Fisher sim

0.0.1

Generated by Doxygen 1.8.9.1

Sat May 9 2015 15:33:42

Contents

Chapter 1

Fisher Sim - Introduction

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++ complier for your system. If you do not already have a complier installed and are on a Windows system then a suitable complier can be obtained by installing a version of Microsoft's visual studio express. On Debian Linux systems, a c++ complier 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 2

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 determine 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 3

Namespace Index

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

Chapter 5

Class Index

5.1 Class List

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

Chapter 6

Namespace Documentation

6.1 Ui Namespace Reference

Chapter 7

Class Documentation

7.1 Agent Class Reference

Public Member Functions

- Agent (vector$<$ Strategy $*$ $>$ newStrat, int newSkill, int newCom)
- void updateStrategyScore (int majorityScore)
- void calcThreshold ()
- void makeEarlyDecision ()
- void makeDecision ()
- void updateHistory ()
- void updateBoostScore (int index, int score)
- void adaptNewStrat ()
- void updateAgentScore (int majorityScore)
- void resetAgentScore ()
- void setTemp (float newTemp)
- void setSkill (int newskill)
- void setFishingduration (int newFishingDuration)
- void setCommunication (int newCommunication)
- void setStrategy (vector$<$ Strategy $*$ $>$ newStrat)
- vector$<$ int $>$ getHistory ()
- int getDecision ()
- int getCommunication ()
- int getSkill ()
- float getTemp ()
- int getFishingDuration ()
- int getEarlyDecision ()
- float getThreshold ()
- int getBoostScore (int index)
- int getAgentScore (int index)
- vector$<$ Strategy $*$ $>$ getStrat ()

Private Attributes

- vector$<$ Strategy $*$ $>$ strat
- vector$<$ int $>$ history
- int decision
- int earlydecison
- int skill
- int fishingduration
- float temp
- int communication
- float threshold
- int booststore [2]
- int agentscore [2]

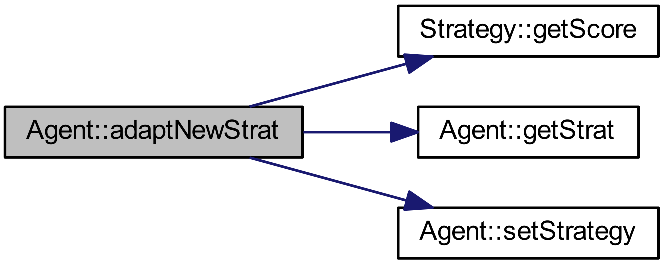
7.1.1 Constructor & Destructor Documentation

7.1.1.1 Agent::Agent (vector$<$ Strategy $*$ $>$ *newStrat,* int *newSkill,* int *newCom*)

7.1.2 Member Function Documentation

7.1.2.1 void Agent::adaptNewStrat ( )

Here is the call graph for this function:



7.1.2.2 void Agent::calcThreshold ( )

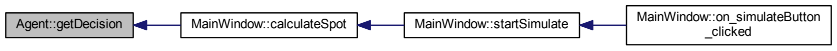
7.1.2.3 int Agent::getAgentScore (int *index*)

7.1.2.4 int Agent::getBoostScore (int *index*)

7.1.2.5 int Agent::getCommunication ( )

7.1.2.6 int Agent::getDecision ( )

Here is the caller graph for this function:


Agent::getDecision
MainWindow::calculateSpot
MainWindow::startSimulate
MainWindow::on_simulateButton _clicked

7.1.2.7 int Agent::getEarlyDecision ( )

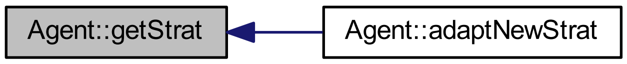
7.1.2.8 int Agent::getFishingDuration ( )

7.1.2.9 vector< int > Agent::getHistory ( )

7.1.2.10 int Agent::getSkill ( )

7.1.2.11 vector< Strategy * > Agent::getStrat ( )

Here is the caller graph for this function:



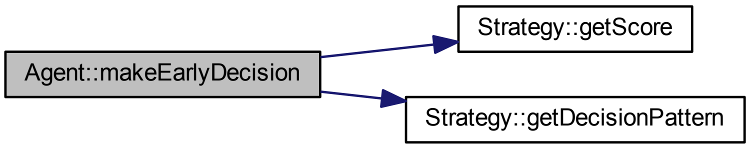
7.1.2.12 float Agent::getTemp ( )

7.1.2.13 float Agent::getThreshold ( )

7.1.2.14 void Agent::makeDecision ( )

7.1.2.15 void Agent::makeEarlyDecision ( )

Here is the call graph for this function:



7.1.2.16 void Agent::resetAgentScore ( )

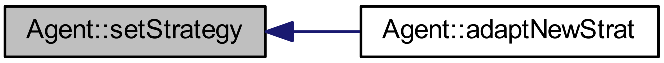
7.1.2.17 void Agent::setCommunication (int *newCommunication*)

