Visualgo CS162

1.0.0

Generated by Doxygen 1.9.6

Chapter 1

Visualgo_CS162

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 constants Namespace Reference

Namespaces

- namespace Arrow
- namespace ControlMenu
- namespace Highlighter
- namespace LinkedList
- namespace MenuArray
- namespace MenuDataStructure
- namespace MenuLinkedList
- namespace NodeInfo
- namespace sceneVariables
- namespace Square
- namespace TitleNode

Functions

- static sf::Color hoverGreen (162, 178, 159)
- static sf::Color clickGreen (121, 135, 119)
- static sf::Color transparentGreen (189, 210, 182, 150)
- static sf::Color hoverGray (150, 150, 150)
- static sf::Color clickGray (100, 100, 100)

Variables

- constexpr char titleWindow [ ] = "Visualgo CS162 - Phan Minh Quang"
- constexpr char fontPath [ ] = "../assets/fonts/Hack_reg.ttf"

6.1.1 Function Documentation

6.1.1.1 clickGray()

```
static sf::Color constants::clickGray (
             100 ,
             100 ,
             100  )
```

6.1.1.2 clickGreen()

```
static sf::Color constants::clickGreen (
             121 ,
             135 ,
             119  )
```

6.1.1.3 hoverGray()

```
static sf::Color constants::hoverGray (
             150 ,
             150 ,
             150  )
```

6.1.1.4 hoverGreen()

```
static sf::Color constants::hoverGreen (
             162 ,
             178 ,
             159  )
```

6.1.1.5 transparentGreen()

```
static sf::Color constants::transparentGreen (
             189 ,
             210 ,
             182 ,
             150  )
```

6.1.2 Variable Documentation

6.1.2.1 fontPath

```
constexpr char constants::fontPath[] = "../assets/fonts/Hack_reg.ttf"  [constexpr]
```

Definition at line 411 of file Constants.hpp.

6.1.2.2 titleWindow

```
constexpr char constants::titleWindow[] = "Visualgo CS162 - Phan Minh Quang"  [constexpr]
```

Definition at line 408 of file Constants.hpp.

6.2 constants::Arrow Namespace Reference

Functions

- static sf::Vector2i sizeRectangle (192, 37)
- static sf::Vector2f defaultScaleRectangle (0.6f, 0.16f)

6.2.1 Function Documentation

6.2.1.1 defaultScaleRectangle()

```
static sf::Vector2f constants::Arrow::defaultScaleRectangle (
            0.  6f,
            0.  16f )
```

6.2.1.2 sizeRectangle()

```
static sf::Vector2i constants::Arrow::sizeRectangle (
            192 ,
            37  )
```

6.3 constants::ControlMenu Namespace Reference

Enumerations

- enum class Button {
  PREVIOUS , PLAY , NEXT , SPEED_DOWN ,
  SPEED_UP , None }

Variables

- constexpr int BUTTON_COUNT = 5
- constexpr int BUTTON_NAME_SIZE = 15
- constexpr int TEXT_SIZE = 15
- constexpr char BUTTON_NAMES [BUTTON_COUNT][50]

6.3.1 Enumeration Type Documentation

6.3.1.1 Button

```
enum class constants::ControlMenu::Button  [strong]
```

Enumerator

PREVIOUS	
PLAY	
NEXT	
SPEED_DOWN	
SPEED_UP	
None	

Definition at line 336 of file Constants.hpp.

```
00336                          {
00337             PREVIOUS,
00338             PLAY,
00339             NEXT,
00340             SPEED_DOWN,
00341             SPEED_UP,
00342             None
00343         };
```

6.3.2 Variable Documentation

6.3.2.1 BUTTON_COUNT

```
constexpr int constants::ControlMenu::BUTTON_COUNT = 5  [constexpr]
```

Definition at line 345 of file Constants.hpp.

6.3.2.2 BUTTON_NAME_SIZE

```
constexpr int constants::ControlMenu::BUTTON_NAME_SIZE = 15
```

Definition at line 346 of file Constants.hpp.

6.3.2.3 BUTTON_NAMES

```
constexpr char constants::ControlMenu::BUTTON_NAMES[BUTTON_COUNT][50]  [constexpr]
```

Initial value:
```
= {
                "<",
                "[=]",
                ">",
                "«",
                "»"
        }
```

Definition at line 348 of file Constants.hpp.

6.3.2.4 TEXT_SIZE

```
constexpr int constants::ControlMenu::TEXT_SIZE = 15
```

Definition at line 347 of file Constants.hpp.

6.4 constants::Highlighter Namespace Reference

Namespaces

- namespace DLL
- namespace SLL

Functions

- static sf::Vector2f codeScale (0.6f, 0.6f)

6.4.1 Function Documentation

6.4.1.1 codeScale()

```
static sf::Vector2f constants::Highlighter::codeScale (
              0.  6f,
              0.  6f )
```

6.5 constants::Highlighter::DLL Namespace Reference

Variables

- const std::pair< const char *, const int > CODES_PATH [ ]

6.5.1 Variable Documentation

6.5.1.1 CODES_PATH

```
const std::pair<const char*, const int> constants::Highlighter::DLL::CODES_PATH[]
```

Initial value:

```
= {
                std::make_pair("../assets/code/DLL/add_beginning.png", 8),
                std::make_pair("../assets/code/DLL/add_ending.png", 5),
                std::make_pair("../assets/code/DLL/add_middle.png", 9),
                std::make_pair("../assets/code/DLL/delete_beginning.png", 8),
                std::make_pair("../assets/code/DLL/delete_ending.png", 5),
                std::make_pair("../assets/code/DLL/delete_middle.png", 7),
                std::make_pair("../assets/code/DLL/update.png", 4),
                std::make_pair("../assets/code/DLL/search.png", 6)
            }
```

Definition at line 383 of file Constants.hpp.

6.6 constants::Highlighter::SLL Namespace Reference

Variables

- const std::pair< const char *, const int > CODES_PATH [4]

6.6.1 Variable Documentation

6.6.1.1 CODES_PATH

```
const std::pair<const char*, const int> constants::Highlighter::SLL::CODES_PATH[4]
```

Initial value:

```
= {
                std::make_pair("../assets/code/SLL/add.png", 10),
                std::make_pair("../assets/code/SLL/delete.png", 11),
                std::make_pair("../assets/code/SLL/update.png", 4),
                std::make_pair("../assets/code/SLL/search.png", 6)
            }
```

Definition at line 374 of file Constants.hpp.

6.7 constants::LinkedList Namespace Reference

6.8 constants::MenuArray Namespace Reference

Namespaces

- namespace AddMode
- namespace AllocateMode
- namespace CreateMode
- namespace DeleteMode
- namespace SearchMode
- namespace UpdateMode

Enumerations

- enum class Type { DYNAMIC , STATIC }
- enum Button {
  CREATE_BUTTON , ADD_BUTTON , DELETE_BUTTON , UPDATE_BUTTON ,
  SEARCH_BUTTON , ALLOCATE_BUTTON , NONE }

Variables

- constexpr int BUTTON_COUNT = 6
- constexpr char BUTTON_NAMES [BUTTON_COUNT][50]
- constexpr int BUTTON_NAME_SIZE = 15

6.8.1 Enumeration Type Documentation

6.8.1.1 Button

```
enum constants::MenuArray::Button
```

Enumerator

CREATE_BUTTON	
ADD_BUTTON	
DELETE_BUTTON	
UPDATE_BUTTON	
SEARCH_BUTTON	
ALLOCATE_BUTTON	
NONE	

Definition at line 52 of file Constants.hpp.

```
                    {
            CREATE_BUTTON,
            ADD_BUTTON,
            DELETE_BUTTON,
            UPDATE_BUTTON,
            SEARCH_BUTTON,
            ALLOCATE_BUTTON,
            NONE
        };
```

6.8.1.2 Type

```
enum class constants::MenuArray::Type  [strong]
```

Enumerator

DYNAMIC	
STATIC	

Definition at line 46 of file Constants.hpp.

```
00046                {
00047           DYNAMIC,
00048           STATIC
00049       };
```

6.8.2 Variable Documentation

6.8.2.1 BUTTON_COUNT

```
constexpr int constants::MenuArray::BUTTON_COUNT = 6  [constexpr]
```

Definition at line 51 of file Constants.hpp.

6.8.2.2 BUTTON_NAME_SIZE

```
constexpr int constants::MenuArray::BUTTON_NAME_SIZE = 15  [constexpr]
```

Definition at line 69 of file Constants.hpp.

6.8.2.3 BUTTON_NAMES

```
constexpr char constants::MenuArray::BUTTON_NAMES[BUTTON_COUNT][50]  [constexpr]
```

Initial value:

```
= {
              "Create",
              "Add",
              "Delete",
              "Update",
              "Search",
              "Allocate"
       }
```

Definition at line 61 of file Constants.hpp.

6.9 constants::MenuArray::AddMode Namespace Reference

Enumerations

- enum Textbox { POSITION_TEXTBOX , VALUE_TEXTBOX , NONE }

Variables

- constexpr int TEXTBOX_COUNT = 2
- constexpr char TEXTBOX_NAMES [2][50]
- constexpr int TEXTBOX_LENGTH [2]

6.9.1 Enumeration Type Documentation

6.9.1.1 Textbox

```
enum constants::MenuArray::AddMode::Textbox
```

Enumerator

POSITION_TEXTBOX	
VALUE_TEXTBOX	
NONE	

Definition at line 107 of file Constants.hpp.

```
                        {
                    POSITION_TEXTBOX,
                    VALUE_TEXTBOX,
                    NONE
                };
```

6.9.2 Variable Documentation

6.9.2.1 TEXTBOX_COUNT

```
constexpr int constants::MenuArray::AddMode::TEXTBOX_COUNT = 2  [constexpr]
```

Definition at line 98 of file Constants.hpp.

6.9.2.2 TEXTBOX_LENGTH

```
constexpr int constants::MenuArray::AddMode::TEXTBOX_LENGTH[2]  [constexpr]
```

Initial value:

```
= {
                    2,
                    2
            }
```

Definition at line 103 of file Constants.hpp.

6.9.2.3 TEXTBOX_NAMES

```
constexpr char constants::MenuArray::AddMode::TEXTBOX_NAMES[2][50]  [constexpr]
```

Initial value:

```
= {
                    "Position = ",
                    "Value = "
            }
```

Definition at line 99 of file Constants.hpp.

