Graphing Calculator

Generated by Doxygen 1.10.0

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Class Documentation

3.1 Application Class Reference

Public Member Functions

- const bool windowIsOpen () const
- void pollEvents ()
- void update ()
- void render ()

3.1.1 Detailed Description

Definition at line 33 of file application.h.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 Application()

```
Application::Application ( )
```

Definition at line 64 of file application.cpp.

```
00064                           {
00065     this->initializeVariables();
00066     this->initializeWindow();
00067 }
```

3.1.2.2 ~Application()

```
Application::~Application ( )  [virtual]
```

Definition at line 69 of file application.cpp.

```
00069                            {
00070     delete this->window;
00071 }
```

3.1.3 Member Function Documentation

3.1.3.1 pollEvents()

```
void Application::pollEvents ( )
```

Definition at line 81 of file application.cpp.

```
00081                               {
00082     while (this->window->pollEvent(this->event)) {
00083         switch (this->event.type) {
00084         case sf::Event::Closed:
00085             this->window->close();
00086             break;
00087         case sf::Event::KeyPressed:
00088             switch (this->event.key.code) {
00089                 case sf::Keyboard::Escape:
00090                     this->window->close();
00091                     break;
00092                 case sf::Keyboard::A:
00093                     this->graphMode = 1;
00094                     break;
00095                 case sf::Keyboard::B:
00096                     this->graphMode = 2;
00097                     break;
00098                 case sf::Keyboard::C:
00099                     this->graphMode = 3;
00100                    break;
00101                case sf::Keyboard::D:
00102                    this->graphMode = 4;
00103                    break;
00104                case sf::Keyboard::E:
00105                    this->graphMode = 5;
00106                    break;
00107                case sf::Keyboard::F:
00108                    this->graphMode = 6;
00109                    break;
00110                case sf::Keyboard::G:
00111                    this->graphMode = 7;
00112                    break;
00113                case sf::Keyboard::Right:
00114                    this->xMin += 0.1;
00115                    this->xMax += 0.1;
00116                    break;
00117                case sf::Keyboard::Left:
00118                    this->xMin -= 0.1;
00119                    this->xMax -= 0.1;
00120                    break;
00121                case sf::Keyboard::Up:
00122                    this->yMin += 0.1;
00123                    this->yMax += 0.1;
00124                    break;
00125                case sf::Keyboard::Down:
00126                    this->yMin -= 0.1;
00127                    this->yMax -= 0.1;
00128                    break;
00129            }
00130            break;
00131        }
00132    }
00133 }
```

3.1.3.2 render()

```
void Application::render ( )
```

Definition at line 181 of file application.cpp.

```
00182 {
00183     this->window->clear();
00184
00185     for (int i=0; i<this->gridRows; i++) {
00186         for (int j=0; j<this->gridCols; j++ ) {
00187             this->window->draw(this->gridVector[i*gridRows + j]);
00188         }
00189     }
00190 //    for (int i=0; i<this->gridRows*this->gridCols; i++) {
00191 //        this->window->draw(this->gridVector[i]);
00192 //    }
```

```
00193 //    for (auto v : this->gridVector) {
00194 //        this->window->draw(v);
00195 //    }
00196
00197     this->window->display();
00198 }
```

