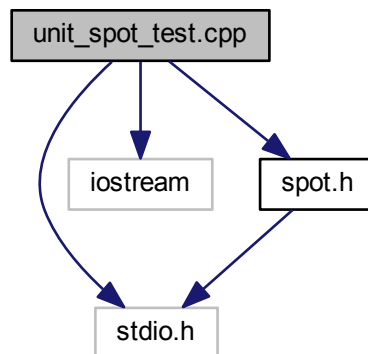
Here is the caller graph for this function:



8.17 unit_spot_test.cpp File Reference

```
#include <stdio.h>
#include <iostream>
#include "spot.h"
```

Include dependency graph for unit_spot_test.cpp:



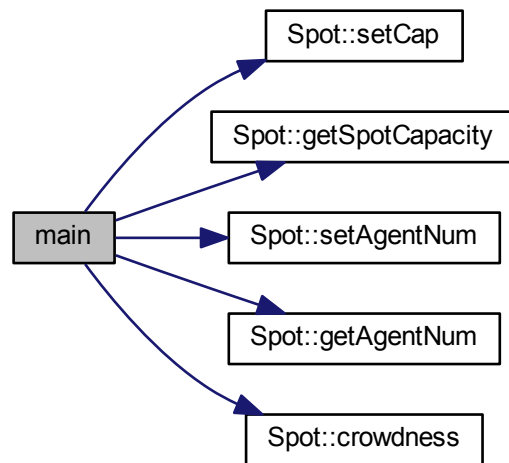
Functions

- int [main](#) ()

8.17.1 Function Documentation

8.17.1.1 int main ()

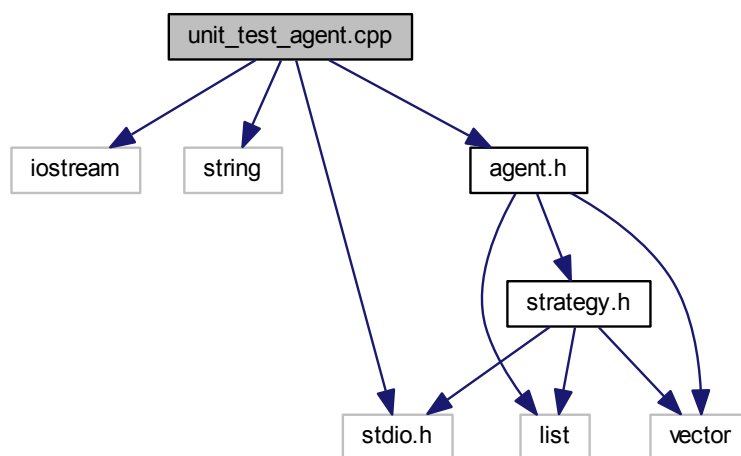
Here is the call graph for this function:



8.18 unit_test_agent.cpp File Reference

```
#include <iostream>
#include <string>
#include <stdio.h>
#include "agent.h"
```

Include dependency graph for unit_test_agent.cpp:



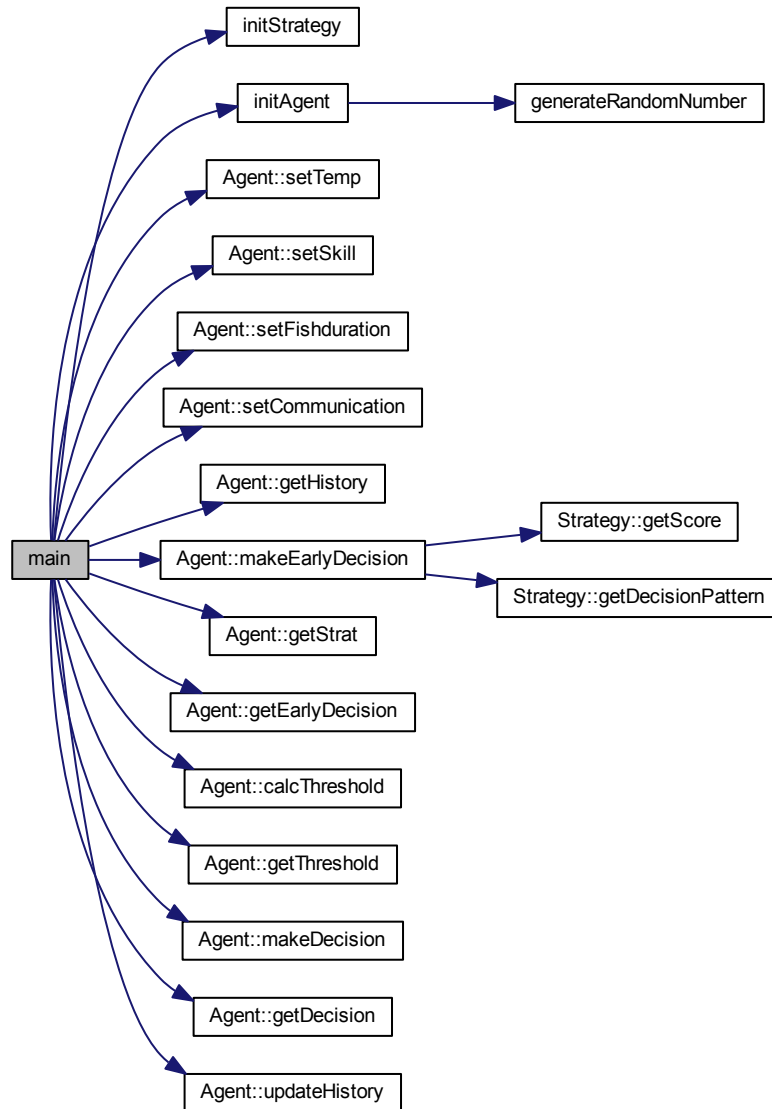
Functions

- void `main` ()

8.18.1 Function Documentation

8.18.1.1 void main ()

Here is the call graph for this function:



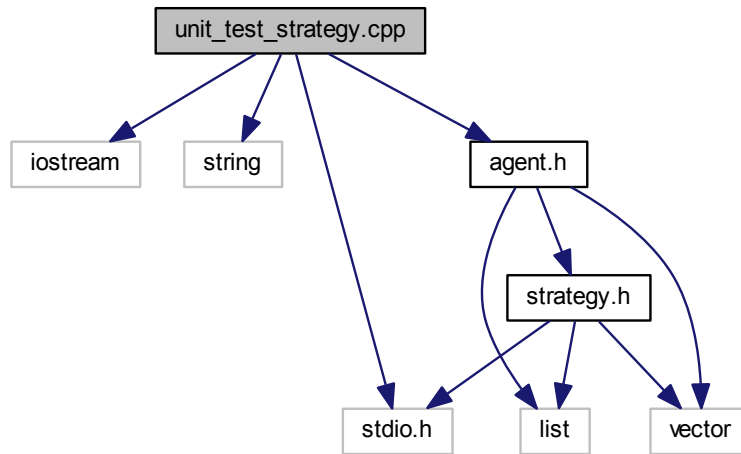
8.19 unit_test_strategy.cpp File Reference

```

#include <iostream>
#include <string>
#include <stdio.h>
#include "agent.h"

```


Include dependency graph for unit_test_strategy.cpp:



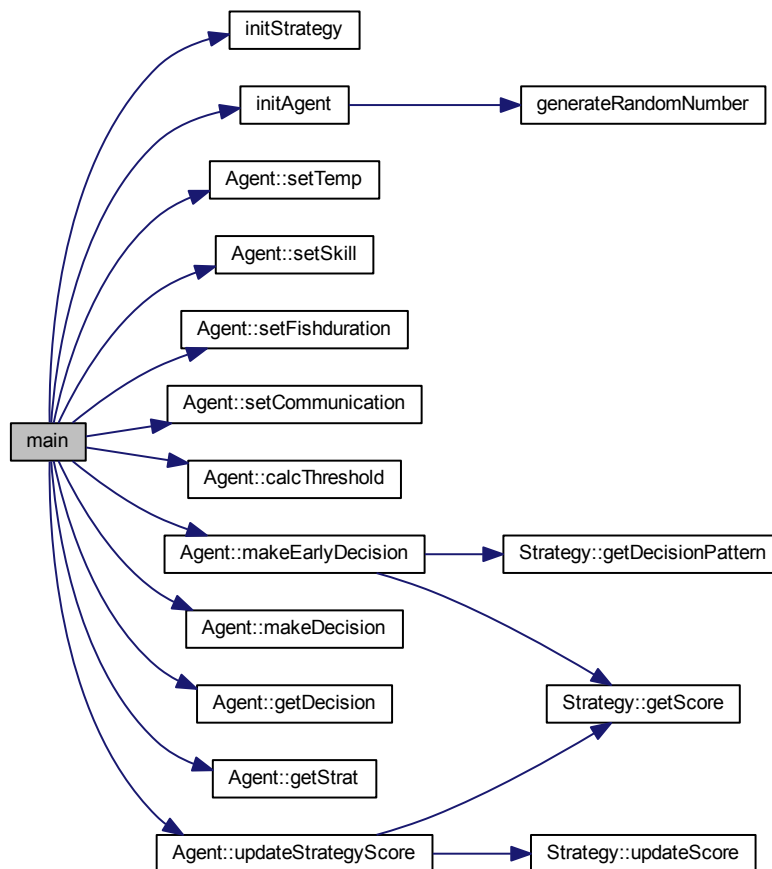
Functions

- void `main` ()