6.10 constants::MenuArray::AllocateMode Namespace Reference

Enumerations

- enum Textbox { VALUE_TEXTBOX , NONE }

Variables

- constexpr int TEXTBOX_COUNT = 1
- constexpr char TEXTBOX_NAME [50] = "Size = "
- constexpr int TEXTBOX_LENGTH = 2

6.10.1 Enumeration Type Documentation

6.10.1.1 Textbox

```
enum constants::MenuArray::AllocateMode::Textbox
```

Enumerator

VALUE_TEXTBOX	
NONE	

Definition at line 151 of file Constants.hpp.

```
                                    {
                        VALUE_TEXTBOX,
                        NONE
                };
```

6.10.2 Variable Documentation

6.10.2.1 TEXTBOX_COUNT

```
constexpr int constants::MenuArray::AllocateMode::TEXTBOX_COUNT = 1  [constexpr]
```

Definition at line 148 of file Constants.hpp.

6.10.2.2 TEXTBOX_LENGTH

```
constexpr int constants::MenuArray::AllocateMode::TEXTBOX_LENGTH = 2  [constexpr]
```

Definition at line 150 of file Constants.hpp.

6.10.2.3 TEXTBOX_NAME

```
constexpr char constants::MenuArray::AllocateMode::TEXTBOX_NAME[50] = "Size = "  [constexpr]
```

Definition at line 149 of file Constants.hpp.

6.11 constants::MenuArray::CreateMode Namespace Reference

Enumerations

- enum Button { RANDOM_BUTTON , DEFINED_LIST_BUTTON , FILE_BUTTON , NONE }

Variables

- constexpr int BUTTON_COUNT = 3
- constexpr char BUTTON_NAMES [BUTTON_COUNT][50]
- constexpr int NAME_SIZE = 15
- constexpr int TEXTBOX_COUNT = 2
- constexpr char TEXTBOX_NAMES [2][50]
- constexpr int TEXTBOX_LENGTH [2]

6.11.1 Enumeration Type Documentation

6.11.1.1 Button

```
enum constants::MenuArray::CreateMode::Button
```

Enumerator

RANDOM_BUTTON	
DEFINED_LIST_BUTTON	
FILE_BUTTON	
NONE	

Definition at line 73 of file Constants.hpp.

```
                         {
                 RANDOM_BUTTON,
                 DEFINED_LIST_BUTTON,
                 FILE_BUTTON,
                 NONE
             };
```

6.11.2 Variable Documentation

6.11.2.1 BUTTON_COUNT

```
constexpr int constants::MenuArray::CreateMode::BUTTON_COUNT = 3  [constexpr]
```

Definition at line 72 of file Constants.hpp.

6.11.2.2 BUTTON_NAMES

```
constexpr char constants::MenuArray::CreateMode::BUTTON_NAMES[BUTTON_COUNT][50]  [constexpr]
```

Initial value:

```
= {
                    "Random",
                    "Defined List",
                    "File"
            }
```

Definition at line 79 of file Constants.hpp.

6.11.2.3 NAME_SIZE

```
constexpr int constants::MenuArray::CreateMode::NAME_SIZE = 15  [constexpr]
```

Definition at line 84 of file Constants.hpp.

6.11.2.4 TEXTBOX_COUNT

```
constexpr int constants::MenuArray::CreateMode::TEXTBOX_COUNT = 2  [constexpr]
```

Definition at line 86 of file Constants.hpp.

6.11.2.5 TEXTBOX_LENGTH

```
constexpr int constants::MenuArray::CreateMode::TEXTBOX_LENGTH[2]  [constexpr]
```

Initial value:

```
= {
                    2,
                    30
            }
```

Definition at line 92 of file Constants.hpp.

6.11.2.6 TEXTBOX_NAMES

```
constexpr char constants::MenuArray::CreateMode::TEXTBOX_NAMES[2][50]  [constexpr]
```

Initial value:
```
= {
                    "Amount = ",
                    "List = "
            }
```

Definition at line 87 of file Constants.hpp.

6.12 constants::MenuArray::DeleteMode Namespace Reference

Enumerations

- enum Textbox { POSITION_TEXTBOX , NONE }

Variables

- constexpr int TEXTBOX_COUNT = 1
- constexpr char TEXTBOX_NAME [50] = "Position = "
- constexpr int TEXTBOX_LENGTH = 2

6.12.1 Enumeration Type Documentation

6.12.1.1 Textbox

```
enum constants::MenuArray::DeleteMode::Textbox
```

Enumerator

POSITION_TEXTBOX	
NONE	

Definition at line 117 of file Constants.hpp.
```
                        {
                    POSITION_TEXTBOX,
                    NONE
                };
```

6.12.2 Variable Documentation

6.12.2.1 TEXTBOX_COUNT

```
constexpr int constants::MenuArray::DeleteMode::TEXTBOX_COUNT = 1  [constexpr]
```

Definition at line 114 of file Constants.hpp.

6.12.2.2 TEXTBOX_LENGTH

```
constexpr int constants::MenuArray::DeleteMode::TEXTBOX_LENGTH = 2  [constexpr]
```

Definition at line 116 of file Constants.hpp.

6.12.2.3 TEXTBOX_NAME

```
constexpr char constants::MenuArray::DeleteMode::TEXTBOX_NAME[50] = "Position = "  [constexpr]
```

Definition at line 115 of file Constants.hpp.

6.13 constants::MenuArray::SearchMode Namespace Reference

Enumerations

- enum Textbox { VALUE_TEXTBOX , NONE }

Variables

- constexpr int TEXTBOX_COUNT = 1
- constexpr char TEXTBOX_NAME [50] = "Value = "
- constexpr int TEXTBOX_LENGTH = 2

6.13.1 Enumeration Type Documentation

6.13.1.1 Textbox

```
enum constants::MenuArray::SearchMode::Textbox
```

Enumerator

VALUE_TEXTBOX	
NONE	

Definition at line 142 of file Constants.hpp.

```
                    {
            VALUE_TEXTBOX,
            NONE
        };
```

6.13.2 Variable Documentation

6.13.2.1 TEXTBOX_COUNT

```
constexpr int constants::MenuArray::SearchMode::TEXTBOX_COUNT = 1  [constexpr]
```

Definition at line 139 of file Constants.hpp.

6.13.2.2 TEXTBOX_LENGTH

```
constexpr int constants::MenuArray::SearchMode::TEXTBOX_LENGTH = 2  [constexpr]
```

Definition at line 141 of file Constants.hpp.

6.13.2.3 TEXTBOX_NAME

```
constexpr char constants::MenuArray::SearchMode::TEXTBOX_NAME[50] = "Value = "  [constexpr]
```

Definition at line 140 of file Constants.hpp.

6.14 constants::MenuArray::UpdateMode Namespace Reference

Enumerations

- enum Textbox { POSITION_TEXTBOX , VALUE_TEXTBOX , NONE }

Variables

- constexpr int TEXTBOX_COUNT = 2
- constexpr char TEXTBOX_NAMES [2][50]
- constexpr int TEXTBOX_LENGTH [2]

6.14.1 Enumeration Type Documentation

6.14.1.1 Textbox

```
enum constants::MenuArray::UpdateMode::Textbox
```

Enumerator

POSITION_TEXTBOX	
VALUE_TEXTBOX	
NONE	

Definition at line 132 of file Constants.hpp.

```
                    {
            POSITION_TEXTBOX,
            VALUE_TEXTBOX,
            NONE
        };
```

6.14.2 Variable Documentation

6.14.2.1 TEXTBOX_COUNT

```
constexpr int constants::MenuArray::UpdateMode::TEXTBOX_COUNT = 2  [constexpr]
```

Definition at line 123 of file Constants.hpp.

6.14.2.2 TEXTBOX_LENGTH

```
constexpr int constants::MenuArray::UpdateMode::TEXTBOX_LENGTH[2]  [constexpr]
```

Initial value:

```
= {
                  2,
                  2
          }
```

Definition at line 128 of file Constants.hpp.

6.14.2.3 TEXTBOX_NAMES

```
constexpr char constants::MenuArray::UpdateMode::TEXTBOX_NAMES[2][50]  [constexpr]
```

Initial value:

```
= {
                  "Position = ",
                  "Value = "
          }
```

Definition at line 124 of file Constants.hpp.

6.15 constants::MenuDataStructure Namespace Reference

Namespaces

- namespace CreateMode
- namespace PushMode

Enumerations

- enum Button {
  CREATE_BUTTON , PUSH_BUTTON , POP_BUTTON , CLEAR_BUTTON ,
  NONE }

Variables

- constexpr int BUTTON_COUNT = 4
- constexpr char BUTTON_NAMES [BUTTON_COUNT][50]
- constexpr int BUTTON_NAME_SIZE = 15

6.15.1 Enumeration Type Documentation

6.15.1.1 Button

```
enum constants::MenuDataStructure::Button
```

Enumerator

CREATE_BUTTON	
PUSH_BUTTON	
POP_BUTTON	
CLEAR_BUTTON	
NONE	

Definition at line 160 of file Constants.hpp.

```
                    {
            CREATE_BUTTON,
            PUSH_BUTTON,
            POP_BUTTON,
            CLEAR_BUTTON,
            NONE
        };
```

6.15.2 Variable Documentation

6.15.2.1 BUTTON_COUNT

```
constexpr int constants::MenuDataStructure::BUTTON_COUNT = 4  [constexpr]
```

Definition at line 159 of file Constants.hpp.

6.15.2.2 BUTTON_NAME_SIZE

```
constexpr int constants::MenuDataStructure::BUTTON_NAME_SIZE = 15  [constexpr]
```

Definition at line 173 of file Constants.hpp.

6.15.2.3 BUTTON_NAMES

```
constexpr char constants::MenuDataStructure::BUTTON_NAMES[BUTTON_COUNT][50]  [constexpr]
```

Initial value:
```
= {
                "Create",
                "Push",
                "Pop",
                "Clear"
        }
```

Definition at line 167 of file Constants.hpp.

6.16 constants::MenuDataStructure::CreateMode Namespace Reference

Enumerations

- enum Button { RANDOM_BUTTON , DEFINED_LIST_BUTTON , FILE_BUTTON , NONE }

Variables

- constexpr int BUTTON_COUNT = 3
- constexpr char BUTTON_NAMES [BUTTON_COUNT][50]
- constexpr int NAME_SIZE = 15
- constexpr int TEXTBOX_COUNT = 2
- constexpr char TEXTBOX_NAMES [2][50]
- constexpr int TEXTBOX_LENGTH [2]

6.16.1 Enumeration Type Documentation

6.16.1.1 Button

```
enum constants::MenuDataStructure::CreateMode::Button
```

Enumerator

RANDOM_BUTTON	
DEFINED_LIST_BUTTON	
FILE_BUTTON	
NONE	

Definition at line 177 of file Constants.hpp.

```
                         {
                    RANDOM_BUTTON,
                    DEFINED_LIST_BUTTON,
                    FILE_BUTTON,
                    NONE
                };
```

6.16.2 Variable Documentation

6.16.2.1 BUTTON_COUNT

`constexpr int constants::MenuDataStructure::CreateMode::BUTTON_COUNT = 3 [constexpr]`

Definition at line 176 of file Constants.hpp.