3.1.3.3 update()

```
void Application::update ( )
```

Definition at line 135 of file application.cpp.

```
00135                           {
00136     this->pollEvents();
00137 //    int k = std::rand() % (this->gridRows * this->gridCols);
00138 //    this->gridVector[k].setFillColor(sf::Color::Red);
00139     for (int i=0; i<this->gridRows; i++) {
00140         for (int j=0; j<this->gridCols; j++) {
00141             sf::Vector2f euclidean = this->screenToEuclidean(
00142                 this->gridToScreen( sf::Vector2u(j,i) )
00143             );
00144             float fx;
00145             switch (this->graphMode) {
00146                 case 1:
00147                     fx = funcA(euclidean.x);
00148                     break;
00149                 case 2:
00150                     fx = funcB(euclidean.x);
00151                     break;
00152                 case 3:
00153                     fx = funcC(euclidean.x);
00154                     break;
00155                 case 4:
00156                     fx = funcD(euclidean.x);
00157                     break;
00158                 case 5:
00159                     fx = funcE(euclidean.x);
00160                     break;
00161                 case 6:
00162                     fx = funcF(euclidean.x);
00163                     break;
00164                 case 7:
00165                     fx = funcG(euclidean.x);
00166                     break;
00167             }
00168             if (std::abs( euclidean.y - fx ) < 0.01) {
00169                 this->gridVector[i*gridRows + j].setFillColor(sf::Color::Red);
00170             } else if ( std::abs( euclidean.y ) < 0.01) {
00171                 this->gridVector[i*gridRows + j].setFillColor(sf::Color::Black);
00172             } else if ( std::abs( euclidean.x ) < 0.01) {
00173                 this->gridVector[i*gridRows + j].setFillColor(sf::Color::Black);
00174             } else {
00175                 this->gridVector[i*gridRows + j].setFillColor(sf::Color::White);
00176             }
00177         }
00178     }
00179 }
```

3.1.3.4 windowIsOpen()

```
const bool Application::windowIsOpen ( ) const
```

Definition at line 75 of file application.cpp.

```
00075                                       {
00076     return this->window->isOpen();
00077 }
```

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

- /home/jakemath/Desktop/code/SFML/GraphingApp/code/src/application.h
- /home/jakemath/Desktop/code/SFML/GraphingApp/code/src/application.cpp