8.19.1 Function Documentation

8.19.1.1 void main ()

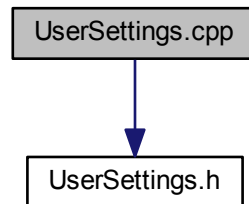
Here is the call graph for this function:



8.20 UserSettings.cpp File Reference

```
#include "UserSettings.h"
```

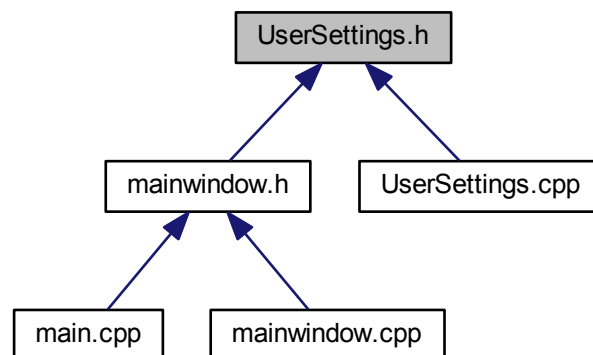
Include dependency graph for UserSettings.cpp:



8.21 UserSettings.h File Reference

contains the users global simulation parameters.

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



Classes

- class [UserSettings](#)

contains the users global simulation parameters.

8.21.1 Detailed Description

contains the users global simulation parameters.

Index

- ~Graphview
 - Graphview, [26](#)
- ~MainWindow
 - MainWindow, [28](#)
- Agent, [15](#)
 - Agent, [17](#)
 - calcThreshold, [17](#)
 - communication, [23](#)
 - decision, [23](#)
 - earlydecision, [23](#)
 - fishduration, [23](#)
 - getCommunication, [17](#)
 - getDecision, [17](#)
 - getEarlyDecision, [18](#)
 - getFishDuration, [18](#)
 - getHistory, [19](#)
 - getSkill, [19](#)
 - getStrat, [19](#)
 - getTemp, [19](#)
 - getThreshold, [19](#)
 - history, [23](#)
 - makeDecision, [20](#)
 - makeEarlyDecision, [20](#)
 - setCommunication, [21](#)
 - setFishduration, [21](#)
 - setSkill, [21](#)
 - setTemp, [22](#)
 - skill, [23](#)
 - strat, [24](#)
 - temp, [24](#)
 - threshold, [24](#)
 - updateHistory, [22](#)
 - updateStrategyScore, [22](#)
- agent.cpp, [43](#)
 - all_agent, [44](#)
 - initAgent, [44](#)
- agent.h, [44](#)
 - initAgent, [46](#)
 - numOfAgent, [46](#)
- Algorithms.dox, [46](#)
- all_agent
 - agent.cpp, [44](#)
- calcThreshold
 - Agent, [17](#)
- communication
 - Agent, [23](#)
- crowdness
 - Spot, [33](#)
- decision
 - Agent, [23](#)
- decisionPattern
 - Strategy, [38](#)
- earlydecision
 - Agent, [23](#)
- fishLoc
 - UserSettings, [41](#)
- fishPop
 - UserSettings, [42](#)
- fishTemp
 - UserSettings, [42](#)
- fishType
 - UserSettings, [42](#)
- fishduration
 - Agent, [23](#)
- fisherNum
 - UserSettings, [41](#)
- generateRandomNumber
 - randomgenerator.cpp, [52](#)
 - randomgenerator.h, [54](#)
- getAgentNum
 - Spot, [33](#)
- getCommunication
 - Agent, [17](#)
- getDecision
 - Agent, [17](#)
- getDecisionPattern
 - Strategy, [37](#)
- getEarlyDecision
 - Agent, [18](#)
- getFishDuration
 - Agent, [18](#)
- getHistory
 - Agent, [19](#)
- getRuntime
 - UserSettings, [41](#)
- getScore