6.16.2.2 BUTTON_NAMES

`constexpr char constants::MenuDataStructure::CreateMode::BUTTON_NAMES[BUTTON_COUNT][50] [constexpr]`

Initial value:

```
= {
                    "Random",
                    "Defined List",
                    "File"
            }
```

Definition at line 183 of file Constants.hpp.

6.16.2.3 NAME_SIZE

`constexpr int constants::MenuDataStructure::CreateMode::NAME_SIZE = 15 [constexpr]`

Definition at line 188 of file Constants.hpp.

6.16.2.4 TEXTBOX_COUNT

```
constexpr int constants::MenuDataStructure::CreateMode::TEXTBOX_COUNT = 2  [constexpr]
```

Definition at line 190 of file Constants.hpp.

6.16.2.5 TEXTBOX_LENGTH

```
constexpr int constants::MenuDataStructure::CreateMode::TEXTBOX_LENGTH[2]  [constexpr]
```

Initial value:

```
= {
                    2,
                    30
          }
```

Definition at line 196 of file Constants.hpp.

6.16.2.6 TEXTBOX_NAMES

```
constexpr char constants::MenuDataStructure::CreateMode::TEXTBOX_NAMES[2][50]  [constexpr]
```

Initial value:

```
= {
                    "Amount = ",
                    "List = "
          }
```

Definition at line 191 of file Constants.hpp.

6.17 constants::MenuDataStructure::PushMode Namespace Reference

Enumerations

- enum Textbox { VALUE_TEXTBOX , NONE }

Variables

- constexpr int TEXTBOX_COUNT = 1
- constexpr char TEXTBOX_NAME [50] = "Value = "
- constexpr int TEXTBOX_LENGTH = 2

6.17.1 Enumeration Type Documentation

6.17.1.1 Textbox

```
enum constants::MenuDataStructure::PushMode::Textbox
```

Enumerator

VALUE_TEXTBOX	
NONE	

Definition at line 205 of file Constants.hpp.

```
            {
        VALUE_TEXTBOX,
        NONE
    };
```

6.17.2 Variable Documentation

6.17.2.1 TEXTBOX_COUNT

```
constexpr int constants::MenuDataStructure::PushMode::TEXTBOX_COUNT = 1  [constexpr]
```

Definition at line 202 of file Constants.hpp.

6.17.2.2 TEXTBOX_LENGTH

```
constexpr int constants::MenuDataStructure::PushMode::TEXTBOX_LENGTH = 2  [constexpr]
```

Definition at line 204 of file Constants.hpp.

6.17.2.3 TEXTBOX_NAME

```
constexpr char constants::MenuDataStructure::PushMode::TEXTBOX_NAME[50] = "Value = "  [constexpr]
```

Definition at line 203 of file Constants.hpp.

6.18 constants::MenuLinkedList Namespace Reference

Namespaces

- namespace AddMode
- namespace CreateMode
- namespace DeleteMode
- namespace SearchMode
- namespace UpdateMode

Enumerations

- enum Button {
  CREATE_BUTTON , ADD_BUTTON , DELETE_BUTTON , UPDATE_BUTTON ,
  SEARCH_BUTTON , NONE }

Variables

- constexpr int BUTTON_COUNT = 5
- constexpr char BUTTON_NAMES [BUTTON_COUNT][50]
- constexpr int BUTTON_NAME_SIZE = 15

6.18.1 Enumeration Type Documentation

6.18.1.1 Button

```
enum constants::MenuLinkedList::Button
```

Enumerator

CREATE_BUTTON	
ADD_BUTTON	
DELETE_BUTTON	
UPDATE_BUTTON	
SEARCH_BUTTON	
NONE	

Definition at line 214 of file Constants.hpp.

```
                    {
            CREATE_BUTTON,
            ADD_BUTTON,
            DELETE_BUTTON,
            UPDATE_BUTTON,
            SEARCH_BUTTON,
            NONE
        };
```

6.18.2 Variable Documentation

6.18.2.1 BUTTON_COUNT

```
constexpr int constants::MenuLinkedList::BUTTON_COUNT = 5  [constexpr]
```

Definition at line 213 of file Constants.hpp.

6.18.2.2 BUTTON_NAME_SIZE

```
constexpr int constants::MenuLinkedList::BUTTON_NAME_SIZE = 15  [constexpr]
```

Definition at line 229 of file Constants.hpp.

6.18.2.3 BUTTON_NAMES

```
constexpr char constants::MenuLinkedList::BUTTON_NAMES[BUTTON_COUNT][50]  [constexpr]
```

Initial value:

```
= {
                "Create",
                "Add",
                "Delete",
                "Update",
                "Search"
        }
```

Definition at line 222 of file Constants.hpp.

6.19 constants::MenuLinkedList::AddMode Namespace Reference

Enumerations

- enum Textbox { POSITION_TEXTBOX , VALUE_TEXTBOX , NONE }

Variables

- constexpr int TEXTBOX_COUNT = 2
- constexpr char TEXTBOX_NAMES [2][50]
- constexpr int TEXTBOX_LENGTH [2]

6.19.1 Enumeration Type Documentation

6.19.1.1 Textbox

```
enum constants::MenuLinkedList::AddMode::Textbox
```

Enumerator

POSITION_TEXTBOX	
VALUE_TEXTBOX	
NONE	

Definition at line 267 of file Constants.hpp.

```
                    {
            POSITION_TEXTBOX,
            VALUE_TEXTBOX,
            NONE
        };
```

6.19.2 Variable Documentation

6.19.2.1 TEXTBOX_COUNT

```
constexpr int constants::MenuLinkedList::AddMode::TEXTBOX_COUNT = 2  [constexpr]
```

Definition at line 258 of file Constants.hpp.

6.19.2.2 TEXTBOX_LENGTH

```
constexpr int constants::MenuLinkedList::AddMode::TEXTBOX_LENGTH[2]  [constexpr]
```

Initial value:

```
= {
                  2,
                  2
          }
```

Definition at line 263 of file Constants.hpp.

6.19.2.3 TEXTBOX_NAMES

```
constexpr char constants::MenuLinkedList::AddMode::TEXTBOX_NAMES[2][50]  [constexpr]
```

Initial value:

```
= {
                  "Position = ",
                  "Value = "
          }
```

Definition at line 259 of file Constants.hpp.

6.20 constants::MenuLinkedList::CreateMode Namespace Reference

Enumerations

- enum Button { RANDOM_BUTTON , DEFINED_LIST_BUTTON , FILE_BUTTON , NONE }

Variables

- constexpr int BUTTON_COUNT = 3
- constexpr char BUTTON_NAMES [BUTTON_COUNT][50]
- constexpr int NAME_SIZE = 15
- constexpr int TEXTBOX_COUNT = 2
- constexpr char TEXTBOX_NAMES [2][50]
- constexpr int TEXTBOX_LENGTH [2]

6.20.1 Enumeration Type Documentation

6.20.1.1 Button

```
enum constants::MenuLinkedList::CreateMode::Button
```

Enumerator

RANDOM_BUTTON	
DEFINED_LIST_BUTTON	
FILE_BUTTON	
NONE	

Definition at line 233 of file Constants.hpp.

```
00233                     {
00234             RANDOM_BUTTON,
00235             DEFINED_LIST_BUTTON,
00236             FILE_BUTTON,
00237             NONE
00238         };
```

6.20.2 Variable Documentation

6.20.2.1 BUTTON_COUNT

```
constexpr int constants::MenuLinkedList::CreateMode::BUTTON_COUNT = 3  [constexpr]
```

Definition at line 232 of file Constants.hpp.

6.20.2.2 BUTTON_NAMES

```
constexpr char constants::MenuLinkedList::CreateMode::BUTTON_NAMES[BUTTON_COUNT][50]  [constexpr]
```

Initial value:

```
= {
                    "Random",
                    "Defined List",
                    "File"
            }
```

Definition at line 239 of file Constants.hpp.

6.20.2.3 NAME_SIZE

```
constexpr int constants::MenuLinkedList::CreateMode::NAME_SIZE = 15  [constexpr]
```

Definition at line 244 of file Constants.hpp.

6.20.2.4 TEXTBOX_COUNT

```
constexpr int constants::MenuLinkedList::CreateMode::TEXTBOX_COUNT = 2  [constexpr]
```

Definition at line 246 of file Constants.hpp.

6.20.2.5 TEXTBOX_LENGTH

```
constexpr int constants::MenuLinkedList::CreateMode::TEXTBOX_LENGTH[2]  [constexpr]
```

Initial value:

```
= {
                    2,
                    30
            }
```

Definition at line 252 of file Constants.hpp.

6.20.2.6 TEXTBOX_NAMES

```
constexpr char constants::MenuLinkedList::CreateMode::TEXTBOX_NAMES[2][50]  [constexpr]
```

Initial value:

```
= {
                    "Amount = ",
                    "List = "
            }
```

Definition at line 247 of file Constants.hpp.

6.21 constants::MenuLinkedList::DeleteMode Namespace Reference

Enumerations

- enum Textbox { POSITION_TEXTBOX , NONE }

Variables

- constexpr int TEXTBOX_COUNT = 1
- constexpr char TEXTBOX_NAME [50] = "Position = "
- constexpr int TEXTBOX_LENGTH = 2

6.21.1 Enumeration Type Documentation

6.21.1.1 Textbox

```
enum constants::MenuLinkedList::DeleteMode::Textbox
```

Enumerator

POSITION_TEXTBOX	
NONE	

Definition at line 277 of file Constants.hpp.

```
                        {
                POSITION_TEXTBOX,
                NONE
            };
```

6.21.2 Variable Documentation

6.21.2.1 TEXTBOX_COUNT

```
constexpr int constants::MenuLinkedList::DeleteMode::TEXTBOX_COUNT = 1  [constexpr]
```

Definition at line 274 of file Constants.hpp.

6.21.2.2 TEXTBOX_LENGTH

```
constexpr int constants::MenuLinkedList::DeleteMode::TEXTBOX_LENGTH = 2  [constexpr]
```

Definition at line 276 of file Constants.hpp.

6.21.2.3 TEXTBOX_NAME

```
constexpr char constants::MenuLinkedList::DeleteMode::TEXTBOX_NAME[50] = "Position = "  [constexpr]
```

Definition at line 275 of file Constants.hpp.

6.22 constants::MenuLinkedList::SearchMode Namespace Reference

Enumerations

- enum Textbox { VALUE_TEXTBOX , NONE }

Variables

- constexpr int TEXTBOX_COUNT = 1
- constexpr char TEXTBOX_NAME [50] = "Value = "
- constexpr int TEXTBOX_LENGTH = 2

6.22.1 Enumeration Type Documentation

6.22.1.1 Textbox

```
enum constants::MenuLinkedList::SearchMode::Textbox
```

Enumerator

VALUE_TEXTBOX	
NONE	

Definition at line 302 of file Constants.hpp.

```
                        {
                    VALUE_TEXTBOX,
                    NONE
                };
```

6.22.2 Variable Documentation

6.22.2.1 TEXTBOX_COUNT

```
constexpr int constants::MenuLinkedList::SearchMode::TEXTBOX_COUNT = 1  [constexpr]
```

Definition at line 299 of file Constants.hpp.