Chapter 4

File Documentation

4.1 application.cpp

```
00001 #include "application.h"
00002 #include "test_funcs.h"
00003
00004 // Private functions
00005
00006 void Application::initializeVariables() {
00007     this->window = nullptr;
00008     this->gridTile.setPosition(sf::Vector2f(W_MIN,H_MIN));
00009     this->gridTile.setSize(sf::Vector2f(TILE_WIDTH, TILE_HEIGHT));
00010     this->gridRows = (H_MAX - H_MIN) / TILE_HEIGHT;
00011     this->gridCols = (W_MAX - W_MIN) / TILE_WIDTH;
00012
00013     this->graphMode = 1;
00014
00015     this->xMin = DEFAULT_X_MIN;
00016     this->xMax = DEFAULT_X_MAX;
00017     this->yMin = DEFAULT_Y_MIN;
00018     this->yMax = DEFAULT_Y_MAX;
00019
00020
00021     for (int i=0; i<this->gridRows; i++) {
00022         for (int j=0; j<this->gridCols; j++) {
00023             this->gridTile.setPosition(sf::Vector2f(
00024                 W_MIN + TILE_WIDTH * j,
00025                 H_MIN + TILE_HEIGHT * i
00026             ));
00027             this->gridVector.push_back(this->gridTile);
00028         }
00029     }
00030 }
00031
00032 void Application::initializeWindow() {
00033     this->videoMode.width = WINDOW_WIDTH;
00034     this->videoMode.height = WINDOW_HEIGHT;
00035     this->window = new sf::RenderWindow(this->videoMode, "Application", sf::Style::None);
00036                                         //  sf::Style::Titlebar | sf::Style::Close);
00037     this->window->setPosition(sf::Vector2i(0, 0));
00038     this->window->setFramerateLimit(60);
00039 }
00040
00041 sf::Vector2f Application::gridToScreen(sf::Vector2u gpos) {
00042     sf::Vector2f spos;
00043     spos.x = W_MIN + (gpos.x) * (W_MAX - W_MIN) / (this->gridCols);
00044     spos.y = H_MIN + (gpos.y) * (H_MAX - H_MIN) / (this->gridRows);
00045     return spos;
00046 }
00047
00048 sf::Vector2f Application::screenToEuclidean(sf::Vector2f spos) {
00049     sf::Vector2f epos;
00050     epos.x = this->xMin + (spos.x - W_MIN) * (this->xMax - this->xMin) / (W_MAX - W_MIN);
00051     epos.y = this->yMax - (spos.y - H_MIN) * (this->yMax - this->yMin) / (H_MAX - H_MIN);
00052     return epos;
00053 }
00054
00055 sf::Vector2f Application::euclideanToScreen(sf::Vector2f epos) {
00056     sf::Vector2f spos;
00057     spos.x = W_MIN + (epos.x - this->xMin) * (W_MAX - W_MIN) / (this->xMax - this->xMin);
00058     spos.y = H_MIN + (this->yMax - epos.y) * (H_MAX - H_MIN) / (this->yMax - this->yMin);
```

```
00059     return spos;
00060 }
00061
00062 // Constructors / Destructors
00063
00064 Application::Application() {
00065     this->initializeVariables();
00066     this->initializeWindow();
00067 }
00068
00069 Application::~Application() {
00070     delete this->window;
00071 }
00072
00073 // Accessors
00074
00075 const bool Application::windowIsOpen() const {
00076     return this->window->isOpen();
00077 }
00078
00079 // Functions
00080
00081 void Application::pollEvents() {
00082     while (this->window->pollEvent(this->event)) {
00083         switch (this->event.type) {
00084         case sf::Event::Closed:
00085             this->window->close();
00086             break;
00087         case sf::Event::KeyPressed:
00088             switch (this->event.key.code) {
00089                 case sf::Keyboard::Escape:
00090                     this->window->close();
00091                     break;
00092                 case sf::Keyboard::A:
00093                     this->graphMode = 1;
00094                     break;
00095                 case sf::Keyboard::B:
00096                     this->graphMode = 2;
00097                     break;
00098                 case sf::Keyboard::C:
00099                     this->graphMode = 3;
00100                     break;
00101                 case sf::Keyboard::D:
00102                     this->graphMode = 4;
00103                     break;
00104                 case sf::Keyboard::E:
00105                     this->graphMode = 5;
00106                     break;
00107                 case sf::Keyboard::F:
00108                     this->graphMode = 6;
00109                     break;
00110                 case sf::Keyboard::G:
00111                     this->graphMode = 7;
00112                     break;
00113                 case sf::Keyboard::Right:
00114                     this->xMin += 0.1;
00115                     this->xMax += 0.1;
00116                     break;
00117                 case sf::Keyboard::Left:
00118                     this->xMin -= 0.1;
00119                     this->xMax -= 0.1;
00120                     break;
00121                 case sf::Keyboard::Up:
00122                     this->yMin += 0.1;
00123                     this->yMax += 0.1;
00124                     break;
00125                 case sf::Keyboard::Down:
00126                     this->yMin -= 0.1;
00127                     this->yMax -= 0.1;
00128                     break;
00129             }
00130             break;
00131         }
00132     }
00133 }
00134
00135 void Application::update() {
00136     this->pollEvents();
00137 //    int k = std::rand() % (this->gridRows * this->gridCols);
00138 //    this->gridVector[k].setFillColor(sf::Color::Red);
00139     for (int i=0; i<this->gridRows; i++) {
00140         for (int j=0; j<this->gridCols; j++) {
00141             sf::Vector2f euclidean = this->screenToEuclidean(
00142                 this->gridToScreen( sf::Vector2u(j,i) )
00143             );
00144             float fx;
00145             switch (this->graphMode) {
```

```
00146                 case 1:
00147                     fx = funcA(euclidean.x);
00148                     break;
00149                 case 2:
00150                     fx = funcB(euclidean.x);
00151                     break;
00152                 case 3:
00153                     fx = funcC(euclidean.x);
00154                     break;
00155                 case 4:
00156                     fx = funcD(euclidean.x);
00157                     break;
00158                 case 5:
00159                     fx = funcE(euclidean.x);
00160                     break;
00161                 case 6:
00162                     fx = funcF(euclidean.x);
00163                     break;
00164                 case 7:
00165                     fx = funcG(euclidean.x);
00166                     break;
00167             }
00168             if (std::abs( euclidean.y - fx ) < 0.01) {
00169                 this->gridVector[i*gridRows + j].setFillColor(sf::Color::Red);
00170             } else if ( std::abs( euclidean.y ) < 0.01) {
00171                 this->gridVector[i*gridRows + j].setFillColor(sf::Color::Black);
00172             } else if ( std::abs( euclidean.x ) < 0.01) {
00173                 this->gridVector[i*gridRows + j].setFillColor(sf::Color::Black);
00174             } else {
00175                 this->gridVector[i*gridRows + j].setFillColor(sf::Color::White);
00176             }
00177         }
00178     }
00179 }
00180
00181 void Application::render()
00182 {
00183     this->window->clear();
00184
00185     for (int i=0; i<this->gridRows; i++) {
00186         for (int j=0; j<this->gridCols; j++ ) {
00187             this->window->draw(this->gridVector[i*gridRows + j]);
00188         }
00189     }
00190 //    for (int i=0; i<this->gridRows*this->gridCols; i++) {
00191 //        this->window->draw(this->gridVector[i]);
00192 //    }
00193 //    for (auto v : this->gridVector) {
00194 //        this->window->draw(v);
00195 //    }
00196
00197     this->window->display();
00198 }
00199
00200
```