7.1.2.18 void Agent::setFishingduration (int *newFishingDuration*)

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

7.1.2.20 void Agent::setStrategy (vector< Strategy * > *newStrat*)

Here is the caller graph for this function:



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

Sets the temperature of the water

Parameters

newTemp	the new temperature in degrees celsius

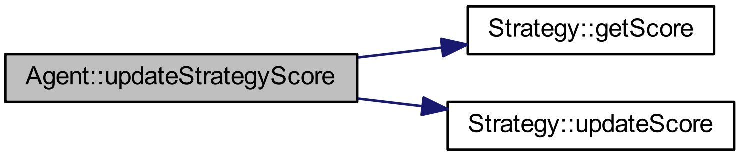
7.1.2.22 void Agent::updateAgentScore (int *majorityScore*)

7.1.2.23 void Agent::updateBoostScore (int *index,* int *score*)

7.1.2.24 void Agent::updateHistory ( )

7.1.2.25 void Agent::updateStrategyScore (int *majorityScore*)

Here is the call graph for this function:



7.1.3 Member Data Documentation

7.1.3.1 int Agent::agentscore[2] `[private]`

7.1.3.2 int Agent::boostscore[2] `[private]`

7.1.3.3 int Agent::communication `[private]`

7.1.3.4 int Agent::decision `[private]`

7.1.3.5 int Agent::earlydecison `[private]`

7.1.3.6 int Agent::fishingduration `[private]`

7.1.3.7 vector<int> Agent::history `[private]`

7.1.3.8 int Agent::skill `[private]`

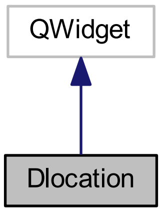
7.1.3.9 vector<Strategy *> Agent::strat `[private]`

7.1.3.10 float Agent::temp `[private]`

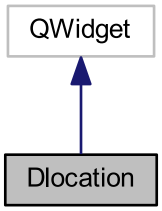
7.1.3.11 float Agent::threshold `[private]`

7.2 Dlocation Class Reference

Inheritance diagram for Dlocation:



Collaboration diagram for Dlocation:



Public Member Functions

- Dlocation (QWidget ∗parent)
- bool setPop (int number)
- int getPop (void)
- bool setCenter (int x, int y)

Protected Member Functions

- void paintEvent (QPaintEvent ∗event) Q_DECL_OVERRIDE

7.2.1 Constructor & Destructor Documentation

7.2.1.1 Dlocation::Dlocation (QWidget ∗ *parent*)

7.2.2 Member Function Documentation

7.2.2.1 int Dlocation::getPop (void)

7.2.2.2 void Dlocation::paintEvent (QPaintEvent ∗ *event*) [protected]

7.2.2.3 bool Dlocation::setCenter (int *x,* int *y*)

7.2.2.4 bool Dlocation::setPop (int *number*)

7.3 Drawing Class Reference

Provides a method of drawing locations and the number of people in each location.

Public Member Functions

- Drawing (QGraphicsView ∗graphics_view, QGraphicsScene ∗drawing_scene)
- void SetNumberOfLocations (int number)
- void DrawPerson (int x, int y)
- double getLocationCenterX (int location)
- double getLocationCenterY (int location)
- void SetLocationPop (int location, int pop)
- void DrawLocationPop (int location)
- void ReDraw ()
- void SetDay (int day)

Public Attributes

- QGraphicsScene ∗ scene
- QGraphicsView ∗ view
- int NumberOfLocations
- int locationPop [10]
- int CurrentDay

7.3.1 Detailed Description

Provides a method of drawing locations and the number of people in each location.

Drawing is used by the main window to create the real time display. It provides methods to get and set the number of locations and the number of people at each location. Setting different values is not updated on the screen without calling ReDraw.

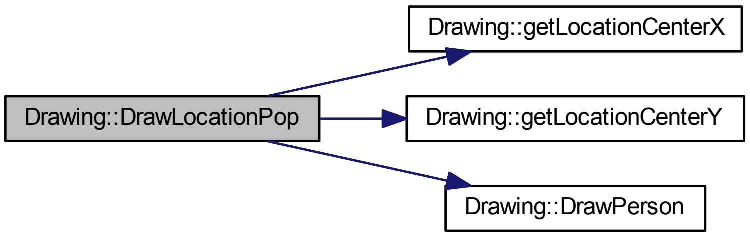
7.3.2 Constructor & Destructor Documentation