6.22.2.2 TEXTBOX_LENGTH

```
constexpr int constants::MenuLinkedList::SearchMode::TEXTBOX_LENGTH = 2  [constexpr]
```

Definition at line 301 of file Constants.hpp.

6.22.2.3 TEXTBOX_NAME

```
constexpr char constants::MenuLinkedList::SearchMode::TEXTBOX_NAME[50] = "Value = "  [constexpr]
```

Definition at line 300 of file Constants.hpp.

6.23 constants::MenuLinkedList::UpdateMode Namespace Reference

Enumerations

- enum Textbox { POSITION_TEXTBOX , VALUE_TEXTBOX , NONE }

Variables

- constexpr int TEXTBOX_COUNT = 2
- constexpr char TEXTBOX_NAMES [2][50]
- constexpr int TEXTBOX_LENGTH [2]

6.23.1 Enumeration Type Documentation

6.23.1.1 Textbox

```
enum constants::MenuLinkedList::UpdateMode::Textbox
```

Enumerator

POSITION_TEXTBOX	
VALUE_TEXTBOX	
NONE	

Definition at line 292 of file Constants.hpp.

```
                        {
                POSITION_TEXTBOX,
                VALUE_TEXTBOX,
                NONE
            };
```

6.23.2 Variable Documentation

6.23.2.1 TEXTBOX_COUNT

```
constexpr int constants::MenuLinkedList::UpdateMode::TEXTBOX_COUNT = 2  [constexpr]
```

Definition at line 283 of file Constants.hpp.

6.23.2.2 TEXTBOX_LENGTH

```
constexpr int constants::MenuLinkedList::UpdateMode::TEXTBOX_LENGTH[2]  [constexpr]
```

Initial value:

```
= {
                    2,
                    2
            }
```

Definition at line 288 of file Constants.hpp.

6.23.2.3 TEXTBOX_NAMES

```
constexpr char constants::MenuLinkedList::UpdateMode::TEXTBOX_NAMES[2][50]  [constexpr]
```

Initial value:

```
= {
                    "Position = ",
                    "Value = "
            }
```

Definition at line 284 of file Constants.hpp.

6.24 constants::NodeInfo Namespace Reference

6.25 constants::sceneVariables Namespace Reference

Enumerations

- enum Scene {
  MAIN_MENU_SCENE , SINGLY_LINKED_LIST_SCENE , DOUBLY_LINKED_LIST_SCENE , CIRCULAR_LINKED_LIST_SCE
  ,
  STACK_SCENE , QUEUE_SCENE , STATIC_ARRAY_SCENE , DYNAMIC_ARRAY_SCENE }

Variables

- constexpr int SCENE_COUNT = 8
- constexpr char SCENE_NAMES [SCENE_COUNT][50]
- constexpr char NAME_MODE_BUTTON [SCENE_COUNT][50]

6.25.1 Enumeration Type Documentation

6.25.1.1 Scene

`enum constants::sceneVariables::Scene`

Enumerator

MAIN_MENU_SCENE	
SINGLY_LINKED_LIST_SCENE	
DOUBLY_LINKED_LIST_SCENE	
CIRCULAR_LINKED_LIST_SCENE	
STACK_SCENE	
QUEUE_SCENE	
STATIC_ARRAY_SCENE	
DYNAMIC_ARRAY_SCENE	

Definition at line 13 of file Constants.hpp.

```
00013                {
00014            MAIN_MENU_SCENE,
00015            SINGLY_LINKED_LIST_SCENE,
00016            DOUBLY_LINKED_LIST_SCENE,
00017            CIRCULAR_LINKED_LIST_SCENE,
00018            STACK_SCENE,
00019            QUEUE_SCENE,
00020            STATIC_ARRAY_SCENE,
00021            DYNAMIC_ARRAY_SCENE,
00022        };
```

6.25.2 Variable Documentation

6.25.2.1 NAME_MODE_BUTTON

```
constexpr char constants::sceneVariables::NAME_MODE_BUTTON[SCENE_COUNT][50]  [constexpr]
```

Initial value:

```
= {
            "Main Menu",
            "SLL",
            "DLL",
            "CLL",
            "Stack",
            "Queue",
            "Static Array",
            "Dynamic Array"
        }
```

Definition at line 33 of file Constants.hpp.

6.25.2.2 SCENE_COUNT

```
constexpr int constants::sceneVariables::SCENE_COUNT = 8  [constexpr]
```

Definition at line 12 of file Constants.hpp.

6.25.2.3 SCENE_NAMES

```
constexpr char constants::sceneVariables::SCENE_NAMES[SCENE_COUNT][50]  [constexpr]
```

Initial value:

```
= {
                "Main Menu",
                "Singly Linked List",
                "Doubly Linked List",
                "Circular Linked List",
                "Stack",
                "Queue",
                "Static Array",
                "Dynamic Array",
        }
```

Definition at line 23 of file Constants.hpp.

6.26 constants::Square Namespace Reference

6.27 constants::TitleNode Namespace Reference

6.28 pfd Namespace Reference

Namespaces

- namespace internal

Classes

- class message
- class notify
- class open_file
- class path
- class save_file
- class select_folder
- class settings

Enumerations

- enum class button {
  cancel = -1 , ok , yes , no ,
  abort , retry , ignore }
- enum class choice {
  ok = 0 , ok_cancel , yes_no , yes_no_cancel ,
  retry_cancel , abort_retry_ignore }
- enum class icon { info = 0 , warning , error , question }
- enum class opt : uint8_t { none = 0 , multiselect = 0x1 , force_overwrite = 0x2 , force_path = 0x4 }

Functions

- opt operator| (opt a, opt b)
- bool operator& (opt a, opt b)
- std::ostream & operator<< (std::ostream &s, std::vector< std::string > const &v)

6.28.1 Enumeration Type Documentation

6.28.1.1 button

```
enum class pfd::button  [strong]
```

Enumerator

cancel	
ok	
yes	
no	
abort	
retry	
ignore	

Definition at line 73 of file FileDialog.h.

```
00074     {
00075         cancel = -1,
00076         ok,
00077         yes,
        no,
        abort,
        retry,
        ignore,
    };
```

6.28.1.2 choice

enum class pfd::choice [strong]

Enumerator

ok	
ok_cancel	
yes_no	
yes_no_cancel	
retry_cancel	
abort_retry_ignore	

Definition at line 84 of file FileDialog.h.

```
    {
        ok = 0,
        ok_cancel,
        yes_no,
        yes_no_cancel,
        retry_cancel,
        abort_retry_ignore,
    };
```

6.28.1.3 icon

enum class pfd::icon [strong]

Enumerator

info	
warning	
error	
question	

Definition at line 94 of file FileDialog.h.

```
    {
        info = 0,
        warning,
        error,
        question,
    };
```

6.28.1.4 opt

enum class pfd::opt : uint8_t [strong]

Enumerator

none	
multiselect	
force_overwrite	
force_path	

Definition at line 103 of file FileDialog.h.

```
    {
        none = 0,
        // For file open, allow multiselect.
        multiselect     = 0x1,
        // For file save, force overwrite and disable the confirmation dialog.
        force_overwrite = 0x2,
        // For folder select, force path to be the provided argument instead
        // of the last opened directory, which is the Microsoft-recommended,
        // user-friendly behaviour.
        force_path      = 0x4,
    };
```

6.28.2 Function Documentation

6.28.2.1 operator&()

```
bool pfd::operator& (
            opt a,
            opt b ) [inline]
```

Definition at line 117 of file FileDialog.h.

```
{ return bool(uint8_t(a) & uint8_t(b)); }
```

6.28.2.2 operator<<()

```
std::ostream & pfd::operator<< (
            std::ostream & s,
            std::vector< std::string > const & v ) [inline]
```

Definition at line 1024 of file FileDialog.h.

```
    {
        int not_first = 0;
        for (auto &e : v)
            s « (not_first++ ? " " : "") « e;
        return s;
    }
```

6.28.2.3 operator"|()

```
opt pfd::operator| (
            opt a,
            opt b ) [inline]
```

Definition at line 116 of file FileDialog.h.

```
{ return opt(uint8_t(a) | uint8_t(b)); }
```

6.29 pfd::internal Namespace Reference

Classes

- class dialog
- class executor
- class file_dialog
- class platform

6.30 Random Namespace Reference

6.31 sf Namespace Reference

Classes

- class RoundedRectangleShape

  Specialized shape representing a rectangle with rounded corners.

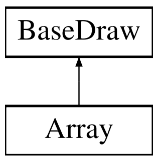
Chapter 7

Class Documentation

7.1 Array Class Reference

```
#include <Array.hpp>
```

Inheritance diagram for Array:



Public Types

- enum class TypeArray { DYNAMIC , STATIC }

Public Member Functions

- Array (sf::RenderWindow ∗window, TypeArray typeArray)
- Array (sf::RenderWindow ∗window, TypeArray typeArray, int size)
- Array (sf::RenderWindow ∗window, TypeArray typeArray, std::vector< std::string > values)
- void init (TypeArray typeArray)
- ∼Array ()=default
- void render () override
- void renderHighlighter ()
- void update ()
- void setSpeed (float speed)
- int findValue (const std::string &value)
- void updateAnimation ()
- void resetEvents ()
- int getSize () const
- int getSquaresSize () const
- void processControlMenu (ControlMenu::StatusCode status)
- void initHighlighter (int linesCount, const char ∗codePath)
- void toggleLines (std::vector< int > lines)

- void createArray (int size)
- void createArray (const std::vector< std::string > &values)
- void allocateSquare (int size, const std::vector< EventAnimation > &listEvents)
- void addSquare (int position, std::string value, const std::vector< EventAnimation > &listEvents)
- void deleteSquare (int position, const std::vector< EventAnimation > &listEvents)
- void updateSquare (int position, std::string value, const std::vector< EventAnimation > &listEvents)
- void searchSquare (const std::vector< EventAnimation > &listEvents)

Public Member Functions inherited from BaseDraw

- BaseDraw (sf::RenderWindow *window)
- virtual void render ()=0

Additional Inherited Members

Protected Attributes inherited from BaseDraw

- sf::RenderWindow * window

7.1.1 Detailed Description

Definition at line 15 of file Array.hpp.