4.2 application.h

```
00001 #pragma once
00002
00003 /*
00004     Manages the application window
00005 */
00006
00007 #include <iostream>
00008 #include <vector>
00009 #include <ctime>
00010
00011 #include <SFML/Graphics.hpp>
00012 #include <SFML/System.hpp>
00013 #include <SFML/Window.hpp>
00014 #include <SFML/Audio.hpp>
00015 #include <SFML/Network.hpp>
00016
00017 const float WINDOW_WIDTH = 1920;
00018 const float WINDOW_HEIGHT = 1080;
00019
00020 const float DEFAULT_X_MIN = -2;
00021 const float DEFAULT_X_MAX = 2;
00022 const float DEFAULT_Y_MIN = -2;
00023 const float DEFAULT_Y_MAX = 2;
00024
00025 const float W_MIN = 560;
```

```
00026 const float W_MAX = 1360;
00027 const float H_MIN = 140;
00028 const float H_MAX = 940;
00029
00030 const float TILE_WIDTH = 3;
00031 const float TILE_HEIGHT = 3;
00032
00033 class Application {
00034 private:
00035     // Variables
00036     sf::RenderWindow* window;
00037     sf::VideoMode videoMode;
00038     sf::Event event;
00039
00040     sf::RectangleShape gridTile;
00041     std::vector<sf::RectangleShape> gridVector;
00042     int gridRows;
00043     int gridCols;
00044     int graphMode;
00045
00046     float xMin;
00047     float xMax;
00048     float yMin;
00049     float yMax;
00050
00051     // Private functions
00052     void initializeVariables();
00053     void initializeWindow();
00054
00055     sf::Vector2f gridToScreen(sf::Vector2u pos);
00056     sf::Vector2f screenToEuclidean(sf::Vector2f pos);
00057     sf::Vector2f euclideanToScreen(sf::Vector2f pos);
00058 public:
00059     // Constructors / Destructors
00060     Application();
00061     virtual ~Application();
00062
00063     // Accessors
00064     const bool windowIsOpen() const;
00065
00066     // Functions
00067     void pollEvents();
00068     void update();
00069     void render();
00070 };
```

4.3 main.cpp

```
00001 #include "application.h"
00002
00003 int main()
00004 {
00005     std::srand(std::time(nullptr));
00006     Application app;
00007     while (app.windowIsOpen()) {
00008         app.update();
00009         app.render();
00010     }
00011     return 0;
00012 }
```

4.4 test_funcs.cpp

```
00001 #include "test_funcs.h"
00002
00003 float funcA(float x) {
00004     return x;
00005 }
00006
00007 float funcB(float x) {
00008     return x * x;
00009 }
00010
00011 float funcC(float x) {
00012     return x * x * x;
00013 }
00014
00015 float funcD(float x) {
00016     return std::abs(x);
```

```
00017 }
00018
00019 float funcE(float x) {
00020     return std::sin(x);
00021 }
00022
00023 float funcF(float x) {
00024     return std::cos(x);
00025 }
00026
00027 float funcG(float x) {
00028     return std::tan(x);
00029 }
```

4.5 test_funcs.h

```
00001 #pragma once
00002
00003 #include "math.h"
00004
00005 float funcA(float);
00006 float funcB(float);
00007 float funcC(float);
00008 float funcD(float);
00009 float funcE(float);
00010 float funcF(float);
00011 float funcG(float);
```

Index