7.3.2.1 Drawing::Drawing (QGraphicsView ∗ *graphics_view,* QGraphicsScene ∗ *drawing_scene*)

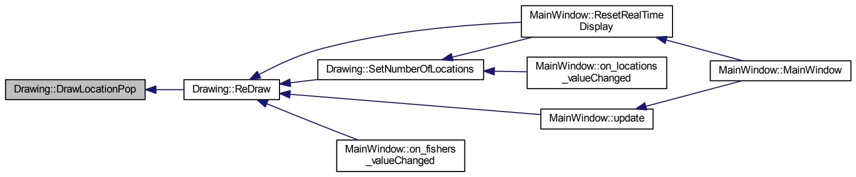
7.3.3 Member Function Documentation

7.3.3.1 void Drawing::DrawLocationPop (int *location*)

Here is the call graph for this function:

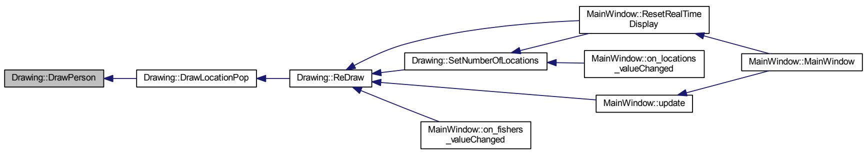


Here is the caller graph for this function:


Drawing::DrawLocationPop
Drawing::ReDraw
MainWindow::ResetRealTime Display
Drawing::SetNumberOfLocations
MainWindow::on_locations _valueChanged
MainWindow::MainWindow
MainWindow::update
MainWindow::on_fishers _valueChanged

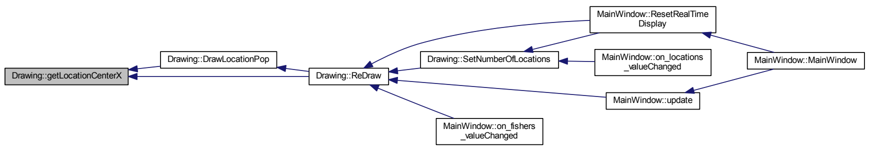
7.3.3.2 void Drawing::DrawPerson (int *x*, int *y*)

Here is the caller graph for this function:


Drawing::DrawPerson
Drawing::DrawLocationPop
Drawing::ReDraw
MainWindow::ResetRealTime Display
Drawing::SetNumberOfLocations
MainWindow::on_locations _valueChanged
MainWindow::MainWindow
MainWindow::update
MainWindow::on_fishers _valueChanged

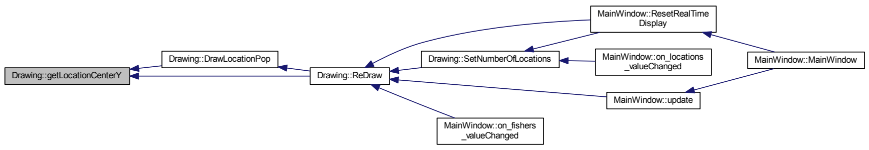
7.3.3.3 double Drawing::getLocationCenterX (int *location*)

Here is the caller graph for this function:


Drawing::getLocationCenterX
Drawing::DrawLocationPop
Drawing::ReDraw
Drawing::SetNumberOfLocations
MainWindow::ResetRealTime Display
MainWindow::on_locations _valueChanged
MainWindow::MainWindow
MainWindow::update
MainWindow::on_fishers _valueChanged

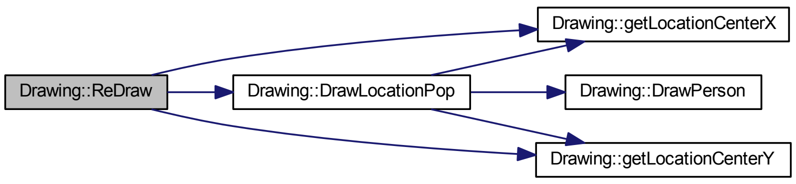
7.3.3.4 double Drawing::getLocationCenterY (int *location*)

Here is the caller graph for this function:

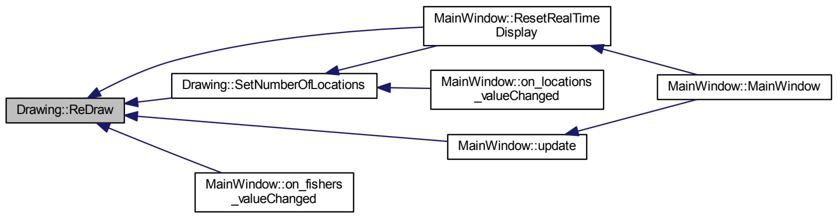

Drawing::getLocationCenterY
Drawing::DrawLocationPop
Drawing::ReDraw
Drawing::SetNumberOfLocations
MainWindow::ResetRealTime Display
MainWindow::on_locations _valueChanged
MainWindow::MainWindow
MainWindow::update
MainWindow::on_fishers _valueChanged

7.3.3.5 void Drawing::ReDraw ( )

Here is the call graph for this function:

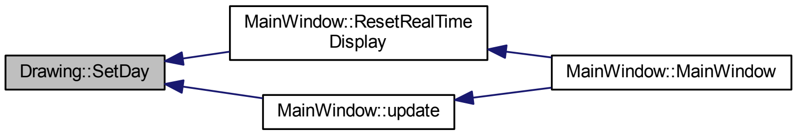


Here is the caller graph for this function:


MainWindow::ResetRealTime Display
Drawing::SetNumberOfLocations
MainWindow::on_locations _valueChanged
MainWindow::MainWindow
Drawing::ReDraw
MainWindow::update
MainWindow::on_fishers _valueChanged

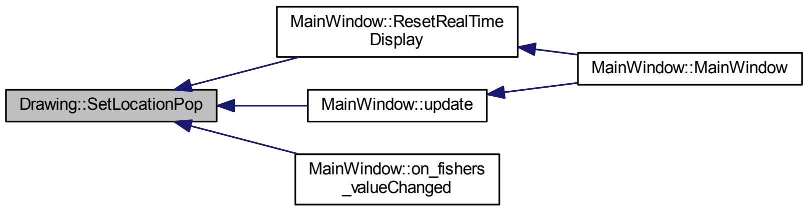
7.3.3.6 void Drawing::SetDay (int *day*)

Here is the caller graph for this function:



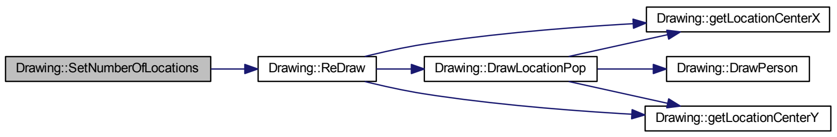
7.3.3.7 void Drawing::SetLocationPop (int *location,* int *pop*)

Here is the caller graph for this function:

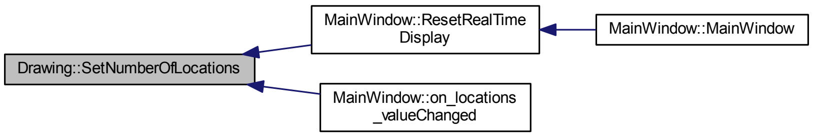

MainWindow::ResetRealTime Display
MainWindow::MainWindow
Drawing::SetLocationPop
MainWindow::update
MainWindow::on_fishers _valueChanged

7.3.3.8 void Drawing::SetNumberOfLocations (int *number*)

Here is the call graph for this function:


Drawing::SetNumberOfLocations
Drawing::ReDraw
Drawing::DrawLocationPop
Drawing::getLocationCenterX
Drawing::DrawPerson
Drawing::getLocationCenterY

Here is the caller graph for this function:


Drawing::SetNumberOfLocations
MainWindow::ResetRealTime Display
MainWindow::MainWindow
MainWindow::on_locations _valueChanged

7.3.4 Member Data Documentation

7.3.4.1 int Drawing::CurrentDay

7.3.4.2 int Drawing::locationPop[10]

7.3.4.3 int Drawing::NumberOfLocations

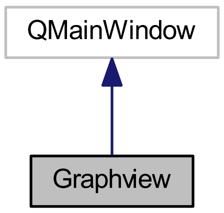
7.3.4.4 QGraphicsScene∗ Drawing::scene

7.3.4.5 QGraphicsView∗ Drawing::view

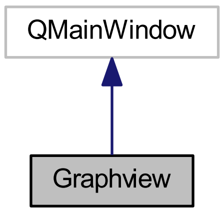
7.4 Graphview Class Reference

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

Inheritance diagram for Graphview:



Collaboration diagram for Graphview:



Public Member Functions

- Graphview (QWidget ∗parent=0)

  constructor for the Graphview class
- ∼Graphview ()
- void setupPlot ()

  setupPlot
- void printSettings (QString s)

Private Slots

- void on_actionInsert_Plot_triggered ()

  Graphview::on_actionInsert_Plot_triggered.
- void on_actionSave_Document_triggered ()

Private Attributes

- Ui::Graphview ∗ ui

7.4.1 Detailed Description

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

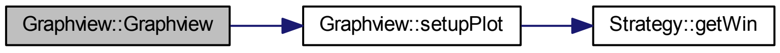
Graphview is intended to be used after the simulation has finished. It will accept data from the simulation module defining plots and display them to the users. There is also a report view on the left side that allows users to insert selected graphs to compile a final report.

7.4.2 Constructor & Destructor Documentation

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

constructor for the Graphview class

Here is the call graph for this function:



7.4.2.2 Graphview::∼Graphview ( )

7.4.3 Member Function Documentation

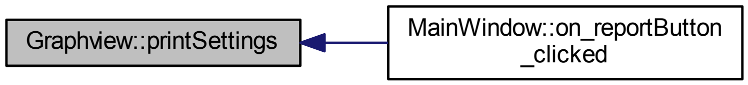
7.4.3.1 void Graphview::on_actionInsert_Plot_triggered ( ) `[private], [slot]`

Graphview::on_actionInsert_Plot_triggered.