7.1.2 Member Enumeration Documentation

7.1.2.1 TypeArray

```
enum class Array::TypeArray  [strong]
```

Enumerator

DYNAMIC	
STATIC	

Definition at line 17 of file Array.hpp.

```
                              {
        DYNAMIC,
        STATIC
    };
```

7.1.3 Constructor & Destructor Documentation

7.1.3.1 Array() [1/3]

```
Array::Array (
            sf::RenderWindow * window,
            TypeArray typeArray )
```

Definition at line 9 of file Array.cpp.

```
: BaseDraw(window) {
    this->init(typeArray);
    this->createArray(0);
}
```

7.1.3.2 Array() [2/3]

```
Array::Array (
            sf::RenderWindow * window,
            Array::TypeArray typeArray,
            int size )
```

Definition at line 14 of file Array.cpp.

```
: BaseDraw(window)  {
    this->init(typeArray);
    this->createArray(size);
}
```

7.1.3.3 Array() [3/3]

```
Array::Array (
            sf::RenderWindow * window,
            Array::TypeArray typeArray,
            std::vector< std::string > values )
```

Definition at line 19 of file Array.cpp.

```
:
     BaseDraw(window)  {
    this->init(typeArray);
    this->createArray(std::move(values));
}
```

7.1.3.4 ∼Array()

```
Array::∼Array ( )  [default]
```

7.1.4 Member Function Documentation

7.1.4.1 addSquare()

```
void Array::addSquare (
            int position,
            std::string value,
            const std::vector< EventAnimation > & listEvents )
```

Definition at line 234 of file Array.cpp.

```
{
    if (position < 0 || position > this->size)
        return;

    ++this->size;
    if (this->typeArray == TypeArray::DYNAMIC && this->size > this->getSquaresSize()) {
        this->squares.push_back(new SquareInfo(
                this->window,
                "",
                sf::Vector2f(
                        constants::Square::originNode.x + static_cast<float>(this->getSquaresSize()) *
constants::Square::offsetX,
                        constants::Square::originNode.y
                )
        ));
        this->squaresTemp.resize(this->size);
        for (int i = 0; i < this->size; ++i) {
            this->squaresTemp[i] = new SquareInfo(
                    this->window,
                    "",
                    sf::Vector2f(
                            constants::Square::originNode.x + static_cast<float>(i) *
constants::Square::offsetX,
                            constants::Square::originNode.y + constants::Square::offsetY
                    )
            );
            this->squaresTemp[i]->setValue(this->squares[i]->getValue());
        }
    }

    if (size > this->getSquaresSize())
        --this->size;

    for (int i = this->size - 1; i > position; --i)
        this->squares[i]->setValue(this->squares[i - 1]->getValue());
    this->squares[position]->setValue(std::move(value));
    for (int i = 0; i < position; ++i)
        this->squares[i]->setValue(this->squares[i]->getValue());

    this->currentEvent = 0;
    this->events = listEvents;
}
```

7.1.4.2 allocateSquare()

```
void Array::allocateSquare (
            int size,
            const std::vector< EventAnimation > & listEvents )
```

Definition at line 203 of file Array.cpp.

```
{
    this->squaresTemp.resize(_size);
//    this->squares.resize();

    while (this->squares.size() < _size)
        this->squares.push_back(new SquareInfo(
                this->window,
                "",
                sf::Vector2f(
                        constants::Square::originNode.x + static_cast<float>(this->squares.size()) *
constants::Square::offsetX,
                        constants::Square::originNode.y
                )
        ));

00217     for (int i = 0; i < _size; ++i) {
00218         this->squaresTemp[i] = new SquareInfo(
00219                 this->window,
00220                 "",
00221                 sf::Vector2f(
00222                         constants::Square::originNode.x + static_cast<float>(i) *
      constants::Square::offsetX,
00223                         constants::Square::originNode.y + constants::Square::offsetY
00224                 )
00225         );
00226         this->squaresTemp[i]->setValue(this->squares[i]->getValue());
00227     }
00228 
00229     this->size = std::min(this->size, _size);
00230     this->currentEvent = 0;
00231     this->events = listEvents;
00232 }
```

7.1.4.3 createArray() [1/2]

```
void Array::createArray (
               const std::vector< std::string > & values )
```

Definition at line 178 of file Array.cpp.

```
00178                                                             {
00179     this->resetEvents();
00180     this->size = (int)values.size();
00181     for (auto &square : this->squares)
00182         delete square;
00183     this->squares.resize(this->size);
00184     for (int i = 0; i < this->size; ++i) {
00185         this->squares[i] = new SquareInfo(
00186                 this->window,
00187                 values[i],
00188                 sf::Vector2f(
00189                         constants::Square::originNode.x + static_cast<float>(i) *
      constants::Square::offsetX,
00190                         constants::Square::originNode.y
00191                 )
00192         );
00193         this->squares[i]->setStatus(Square::Status::active);
00194     }
00195     if (this->size)
00196         this->squares[this->size - 1]->setTitle("n");
00197 }
```

7.1.4.4 createArray() [2/2]

```
void Array::createArray (
               int size )
```

Definition at line 157 of file Array.cpp.

```
00157                                         {
00158     this->resetEvents();
00159     this->size = _size;
00160     for (auto &square : this->squares)
00161         delete square;
00162     this->squares.resize(this->size);
00163     for (int i = 0; i < this->size; ++i) {
00164         this->squares[i] = new SquareInfo(
00165                 this->window,
00166                 std::to_string(Random::randomInt(0, 99)),
00167                 sf::Vector2f(
00168                         constants::Square::originNode.x + static_cast<float>(i) *
      constants::Square::offsetX,
00169                         constants::Square::originNode.y
00170                         )
00171         );
00172         this->squares[i]->setStatus(Square::Status::active);
00173     }
00174     if (this->size)
00175         this->squares[this->size - 1]->setTitle("n");
00176 }
```

7.1.4.5 deleteSquare()

```
void Array::deleteSquare (
            int position,
            const std::vector< EventAnimation > & listEvents )
```

Definition at line 275 of file Array.cpp.

```
                                                                                  {
    if (position < 0 || position >= this->size)
        return;

    --this->size;

    for (int i = position; i < this->size; ++i)
        this->squares[i]->setValue(this->squares[i + 1]->getValue());
    for (int i = 0; i < position; ++i)
        this->squares[i]->setValue(this->squares[i]->getValue());
    this->squares[this->size]->setValue(this->squares[this->size]->getValue());

    this->currentEvent = 0;
    this->events = listEvents;
}
```

7.1.4.6 findValue()

```
int Array::findValue (
            const std::string & value )
```

Definition at line 55 of file Array.cpp.

```
                                           {
    for (int i = 0; i < this->size; i++) {
        if (this->squares[i]->getValue() == value)
            return i;
    }
    return this->size;
}
```

7.1.4.7 getSize()

```
int Array::getSize ( ) const
```

Definition at line 63 of file Array.cpp.

```
                            {
    return this->size;
}
```

7.1.4.8 getSquaresSize()

```
int Array::getSquaresSize ( ) const
```

Definition at line 199 of file Array.cpp.

```
                                   {
    return (int)this->squares.size();
}
```

7.1.4.9 init()

```
void Array::init (
            Array::TypeArray typeArray )
```

Definition at line 24 of file Array.cpp.

```
                                         {
    this->typeArray = typeArray;
    this->highlighter = nullptr;
    this->delayTime = constants::LinkedList::DELAY_TIME;
    this->size = 0;
}
```

7.1.4.10 initHighlighter()

```
void Array::initHighlighter (
            int linesCount,
            const char * codePath )
```

Definition at line 92 of file Array.cpp.

```
                                                                        {
    delete this->highlighter;
    this->highlighter = new Highlighter(
            this->window,
            linesCount,
            codePath
    );
}
```

7.1.4.11 processControlMenu()

```
void Array::processControlMenu (
            ControlMenu::StatusCode status )
```

Definition at line 67 of file Array.cpp.

```
                                                                   {
    if (this->clock.getElapsedTime().asSeconds() < this->delayTime / this->speed)
        return;
    switch (status){
        case ControlMenu::StatusCode::PREVIOUS:
            if (this->currentEvent > 0)
                --this->currentEvent;
            break;
        case ControlMenu::StatusCode::PAUSE:
//              std::cout « "PAUSE" « std::endl;
            break;
        case ControlMenu::StatusCode::PLAY:
            if (this->currentEvent + 1 < this->events.size()) {
                this->isDelay = true;
                this->clock.restart();
            }
        case ControlMenu::StatusCode::NEXT:
            if (this->currentEvent + 1 < this->events.size())
                ++this->currentEvent;
            break;
        default:
            break;
    }
}
```

7.1.4.12 render()

```
void Array::render ( )  [override], [virtual]
```

Implements BaseDraw.

Definition at line 31 of file Array.cpp.

```
                       {
    for (auto &square : this->squares) {
        square->render();
    }
    for (auto &square: this->squaresTemp) {
        square->render();
    }
}
```

7.1.4.13 renderHighlighter()

```
void Array::renderHighlighter ( )
```

Definition at line 40 of file Array.cpp.

```
                                  {
    if (this->highlighter)
        this->highlighter->render();
}
```

7.1.4.14 resetEvents()

```
void Array::resetEvents ( )
```

Definition at line 139 of file Array.cpp.

```
                            {
    delete this->highlighter;
    this->highlighter = nullptr;
    this->currentEvent = 0;
    this->events.clear();
    this->squaresTemp.clear();

    while (!this->squares.empty() && this->squares.back()->getStatus() == Square::Status::hidden)
        this->squares.pop_back();

    for (int i = 0; i < this->size; ++i)
        this->squares[i]->setStatus(Square::Status::active);
    for (int i = this->size; i < this->squares.size(); ++i)
        this->squares[i]->setStatus(Square::Status::inactive);
    if (this->size)
        this->squares[this->size - 1]->setTitle("n");
}
```

7.1.4.15 searchSquare()

```
void Array::searchSquare (
            const std::vector< EventAnimation > & listEvents )
```

Definition at line 301 of file Array.cpp.

```
                                                                        {
    this->currentEvent = 0;
    this->events = listEvents;
}
```

7.1.4.16 setSpeed()

```
void Array::setSpeed (
             float speed )
```

Definition at line 51 of file Array.cpp.

```
00051                                   {
00052     this->speed = _speed;
00053 }
```

7.1.4.17 toggleLines()

```
void Array::toggleLines (
             std::vector< int > lines )
```

Definition at line 101 of file Array.cpp.

```
00101                                              {
00102     this->highlighter->toggle(std::move(lines));
00103 }
```

7.1.4.18 update()

```
void Array::update ( )
```

Definition at line 45 of file Array.cpp.

```
00045                       {
00046     if ((int)this->events.size() && (this->isDelay or this->clock.getElapsedTime().asSeconds() >
      this->delayTime / this->speed))
00047         this->updateAnimation();
00048     this->isDelay = false;
00049 }
```

