Fisher sim

0.0.1

Generated by Doxygen 1.8.9.1

Sun Mar 29 2015 17:55:12

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

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

Class Documentation

4.1 Agent Class Reference

Public Member Functions

- **Agent** (vector$<$ Strategy $*$ $>$ strat)
- void updateStrategyScore (int winnigScore)

    constructor
- void **calcThreshold** ()
- void **makeEarlyDecision** ()
- void **makeDecision** ()
- void updateHistory ()

    will be based on earlydecision and threshold
- void setTemp (float newTemp)

    push new decision on
- void setSkill (int newskill)

    from input
- void setFishduration (float newFishDuration)

    can be randomize
- void setCommunication (int newCommunication)

    can be randomize
- vector$<$ int $>$ **getHistory** ()
- int getDecision ()

    Returns the Decision of the Agent.
- int getCommunication ()

    Returns the amount the agent communicates with other agents.
- int getSkill ()

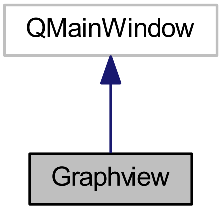
    Returns the current skill of the agent.
- float **getTemp** ()
- float **getFishDuration** ()
- int **getEarlyDecision** ()
- float **getThreshold** ()
- vector$<$ Strategy $*$ $>$ **getStrat** ()

4.1.1 Detailed Description

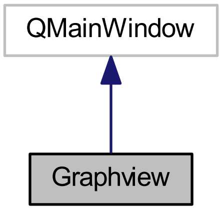
records the total number of agents created. influence threshold, if based on report $>$ 70 will make agent's decison change to 1 $<$ 70 agent's decision remain the same new rule: p =$>$ 85 change decision to 1 40 $<$ p $<$ 85 decision remain p $<$= 40 decision change to -1

4.2 Graphview Class Reference

Inheritance diagram for Graphview:



Collaboration diagram for Graphview:



Public Member Functions

- Graphview (QWidget ∗parent=0)

  constructor for the Graphview class

- void setupPlot ()

  setupPlot

4.2.1 Detailed Description

provides a view that shows the colected 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 deffineing 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.

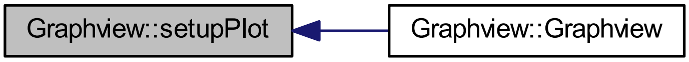
4.2.2 Member Function Documentation

4.2.2.1 void Graphview::setupPlot ( )

setupPlot

configures the plots

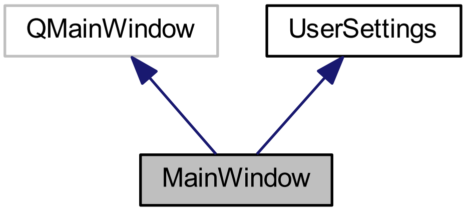
Here is the caller graph for this function:



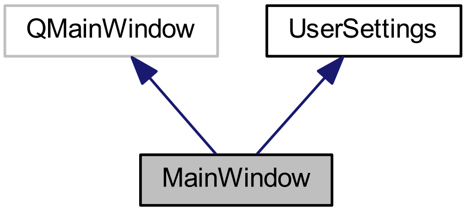
4.3 MainWindow Class Reference

The MainWindow class Provides the Main windows for the Fisher sim project.

Inheritance diagram for MainWindow:



Collaboration diagram for MainWindow:



Public Member Functions

- **MainWindow** (QWidget ∗parent=0)
- void log (const QString &text)

  Sends a string to the simulation log.

Additional Inherited Members

4.3.1 Detailed Description

The MainWindow class Provides the Main windows for the Fisher sim project.

4.3.2 Member Function Documentation

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

Sends a string to the simulation log.

Parameters

text	to display in the log.

4.4 Spot Class Reference

Public Member Functions

- void **setCap** (double cap)
- double **getSpotCapacity** ()
- void **setAgentNum** (int fisherNum)
- int **getAgentNum** ()
- double **crowdness** (double goFish)

4.5 Strategy Class Reference

Public Member Functions

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

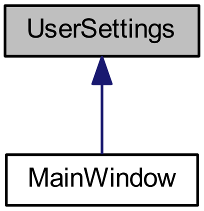
4.5.1 Detailed Description

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

4.6 UserSettings Class Reference

Records the global simulation settings.

Inheritance diagram for UserSettings:



Public Member Functions

- 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

4.6.1 Detailed Description

Records the global simulation settings.

4.6.2 Member Function Documentation

4.6.2.1 int UserSettings::getfishLoc ( )

Returns the number of Fishers to use in the simulation

4.6.2.2 int UserSettings::getfishPop ( )

Returns the number of fish types.

4.6.2.3 int UserSettings::getfishTemp ( )

Returns the inital population of fish when the simulation starts.

4.6.2.4 int UserSettings::getfishType ( )

Returns the number of different locations

4.6.2.5 int UserSettings::getRuntime ( )

Returns the conditions: overcast, snow, rain.

4.6.3 Member Data Documentation

4.6.3.1 int UserSettings::fisherNum [protected]

Returns the number of days to run the simulation.

4.6.3.2 int UserSettings::fishLoc [protected]

The number of Fishers to use in the simulation

4.6.3.3 int UserSettings::fishPop [protected]

The number of fish types.

4.6.3.4 int UserSettings::fishTemp [protected]

The inital population of fish when the simulation starts.

4.6.3.5 int UserSettings::fishType [protected]

The number of different locations

4.6.3.6 int UserSettings::runtime [protected]

The conditions: overcast, snow, rain

Index