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

7.4.3.3 void Graphview::printSettings (QString *s*)

Here is the caller graph for this function:

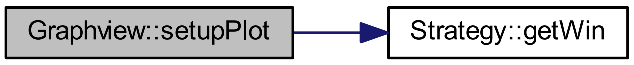


7.4.3.4 void Graphview::setupPlot ( )

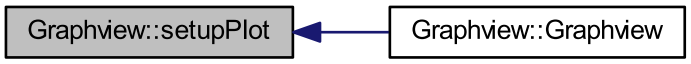
setupPlot

configures the plots

Here is the call graph for this function:



Here is the caller graph for this function:

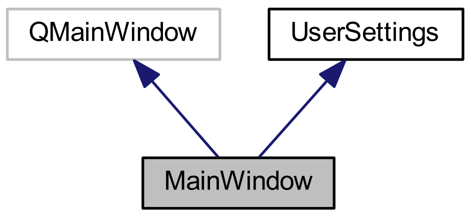


7.4.4 Member Data Documentation

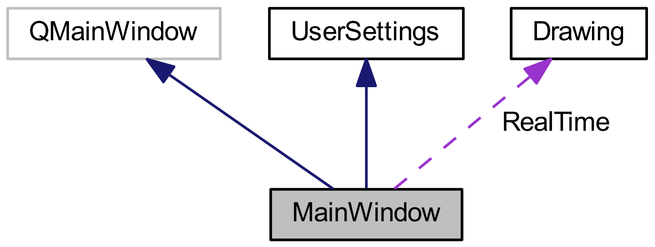
7.4.4.1 Ui::Graphview∗ Graphview::ui [private]

7.5 MainWindow Class Reference

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.

Protected Attributes

- Ui::MainWindow ∗ ui
- QString settings
- bool simulated = false
- QTimer ∗ timer

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 ()
- void startSimulate (int fisherNum, int fishLoc, int fishType, int fishPop, int fishTemp, int runtime)
- void calculateSpot (int fisherNum, int fishLoc)
- void update ()
- void ResetRealTimeDisplay (void)

Private Attributes

- QGraphicsScene $*$ drawing_scene
- Drawing $*$ RealTime

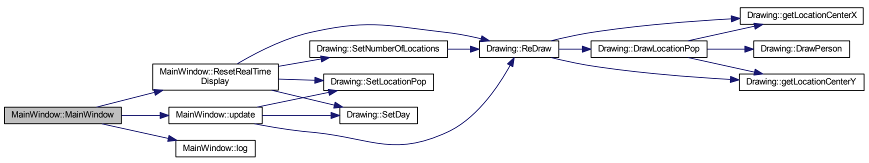
Friends

- class Graphview
- double getSpotppp ()

7.5.1 Constructor & Destructor Documentation

7.5.1.1 MainWindow::MainWindow (QWidget $*$ *parent* = 0) `[explicit]`

Here is the call graph for this function:

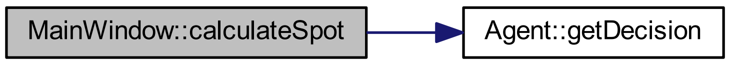


7.5.1.2 MainWindow::∼MainWindow ( )

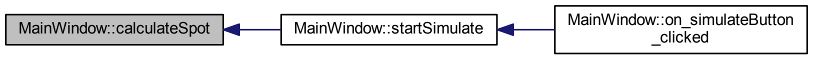
7.5.2 Member Function Documentation

7.5.2.1 void MainWindow::calculateSpot (int *fisherNum*, int *fishLoc*) [private], [slot]

Here is the call graph for this function:



Here is the caller graph for this function:


MainWindow::calculateSpot
MainWindow::startSimulate
MainWindow::on_simulateButton _clicked

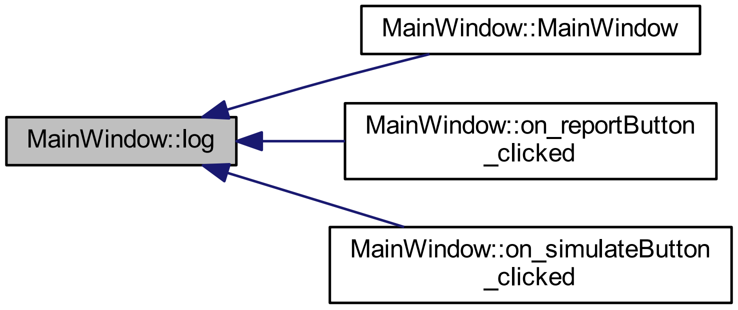
7.5.2.2 void MainWindow::log (const QString & *text*)

Sends a string to the simulation log.