7.1.4.19 updateAnimation()

```
void Array::updateAnimation ( )
```

Definition at line 105 of file Array.cpp.

```
00105                              {
00106     if (this->squares.empty())
00107         return;
00108 
00109     for (auto &square : this->squares) {
00110         square->reset();
00111     }
00112     for (auto &square : this->squaresTemp) {
00113         square->reset();
00114     }
00115 
00116     EventAnimation &event = this->events[this->currentEvent];
00117     for (int i = 0; i < event.eventSquares.size(); ++i) {
00118         this->squares[i]->setStatus(event.eventSquares[i].status);
00119         this->squares[i]->setPrintPreVal(event.eventSquares[i].isPrintPreVal);
00120         this->squares[i]->setTitle(event.eventSquares[i].title);
00121     }
00122     for (int i = 0; i < event.eventSquaresTemp.size(); ++i) {
00123         this->squaresTemp[i]->setStatus(event.eventSquaresTemp[i].status);
00124         this->squaresTemp[i]->setPrintPreVal(event.eventSquaresTemp[i].isPrintPreVal);
00125         this->squaresTemp[i]->setTitle(event.eventSquaresTemp[i].title);
00126     }
00127
00128     if (this->highlighter)
00129         this->highlighter->toggle(event.lines);
00130
00131     for (auto &square : this->squares) {
00132         square->update();
00133     }
00134     for (auto &square : this->squaresTemp) {
00135         square->update();
00136     }
00137 }
```

7.1.4.20 updateSquare()

```
void Array::updateSquare (
            int position,
            std::string value,
            const std::vector< EventAnimation > & listEvents )
```

Definition at line 291 of file Array.cpp.

```
00291 {
00292     if (position < 0 || position >= this->size)
00293         return;
00294
00295     this->squares[position]->setValue(std::move(value));
00296
00297     this->currentEvent = 0;
00298     this->events = listEvents;
00299 }
```

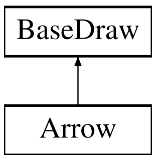
The documentation for this class was generated from the following files:

- include/core/Array.hpp
- include/core/Array.cpp

7.2 Arrow Class Reference

`#include <Arrow.hpp>`

Inheritance diagram for Arrow:



Public Member Functions

- Arrow (sf::RenderWindow ∗window, sf::Vector2f start, sf::Vector2f end)
- void render () override
- void toggleActiveColor ()
- void resetColor ()
- void setStart (sf::Vector2f start, bool needSetMid)
- void setPositions (sf::Vector2f start, sf::Vector2f end, bool needSetMid)
- void setMid ()
- void autoRotate ()
- void autoScale ()
- void hide ()
- void show ()

Public Member Functions inherited from BaseDraw

- BaseDraw (sf::RenderWindow *window)
- virtual void render ()=0

Protected Attributes

- sf::Vector2f points [2]
- sf::Texture arrowTexture [2]
- sf::Sprite arrowSprite
- float length
- bool hasSetMid

Protected Attributes inherited from BaseDraw

- sf::RenderWindow * window

7.2.1 Detailed Description

Definition at line 13 of file Arrow.hpp.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 Arrow()

```
Arrow::Arrow (
            sf::RenderWindow * window,
            sf::Vector2f start,
            sf::Vector2f end )
```

Definition at line 7 of file Arrow.cpp.

```
                                                                      : BaseDraw(window) {
    this->points[0] = start;
    this->points[1] = end;

    this->arrowTexture[0].loadFromFile("../assets/arrow/arrow_black.png");
    this->arrowTexture[1].loadFromFile("../assets/arrow/arrow_orange.png");

    this->arrowTexture[0].setSmooth(true);
    this->arrowTexture[1].setSmooth(true);

    this->arrowSprite.setTexture(this->arrowTexture[0]);
    sf::Vector2i topLeftCorner(
            static_cast<int>(this->arrowTexture[0].getSize().x / 2.0 - constants::Arrow::sizeArrow.x /
      2.0),
            static_cast<int>(this->arrowTexture[0].getSize().y / 2.0 - constants::Arrow::sizeArrow.y /
      2.0)
    );
    this->arrowSprite.setTextureRect(sf::IntRect(
            topLeftCorner.x,
            topLeftCorner.y,
            constants::Arrow::sizeArrow.x,
            constants::Arrow::sizeArrow.y
    ));

    this->autoScale();
    this->autoRotate();
```

```
00031
00032 //    this->rectangleTexture[0].loadFromFile("../assets/rectangle/rectangle_black.png");
00033 //    this->rectangleTexture[1].loadFromFile("../assets/rectangle/rectangle_orange.png");
00034 //    topLeftCorner = sf::Vector2i(
00035 //            static_cast<int>(this->rectangleTexture[0].getSize().x / 2.0 -
      constants::Arrow::sizeRectangle.x / 2.0),
00036 //            static_cast<int>(this->rectangleTexture[0].getSize().y / 2.0 -
      constants::Arrow::sizeRectangle.y / 2.0)
00037 //    );
00038 //    this->rectangleSprite.setTexture(this->rectangleTexture[0]);
00039 //    this->rectangleSprite.setTextureRect(sf::IntRect(
00040 //            topLeftCorner.x,
00041 //            topLeftCorner.y,
00042 //            constants::Arrow::sizeRectangle.x,
00043 //            constants::Arrow::sizeRectangle.y
00044 //    ));
00045 //    this->rectangleSprite.setScale(
00046 //            constants::Arrow::defaultScaleRectangle.x,
00047 //            constants::Arrow::defaultScaleRectangle.y
00048 //            );
00049 //    this->rectangleSprite.setOrigin(
00050 //            0,
00051 //            this->rectangleSprite.getLocalBounds().height / 2.0f
00052 //    );
00053 //    this->rectangleSprite.setPosition(sf::Vector2f(50, 200));
00054 //    this->rectangleSprite.setRotation(angle);
00055
00056     this->hasSetMid = false;
00057 }
```

7.2.3 Member Function Documentation

7.2.3.1 autoRotate()

```
void Arrow::autoRotate ( )
```

Definition at line 88 of file Arrow.cpp.

```
00088                        {
00089     sf::Vector2f vector2point = this->points[1] - this->points[0];
00090     auto angle = static_cast<float>(atan2(vector2point.y, vector2point.x) * 180 / M_PI);
00091     this->arrowSprite.setRotation(angle);
00092 }
```

7.2.3.2 autoScale()

```
void Arrow::autoScale ( )
```

Definition at line 94 of file Arrow.cpp.

```
00094                       {
00095     this->length = static_cast<float>(
00096             sqrt(
00097                     pow(this->points[1].x - this->points[0].x, 2) + pow(this->points[1].y -
      this->points[0].y, 2)
00098             ) - constants::NodeInfo::radius - 2.f
00099             );
00100     this->arrowSprite.setScale(
00101             this->length / this->arrowSprite.getLocalBounds().width,
00102             constants::Arrow::defaultScaleArrow.y
00103     );
00104     this->arrowSprite.setOrigin(
00105             0,
00106             this->arrowSprite.getLocalBounds().height / 2.0f
00107     );
00108     this->arrowSprite.setPosition(this->points[0]);
00109 }
```

7.2.3.3 hide()

```
void Arrow::hide ( )
```

Definition at line 125 of file Arrow.cpp.

```
00125                   {
00126     sf::Color tmp = this->arrowSprite.getColor();
00127     tmp.a = 0;
00128     this->arrowSprite.setColor(tmp);
00129 }
```

7.2.3.4 render()

```
void Arrow::render ( )  [override], [virtual]
```

Implements BaseDraw.

Definition at line 59 of file Arrow.cpp.

```
00059                     {
00060     this->window->draw(this->arrowSprite);
00061 //    this->window->draw(this->rectangleSprite);
00062 }
```

7.2.3.5 resetColor()

```
void Arrow::resetColor ( )
```

Definition at line 69 of file Arrow.cpp.

```
00069                         {
00070     this->arrowSprite.setTexture(this->arrowTexture[0]);
00071 //    this->rectangleSprite.setTexture(this->rectangleTexture[0]);
00072 }
```

7.2.3.6 setMid()

```
void Arrow::setMid ( )
```

Definition at line 111 of file Arrow.cpp.

```
00111                     {
00112     if (this->hasSetMid) return;
00113     this->hasSetMid = true;
00114     this->points[0] = sf::Vector2f(
00115             (this->points[0].x + this->points[1].x) / 2.0f,
00116             (this->points[0].y + this->points[1].y) / 2.0f
00117     );
00118     this->setStart(this->points[0], false);
00119 }
```

7.2.3.7 setPositions()

```
void Arrow::setPositions (
            sf::Vector2f start,
            sf::Vector2f end,
            bool needSetMid )
```

Definition at line 74 of file Arrow.cpp.

```
                                                                                {
    this->points[0] = start;
    this->points[1] = end;
    if (needSetMid) {
        this->hasSetMid = false;
        this->setMid();
    }
    else {
        this->arrowSprite.setPosition(this->points[0]);
        this->autoScale();
        this->autoRotate();
    }
}
```

7.2.3.8 setStart()

```
void Arrow::setStart (
            sf::Vector2f start,
            bool needSetMid )
```

Definition at line 121 of file Arrow.cpp.

```
                                                      {
    this->setPositions(start, this->points[1], needSetMid);
}
```

7.2.3.9 show()

```
void Arrow::show ( )
```

Definition at line 131 of file Arrow.cpp.

```
                 {
    sf::Color tmp = this->arrowSprite.getColor();
    tmp.a = 255;
    this->arrowSprite.setColor(tmp);
}
```

7.2.3.10 toggleActiveColor()

```
void Arrow::toggleActiveColor ( )
```

Definition at line 64 of file Arrow.cpp.

```
                             {
    this->arrowSprite.setTexture(this->arrowTexture[1]);
//    this->rectangleSprite.setTexture(this->rectangleTexture[1]);
}
```

7.2.4 Member Data Documentation

7.2.4.1 arrowSprite

```
sf::Sprite Arrow::arrowSprite [protected]
```

Definition at line 17 of file Arrow.hpp.

7.2.4.2 arrowTexture

```
sf::Texture Arrow::arrowTexture[2] [protected]
```

Definition at line 16 of file Arrow.hpp.

7.2.4.3 hasSetMid

```
bool Arrow::hasSetMid [protected]
```

Definition at line 19 of file Arrow.hpp.

7.2.4.4 length

```
float Arrow::length [protected]
```

Definition at line 18 of file Arrow.hpp.

7.2.4.5 points

```
sf::Vector2f Arrow::points[2] [protected]
```

Definition at line 15 of file Arrow.hpp.