- Strategy, 37
- getSkill
 - Agent, 19
- getSpotCapacity
 - Spot, 34
- getStrat
 - Agent, 19
- getTemp
 - Agent, 19
- getThreshold
 - Agent, 19
- getfishLoc
 - UserSettings, 40
- getfishPop
 - UserSettings, 40
- getfishTemp
 - UserSettings, 40
- getfishType
 - UserSettings, 41
- getfisherNum
 - UserSettings, 39
- Graphview, 24
 - ~Graphview, 26
 - Graphview, 25
 - on_actionInsert_Plot_triggered, 26
 - on_actionSave_Document_triggered, 26
 - setupPlot, 26
 - ui, 26
- graphview
 - MainWindow, 32
- graphview.cpp, 46
- graphview.h, 47
- history
 - Agent, 23
- initAgent
 - agent.cpp, 44
 - agent.h, 46
- initStrategy
 - strategy.cpp, 59
 - strategy.h, 61
- log
 - MainWindow, 28
- main
 - main.cpp, 49
 - spottest.cpp, 57
 - unit_spot_test.cpp, 61
 - unit_test_agent.cpp, 63
 - unit_test_strategy.cpp, 65
- MainWindow, 27
 - ~MainWindow, 28
 - graphview, 32
 - log, 28
 - MainWindow, 28
 - on_fishers_valueChanged, 30
 - on_fishpop_valueChanged, 30
 - on_fishtypes_valueChanged, 30
 - on_lineEdit_0_textEdited, 30
 - on_lineEdit_1_textEdited, 30
 - on_lineEdit_2_textEdited, 31
 - on_lineEdit_3_textEdited, 31
 - on_lineEdit_4_textEdited, 31
 - on_locations_valueChanged, 31
 - on_reportButton_clicked, 31
 - on_runtime_valueChanged, 31
 - on_simulateButton_clicked, 31
 - on_weather_clicked, 31
 - ui, 32
- mainpage.dox, 49
- mainwindow.cpp, 49
- mainwindow.h, 50
- makeDecision
 - Agent, 20
- makeEarlyDecision
 - Agent, 20
- maxcapacity
 - Spot, 35
- numAgent
 - Spot, 35
- numOfAgent
 - agent.h, 46
- on_actionInsert_Plot_triggered
 - Graphview, 26
- on_actionSave_Document_triggered
 - Graphview, 26
- on_fishers_valueChanged
 - MainWindow, 30
- on_fishpop_valueChanged
 - MainWindow, 30
- on_fishtypes_valueChanged
 - MainWindow, 30
- on_lineEdit_0_textEdited
 - MainWindow, 30
- on_lineEdit_1_textEdited
 - MainWindow, 30
- on_lineEdit_2_textEdited
 - MainWindow, 31
- on_lineEdit_3_textEdited
 - MainWindow, 31
- on_lineEdit_4_textEdited
 - MainWindow, 31
- on_locations_valueChanged
 - MainWindow, 31

- on_reportButton_clicked
 - MainWindow, 31
- on_runtime_valueChanged
 - MainWindow, 31
- on_simulateButton_clicked
 - MainWindow, 31
- on_weather_clicked
 - MainWindow, 31
- randomgenerator.cpp, 51
 - generateRandomNumber, 52
- randomgenerator.h, 53
 - generateRandomNumber, 54
- runtime
 - UserSettings, 42
- score
 - Strategy, 38
- setAgentNum
 - Spot, 34
- setCap
 - Spot, 35
- setCommunication
 - Agent, 21
- setFishduration
 - Agent, 21
- setSkill
 - Agent, 21
- setTemp
 - Agent, 22
- setupPlot
 - Graphview, 26
- skill
 - Agent, 23
- Spot, 32
 - crowdness, 33
 - getAgentNum, 33
 - getSpotCapacity, 34
 - maxcapacity, 35
 - numAgent, 35
 - setAgentNum, 34
 - setCap, 35
 - Spot, 33
- spot.cpp, 54
- spot.h, 55
- spottest.cpp, 56
 - main, 57
- strat
 - Agent, 24
- Strategy, 36
 - decisionPattern, 38
 - getDecisionPattern, 37
 - getScore, 37
 - score, 38
 - Strategy, 37
 - updateScore, 38
- strategy.cpp, 59
 - initStrategy, 59
- strategy.h, 59
 - initStrategy, 61
- temp
 - Agent, 24
- threshold
 - Agent, 24
- Ui, 13
- ui
 - Graphview, 26
 - MainWindow, 32
- unit_spot_test.cpp, 61
 - main, 61
- unit_test_agent.cpp, 62
 - main, 63
- unit_test_strategy.cpp, 64
 - main, 65
- updateHistory
 - Agent, 22
- updateScore
 - Strategy, 38
- updateStrategyScore
 - Agent, 22
- UserSettings, 38
 - fishLoc, 41
 - fishPop, 42
 - fishTemp, 42
 - fishType, 42
 - fisherNum, 41
 - getRuntime, 41
 - getfishLoc, 40
 - getfishPop, 40
 - getfishTemp, 40
 - getfishType, 41
 - getfisherNum, 39
 - runtime, 42
 - UserSettings, 39
- UserSettings.cpp, 67
- UserSettings.h, 67