Parameters

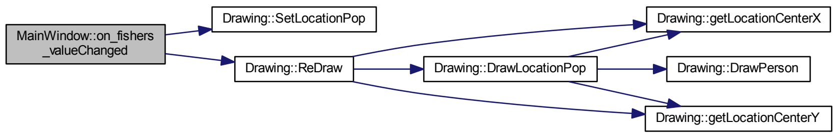
text	to display in the log.

Here is the caller graph for this function:


MainWindow::log
MainWindow::MainWindow
MainWindow::on_reportButton_clicked
MainWindow::on_simulateButton_clicked

7.5.2.3 void MainWindow::on_fishers_valueChanged (int *value*) [private], [slot]

Here is the call graph for this function:


MainWindow::on_fishers_valueChanged
Drawing::SetLocationPop
Drawing::ReDraw
Drawing::DrawLocationPop
Drawing::getLocationCenterX
Drawing::DrawPerson
Drawing::getLocationCenterY

7.5.2.4 void MainWindow::on_fishpop_valueChanged (int *value*) [private], [slot]

7.5.2.5 void MainWindow::on_fishtypes_valueChanged (int *value*) [private], [slot]

7.5.2.6 void MainWindow::on_lineEdit_0_textEdited (const QString & *arg1*) [private], [slot]

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

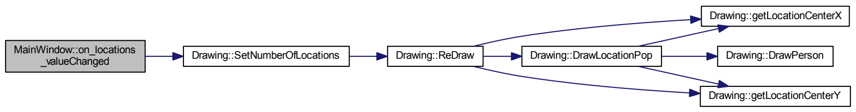
7.5.2.8 void MainWindow::on_lineEdit_2_textEdited (const QString & *arg1*) [private], [slot]

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

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

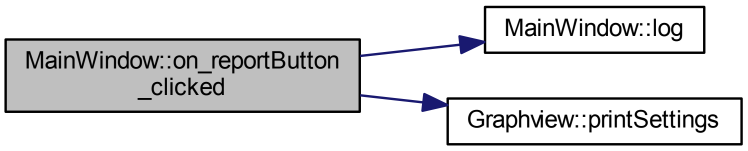
7.5.2.11 void MainWindow::on_locations_valueChanged (int *value*) [private], [slot]

Here is the call graph for this function:


MainWindow::on_locations_valueChanged
Drawing::SetNumberOfLocations
Drawing::ReDraw
Drawing::DrawLocationPop
Drawing::getLocationCenterX
Drawing::DrawPerson
Drawing::getLocationCenterY

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

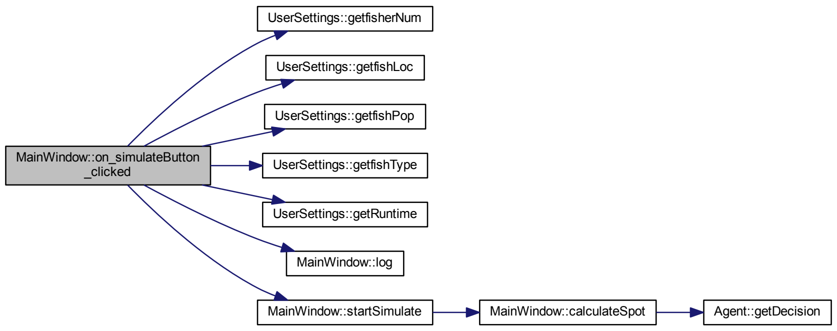
Here is the call graph for this function:



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

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

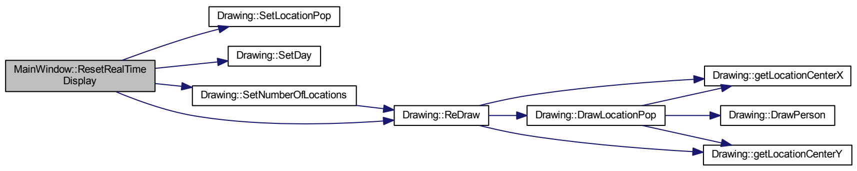
Here is the call graph for this function:


MainWindow::on_simulateButton_clicked
UserSettings::getfisherNum
UserSettings::getfishLoc
UserSettings::getfishPop
UserSettings::getfishType
UserSettings::getRuntime
MainWindow::log
MainWindow::startSimulate
MainWindow::calculateSpot
Agent::getDecision

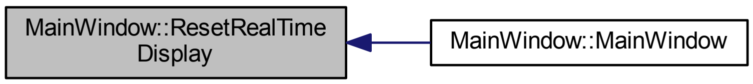
7.5.2.15 void MainWindow::on_weather_clicked ( ) [private], [slot]

7.5.2.16 void MainWindow::ResetRealTimeDisplay (void) [private], [slot]

Here is the call graph for this function:

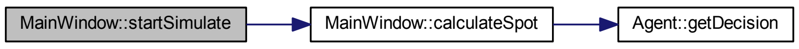

MainWindow::ResetRealTime Display
Drawing::SetLocationPop
Drawing::SetDay
Drawing::SetNumberOfLocations
Drawing::ReDraw
Drawing::DrawLocationPop
Drawing::getLocationCenterX
Drawing::DrawPerson
Drawing::getLocationCenterY