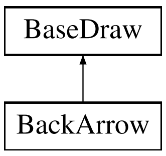
The documentation for this class was generated from the following files:

- include/draw/Arrow.hpp
- include/draw/Arrow.cpp

7.3 BackArrow Class Reference

```
#include <BackArrow.hpp>
```

Inheritance diagram for BackArrow:



Public Member Functions

- BackArrow (sf::RenderWindow *window, sf::Vector2f start, sf::Vector2f end)
- void render () override
- void autoScale ()
- void autoRotate ()
- void toggleActiveColorNode ()
- void resetColor ()
- void setPosition (sf::Vector2f start, sf::Vector2f end)
- void show ()
- void hide ()

Public Member Functions inherited from BaseDraw

- BaseDraw (sf::RenderWindow *window)
- virtual void render ()=0

Additional Inherited Members

Protected Attributes inherited from BaseDraw

- sf::RenderWindow * window

7.3.1 Detailed Description

Definition at line 11 of file BackArrow.hpp.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 BackArrow()

```
BackArrow::BackArrow (
            sf::RenderWindow * window,
            sf::Vector2f start,
            sf::Vector2f end )
```

Definition at line 7 of file BackArrow.cpp.

```
00007                                                                                  : BaseDraw(window) {
00008     this->isShow = false;
00009 
00010     this->points[0] = end;
00011     this->points[1] = start;
00012     this->points[2] = sf::Vector2f(
00013             this->points[0].x,
00014             this->points[0].y - constants::NodeInfo::offsetX
00015     );
00016     this->points[3] = sf::Vector2f(
00017             this->points[1].x,
00018             this->points[2].y
00019     );
00020     this->arrow = new Arrow(window, this->points[2], this->points[0]);
00021 
00022     this->rectangleTexture[0].loadFromFile("../assets/rectangle/rectangle_black.png");
00023     this->rectangleTexture[1].loadFromFile("../assets/rectangle/rectangle_orange.png");
00024 
00025     this->rectangleTexture[0].setRepeated(true);
00026     this->rectangleTexture[1].setRepeated(true);
00027 
00028     sf::Vector2i topLeftCorner = sf::Vector2i(
00029             static_cast<int>(this->rectangleTexture[0].getSize().x / 2.0 -
      constants::Arrow::sizeRectangle.x / 2.0),
00030             static_cast<int>(this->rectangleTexture[0].getSize().y / 2.0 -
      constants::Arrow::sizeRectangle.y / 2.0)
00031     );
00032     for (auto & rectangleSprite : this->rectangleSprites) {
00033         rectangleSprite.setTexture(this->rectangleTexture[0]);
00034         rectangleSprite.setTextureRect(sf::IntRect(
00035                 topLeftCorner.x,
00036                 topLeftCorner.y,
00037                 constants::Arrow::sizeRectangle.x,
00038                 constants::Arrow::sizeRectangle.y
00039         ));
00040     }
00041 
00042     this->setPosition(start, end);
00043 }
```

7.3.3 Member Function Documentation

7.3.3.1 autoRotate()

```
void BackArrow::autoRotate ( )
```

Definition at line 123 of file BackArrow.cpp.

```
00123                            {
00124     sf::Vector2f vector2point = this->points[3] - this->points[2];
00125     float angle = atan2f(vector2point.y, vector2point.x) * 180.0f / (float)M_PI;
00126     this->rectangleSprites[0].setRotation(angle);
00127     vector2point = this->points[1] - this->points[3];
00128     angle = atan2f(vector2point.y, vector2point.x) * 180.0f / (float)M_PI;
00129     this->rectangleSprites[1].setRotation(angle);
00130 }
```

7.3.3.2 autoScale()

```
void BackArrow::autoScale ( )
```

Definition at line 93 of file BackArrow.cpp.

```
                        {
    float length = sqrtf(
            powf(this->points[3].x - this->points[2].x, 2) + powf(this->points[3].y -
   this->points[2].y, 2)
            );
    this->rectangleSprites[0].setScale(
            length / this->rectangleSprites[0].getLocalBounds().width,
            constants::Arrow::defaultScaleRectangle.y
    );
    length = sqrtf(
            powf(this->points[3].x - this->points[1].x, 2) + powf(this->points[3].y -
   this->points[1].y, 2)
            );
    this->rectangleSprites[1].setScale(
            length / this->rectangleSprites[1].getLocalBounds().width,
            constants::Arrow::defaultScaleRectangle.y
    );
    this->rectangleSprites[0].setOrigin(
            this->rectangleSprites[0].getLocalBounds().width / 2.0f,
            0
    );
    this->rectangleSprites[1].setOrigin(
            this->rectangleSprites[1].getLocalBounds().width,
            this->rectangleSprites[1].getLocalBounds().height / 2.0f
    );
    this->rectangleSprites[0].setPosition(
            (this->points[3].x + this->points[2].x) / 2.0f,
            (this->points[3].y + this->points[2].y) / 2.0f
            );
    this->rectangleSprites[1].setPosition(this->points[1]);
}
```

7.3.3.3 hide()

```
void BackArrow::hide ( )
```

Definition at line 57 of file BackArrow.cpp.

```
                   {
    this->isShow = false;
}
```

7.3.3.4 render()

```
void BackArrow::render ( )  [override], [virtual]
```

Implements BaseDraw.

Definition at line 45 of file BackArrow.cpp.

```
                     {
    if (this->isShow) {
        this->window->draw(this->rectangleSprites[0]);
        this->window->draw(this->rectangleSprites[1]);
        this->arrow->render();
    }
}
```

7.3.3.5 resetColor()

```
void BackArrow::resetColor ( )
```

Definition at line 67 of file BackArrow.cpp.

```
                       {
    this->rectangleSprites[0].setTexture(this->rectangleTexture[0]);
    this->rectangleSprites[1].setTexture(this->rectangleTexture[0]);
    this->arrow->resetColor();
}
```

7.3.3.6 setPosition()

```
void BackArrow::setPosition (
            sf::Vector2f start,
            sf::Vector2f end )
```

Definition at line 73 of file BackArrow.cpp.

```
                                                                 {
    this->points[0] = end;
    this->points[1] = start;
    if (end == start) {
        this->hide();
        return;
    }
    this->points[2] = sf::Vector2f(
            this->points[0].x,
            this->points[0].y - constants::NodeInfo::offsetX
    );
    this->points[3] = sf::Vector2f(
            this->points[1].x,
            this->points[2].y
    );
    this->arrow->setPositions(this->points[2], this->points[0], false);
    this->autoRotate();
    this->autoScale();
}
```

7.3.3.7 show()

```
void BackArrow::show ( )
```

Definition at line 53 of file BackArrow.cpp.

```
                 {
    this->isShow = true;
}
```

7.3.3.8 toggleActiveColorNode()

```
void BackArrow::toggleActiveColorNode ( )
```

Definition at line 61 of file BackArrow.cpp.

```
                                  {
    this->rectangleSprites[0].setTexture(this->rectangleTexture[1]);
    this->rectangleSprites[1].setTexture(this->rectangleTexture[1]);
    this->arrow->toggleActiveColor();
}
```

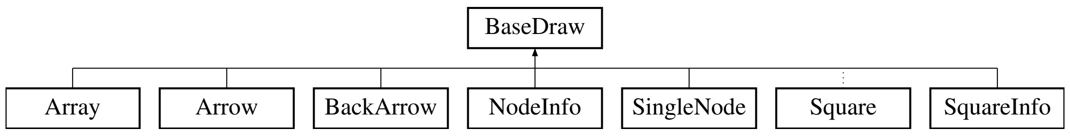
The documentation for this class was generated from the following files:

- include/draw/BackArrow.hpp
- include/draw/BackArrow.cpp

7.4 BaseDraw Class Reference

`#include <BaseDraw.hpp>`

Inheritance diagram for BaseDraw:



Public Member Functions

- BaseDraw (sf::RenderWindow ∗window)
- virtual void render ()=0

Protected Attributes

- sf::RenderWindow ∗ window

7.4.1 Detailed Description

Definition at line 10 of file BaseDraw.hpp.

7.4.2 Constructor & Destructor Documentation

7.4.2.1 BaseDraw()

```
BaseDraw::BaseDraw (
                sf::RenderWindow ∗ window )  [explicit]
```

Definition at line 7 of file BaseDraw.cpp.

```
00007                                              {
00008     this->window = window;
00009 }
```

7.4.3 Member Function Documentation

7.4.3.1 render()

```
virtual void BaseDraw::render ( )  [pure virtual]
```

Implemented in Array, Arrow, BackArrow, NodeInfo, SingleNode, Square, and SquareInfo.

7.4.4 Member Data Documentation

7.4.4.1 window

```
sf::RenderWindow* BaseDraw::window [protected]
```

Definition at line 12 of file BaseDraw.hpp.

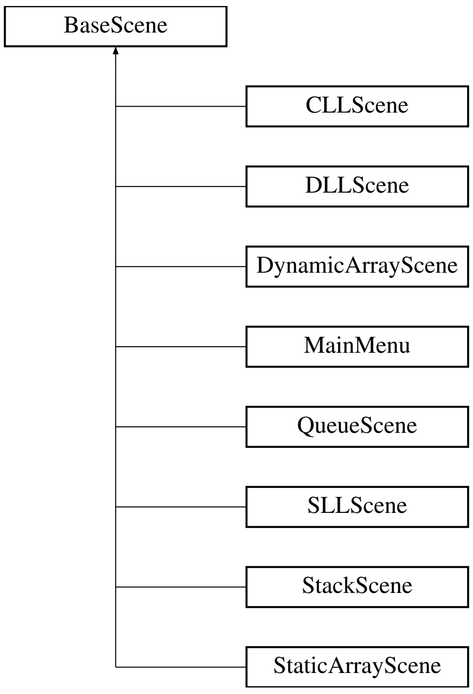
The documentation for this class was generated from the following files:

- include/draw/BaseDraw.hpp
- include/draw/BaseDraw.cpp

7.5 BaseScene Class Reference

```
#include <BaseScene.hpp>
```

Inheritance diagram for BaseScene:



Public Member Functions

- BaseScene (sf::RenderWindow ∗window)
- void createModeButton (sf::Vector2f position, std::string textString)
- virtual void pollEvent (sf::Event event, sf::Vector2f mousePosView)=0
- virtual void update ()=0
- virtual void render ()=0

Public Attributes

- Button ∗ modeButton {}
- bool isMenuOpen {}
- bool isDemoCodeOpen {}

Protected Member Functions

- void setWindow (sf::RenderWindow ∗window)

Protected Attributes

- sf::RenderWindow ∗ window {}
- ControlMenu ∗ controlMenu

7.5.1 Detailed Description

Definition at line 12 of file BaseScene.hpp.