Here is the caller graph for this function:

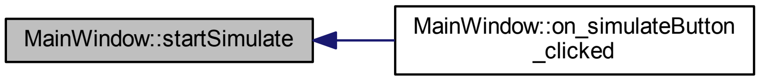


7.5.2.17 void MainWindow::startSimulate (int *fisherNum,* int *fishLoc,* int *fishType,* int *fishPop,* int *fishTemp,* int *runtime*) [private], [slot]

Here is the call graph for this function:

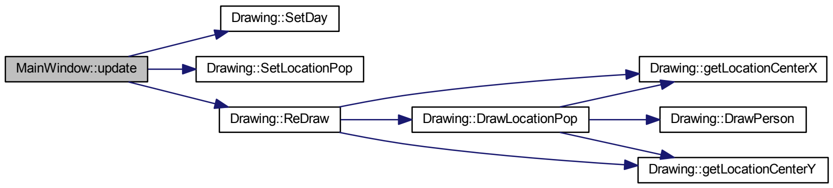


Here is the caller graph for this function:

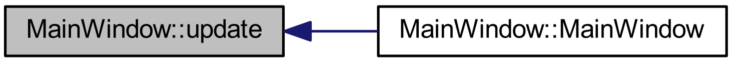

MainWindow::startSimulate
MainWindow::on_simulateButton _clicked

7.5.2.18 void MainWindow::update ( ) [private], [slot]

Here is the call graph for this function:



Here is the caller graph for this function:



7.5.3 Friends And Related Function Documentation

7.5.3.1 double getSpotppp ( ) [friend]

7.5.3.2 friend class Graphview [friend]

7.5.4 Member Data Documentation

7.5.4.1 QGraphicsScene∗ MainWindow::drawing_scene [private]

7.5.4.2 Drawing∗ MainWindow::RealTime [private]

7.5.4.3 QString MainWindow::settings [protected]

7.5.4.4 bool MainWindow::simulated = false [protected]

7.5.4.5 QTimer∗ MainWindow::timer [protected]

7.5.4.6 Ui::MainWindow∗ MainWindow::ui [protected]

7.6 Spot Class Reference

Spot is used to create a location and calculate how crowded it is.

Public Member Functions

- Spot (list< Agent ∗ > newAgents)
- void setCap (double cap)
- double getSpotCapacity ()
- void setAgentNum (int fisherNum)
- int getAgentNum ()
- list< Agent ∗ > getAgents ()

Private Attributes

- double maxcapacity

  max agents per spot
- int numAgent

  number of agents possibly going fishing per spo
- list< Agent ∗ > agents

  list of agents

7.6.1 Detailed Description

Spot is used to create a location and calculate how crowded it is.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 Spot::Spot (list< Agent ∗ > *newAgents*)

Constructor for a spot

Precondition

A list of agents

Postcondition

numAgent and maxcapacity is initialized to zero, sets a new list of agents

Returns

none

7.6.3 Member Function Documentation

7.6.3.1 int Spot::getAgentNum ( )

Get back total number agents

Precondition

Number of agents is already set

Postcondition

none

Returns

Integer number of total agents (numAgent)

7.6.3.2 list< Agent * > Spot::getAgents ( )

Precondition

A Spot wa initialized

Postcondition

Creates a list of agents

Returns

List of agents

7.6.3.3 double Spot::getSpotCapacity ( )

Get the maxcapacity

Precondition

A capacity is already set

Postcondition

none

Returns

The number of maxcapacity

7.6.3.4 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

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

7.6.4 Member Data Documentation

7.6.4.1 list< Agent *> Spot::agents [private]

list of agents

7.6.4.2 double Spot::maxcapacity [private]

max agents per spot

7.6.4.3 int Spot::numAgent [private]

number of agents possibly going fishing per spo

7.7 Strategy Class Reference

Public Member Functions

- Strategy (vector< int > randDecision)
- void updateScore (int point)

  records the secess of this strategy
- vector< int > getDecisionPattern ()
- int getScore ()

  returns the score This value represents the number of wins that an agent has made using this strategy
- int getWin ()
- int getLose ()
- void clearWin ()
- void clearLose ()

Private Attributes

- int score
- int win
- int lose
- vector< int > decisionPattern

  brief represents a strategy for determining the conditions of going fishing.

7.7.1 Constructor & Destructor Documentation

7.7.1.1 Strategy::Strategy (vector< int > *randDecision*)

7.7.2 Member Function Documentation

7.7.2.1 void Strategy::clearLose ( )

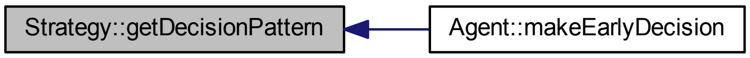
7.7.2.2 void Strategy::clearWin ( )

7.7.2.3 vector< int > Strategy::getDecisionPattern ( )

Returns the decision pattern used by this strategy

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

Here is the caller graph for this function:



7.7.2.4 int Strategy::getLose ( )

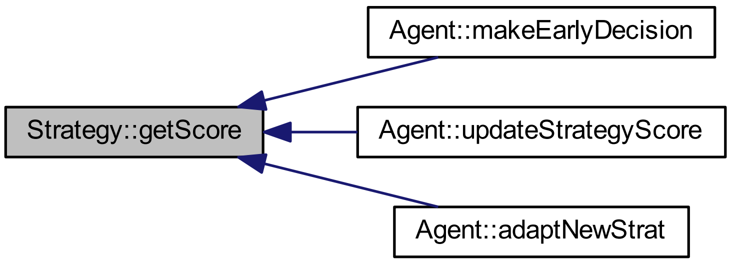
7.7.2.5 int Strategy::getScore ( )

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

Returns

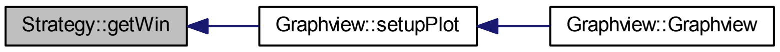
the strategy score

Here is the caller graph for this function:



7.7.2.6 int Strategy::getWin ( )

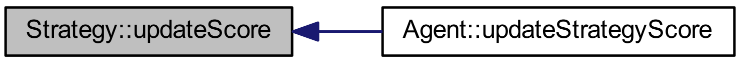
Here is the caller graph for this function:



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

records the secess of this strategy

Here is the caller graph for this function:



7.7.3 Member Data Documentation

7.7.3.1 vector<int> Strategy::decisionPattern [private]

brief represents a strategy for determining the conditions of going fishing.

7.7.3.2 int Strategy::lose [private]

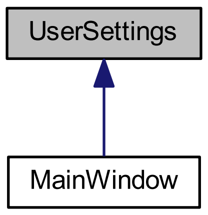
7.7.3.3 int Strategy::score [private]

7.7.3.4 int Strategy::win [private]

7.8 UserSettings Class Reference

contains the users global simulation parameters.

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

7.8.1 Detailed Description

contains the users global simulation parameters.

7.8.2 Constructor & Destructor Documentation

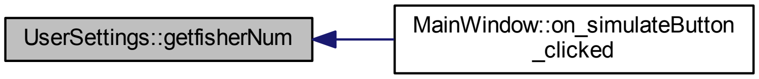
7.8.2.1 UserSettings::UserSettings ( )

7.8.3 Member Function Documentation

7.8.3.1 int UserSettings::getfisherNum ( )

Returns the number of Fishers to use in the simulation

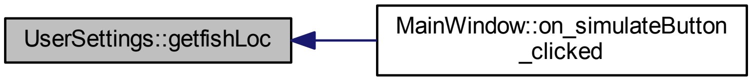
Here is the caller graph for this function:


UserSettings::getfisherNum
MainWindow::on_simulateButton _clicked

7.8.3.2 int UserSettings::getfishLoc ( )

Returns the number of different locations

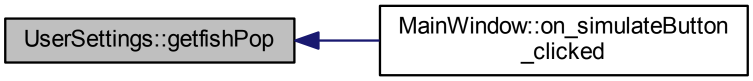
Here is the caller graph for this function:



7.8.3.3 int UserSettings::getfishPop ( )

Returns the inital population of fish when the simulation starts.

Here is the caller graph for this function:



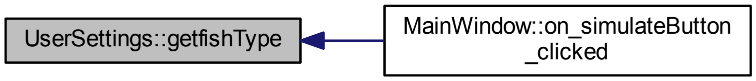
7.8.3.4 int UserSettings::getfishTemp ( )

Returns the conditions: overcast, snow, rain.

7.8.3.5 int UserSettings::getfishType ( )

Returns the number of fish types.

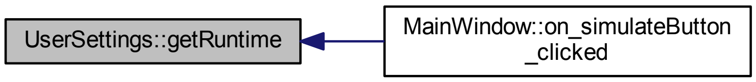
Here is the caller graph for this function:



7.8.3.6 int UserSettings::getRuntime ( )

Returns the number of days to run the simulation.

Here is the caller graph for this function:



7.8.4 Member Data Documentation

7.8.4.1 int UserSettings::fisherNum [protected]

The number of Fishers to use in the simulation

7.8.4.2 int UserSettings::fishLoc [protected]

The number of different locations

7.8.4.3 int UserSettings::fishPop [protected]

The inital population of fish when the simulation starts.

7.8.4.4 int UserSettings::fishTemp [protected]

The conditions: overcast, snow, rain

7.8.4.5 int UserSettings::fishType `[protected]`

The number of fish types.

7.8.4.6 int UserSettings::runtime `[protected]`

The number of days to run the simulation

Index