7.5.2 Constructor & Destructor Documentation

7.5.2.1 BaseScene()

```
BaseScene::BaseScene (
              sf::RenderWindow * window )  [explicit]
```

Definition at line 26 of file BaseScene.cpp.

```
                                          {
    this->setWindow(window);
    this->isMenuOpen = false;
    this->isDemoCodeOpen = false;

    this->controlMenu = new ControlMenu(this->window);
}
```

7.5.3 Member Function Documentation

7.5.3.1 createModeButton()

```
void BaseScene::createModeButton (
            sf::Vector2f position,
            std::string textString )
```

Definition at line 11 of file BaseScene.cpp.

```
                                                                    {
    this->modeButton = new Button(
            this->window,
            position,
            constants::modeButtonSize,
            textString,
            textString,
            constants::sizeTextModeButton,
            sf::Color::Black,
            constants::normalGray,
            constants::hoverGray,
            constants::clickGray
            );
}
```

7.5.3.2 pollEvent()

```
virtual void BaseScene::pollEvent (
            sf::Event event,
            sf::Vector2f mousePosView )  [pure virtual]
```

Implemented in CLLScene, DLLScene, DynamicArrayScene, MainMenu, QueueScene, SLLScene, StackScene, and StaticArrayScene.

7.5.3.3 render()

```
virtual void BaseScene::render ( )  [pure virtual]
```

Implemented in CLLScene, DLLScene, DynamicArrayScene, MainMenu, QueueScene, SLLScene, StackScene, and StaticArrayScene.

7.5.3.4 setWindow()

```
void BaseScene::setWindow (
            sf::RenderWindow * window )  [protected]
```

Definition at line 7 of file BaseScene.cpp.

```
                                            {
    this->window = window;
}
```

7.5.3.5 update()

```
virtual void BaseScene::update ( )  [pure virtual]
```

Implemented in CLLScene, DLLScene, DynamicArrayScene, MainMenu, QueueScene, SLLScene, StackScene, and StaticArrayScene.

7.5.4 Member Data Documentation

7.5.4.1 controlMenu

```
ControlMenu* BaseScene::controlMenu  [protected]
```

Definition at line 15 of file BaseScene.hpp.

7.5.4.2 isDemoCodeOpen

```
bool BaseScene::isDemoCodeOpen {}
```

Definition at line 20 of file BaseScene.hpp.

7.5.4.3 isMenuOpen

```
bool BaseScene::isMenuOpen {}
```

Definition at line 20 of file BaseScene.hpp.

7.5.4.4 modeButton

```
Button* BaseScene::modeButton {}
```

Definition at line 19 of file BaseScene.hpp.

7.5.4.5 window

```
sf::RenderWindow* BaseScene::window {}  [protected]
```

Definition at line 14 of file BaseScene.hpp.

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

- include/libScene/BaseScene.hpp
- include/libScene/BaseScene.cpp

7.6 Button Class Reference

```
#include <button.hpp>
```

Public Member Functions

- Button ()
- Button (sf::RenderWindow *window, sf::Vector2f position, sf::Vector2f size, std::string textString, std::string changedTextString, int textSize, sf::Color textColor, sf::Color color, sf::Color hoverColor, sf::Color clickColor)
- bool pollEvent (sf::Vector2f mousePosView)
- void update ()
- void render ()
- void setColor (sf::Color _color)
- std::string getTextString () const
- sf::Vector2f getPosition () const
- sf::Vector2f getSize () const
- bool checkClicked () const

7.6.1 Detailed Description

Definition at line 12 of file button.hpp.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 Button() `[1/2]`

```
Button::Button ( )
```

7.6.2.2 Button() [2/2]

```
Button::Button (
             sf::RenderWindow * window,
             sf::Vector2f position,
             sf::Vector2f size,
             std::string textString,
             std::string changedTextString,
             int textSize,
             sf::Color textColor,
             sf::Color color,
             sf::Color hoverColor,
             sf::Color clickColor )
```

7.6.3 Member Function Documentation

7.6.3.1 checkClicked()

```
bool Button::checkClicked ( ) const
```

7.6.3.2 getPosition()

```
sf::Vector2f Button::getPosition ( ) const
```

7.6.3.3 getSize()

```
sf::Vector2f Button::getSize ( ) const
```

7.6.3.4 getTextString()

```
std::string Button::getTextString ( ) const
```

7.6.3.5 pollEvent()

```
bool Button::pollEvent (
             sf::Vector2f mousePosView )
```

7.6.3.6 render()

```
void Button::render ( )
```

7.6.3.7 setColor()

```
void Button::setColor (
             sf::Color _color )
```

7.6.3.8 update()

```
void Button::update ( )
```

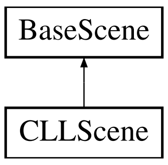
The documentation for this class was generated from the following file:

- include/stuff/button.hpp

7.7 CLLScene Class Reference

```
#include <CLLScene.hpp>
```

Inheritance diagram for CLLScene:



Public Member Functions

- CLLScene (sf::RenderWindow ∗window)
- void reset ()
- void pollEvent (sf::Event event, sf::Vector2f mousePosView) override
- void update () override
- void render () override
- std::vector< EventAnimation > addModeEvents (int chosenNode)
- std::vector< EventAnimation > deleteModeEvents (int chosenNode)
- std::vector< EventAnimation > updateModeEvents (int chosenNode)
- std::vector< EventAnimation > searchModeEvents (int chosenNode)

Public Member Functions inherited from BaseScene

- BaseScene (sf::RenderWindow ∗window)
- void createModeButton (sf::Vector2f position, std::string textString)
- virtual void pollEvent (sf::Event event, sf::Vector2f mousePosView)=0
- virtual void update ()=0
- virtual void render ()=0

Additional Inherited Members

Public Attributes inherited from BaseScene

- Button ∗ modeButton {}
- bool isMenuOpen {}
- bool isDemoCodeOpen {}

Protected Member Functions inherited from BaseScene

- void setWindow (sf::RenderWindow ∗window)

Protected Attributes inherited from BaseScene

- sf::RenderWindow ∗ window {}
- ControlMenu ∗ controlMenu

7.7.1 Detailed Description

Definition at line 12 of file CLLScene.hpp.

7.7.2 Constructor & Destructor Documentation

7.7.2.1 CLLScene()

```
CLLScene::CLLScene (
            sf::RenderWindow * window )   [explicit]
```

Definition at line 7 of file CLLScene.cpp.

```
                                        : BaseScene(window) {
    this->init();
}
```

7.7.3 Member Function Documentation

7.7.3.1 addModeEvents()

```
std::vector< EventAnimation > CLLScene::addModeEvents (
                int chosenNode )
```

Definition at line 143 of file CLLScene.cpp.

```
00143                                                                          {
00144     this->linkedList->resetEvents();
00145     if (chosenNode < 0 || chosenNode > this->linkedList->getSize())
00146         return {};
00147 
00148     this->linkedList->initHighlighter(
00149             constants::Highlighter::SLL::CODES_PATH[0].second,
00150             constants::Highlighter::SLL::CODES_PATH[0].first
00151     );
00152 
00153     std::vector<EventAnimation> events;
00154     EventAnimation event;
00155 
00156     if (chosenNode) {
00157         event.titleNodes = {
00158                 {0,          "head"},
00159                 {chosenNode, "temp"}
00160         };
00161         event.indexBackArrow.second = 0;
00162     }
00163     else {
00164         event.titleNodes.emplace_back(chosenNode, "temp");
00165         if (this->linkedList->getSize()) {
00166             event.titleNodes.emplace_back(1, "head");
00167             event.indexBackArrow.second = 1;
00168         }
00169     }
00170     event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00171     if (chosenNode && chosenNode == this->linkedList->getSize())
00172         event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00173     event.colorNodes.push_back(chosenNode);
00174     event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00175     event.indexBackArrow.first = this->linkedList->getSize();
00176     event.lines = {0};
00177 
00178     events.emplace_back(event);
00179 
00180     if (chosenNode == 0) {
00181         if (this->linkedList->getSize()) {
00182             event.reset();
00183             event.titleNodes = {
00184                     {1, "head"},
00185                     {chosenNode, "temp"}
00186             };
00187             event.colorNodes = std::vector<int>{0};
00188             event.colorArrows.emplace_back(0, NodeInfo::ArrowType::RIGHT);
00189             event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00190             event.isPrintNormal = true;
00191             event.indexBackArrow = {this->linkedList->getSize(), 1};
00192             event.lines = {1, 2};
00193 
00194             events.emplace_back(event);
00195         }
00196 
00197         event.reset();
00198         event.titleNodes.emplace_back(chosenNode, "head|temp");
00199         event.lines = {3};
00200         event.statusChosenNode = NodeInfo::StatusNode::InChain;
00201         event.indexBackArrow = {this->linkedList->getSize(), 0};
00202         events.emplace_back(event);
00203     } else {
00204         event.reset();
00205         event.titleNodes = {
00206                 {0, "head|current"},
00207                 {chosenNode, "temp"}
00208         };
00209         event.colorNodes.push_back(0);
00210         event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00211         if (chosenNode == this->linkedList->getSize())
00212             event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00213         event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00214         event.indexBackArrow = {this->linkedList->getSize(), 0};
00215         event.lines = {5};
00216 
00217         events.emplace_back(event);
00218 
00219         for (int i = 0; i < chosenNode; ++i) {
00220             event.reset();
00221             event.titleNodes = {
00222                     {0, "head"},
00223                     {chosenNode, "temp"},
00224                     {i, "current"}
00225             };
00226             event.colorNodes.push_back(i);
00227             event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00228             if (chosenNode == this->linkedList->getSize())
00229                 event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00230             event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00231             event.indexBackArrow = {this->linkedList->getSize(), 0};
00232             event.lines = {6};
00233 
00234             events.emplace_back(event);
00235 
00236             if (i == chosenNode - 1) break;
00237 
00238             event.reset();
00239             event.titleNodes = {
00240                     {0, "head"},
00241                     {chosenNode, "temp"},
00242                     {i, "current"}
00243             };
00244             event.colorNodes.push_back(i);
00245             event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
00246             event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00247             if (chosenNode == this->linkedList->getSize())
00248                 event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00249             event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00250             event.indexBackArrow = {this->linkedList->getSize(), 0};
00251             event.lines = {7};
00252 
00253             events.emplace_back(event);
00254         }
00255 
00256         if (chosenNode != this->linkedList->getSize()) {
00257             event.reset();
00258             event.titleNodes = {
00259                     {0, "head"},
00260                     {chosenNode, "temp"},
00261                     {chosenNode - 1, "current"}
00262             };
00263             event.colorNodes.push_back(chosenNode);
00264             event.colorArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00265             event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00266             event.isPrintNormal = true;
00267             event.indexBackArrow = {this->linkedList->getSize(), 0};
00268             event.lines = {8};
00269 
00270             events.emplace_back(event);
00271         }
00272 
00273         event.reset();
00274         event.titleNodes = {
00275                 {0, "head"},
00276                 {chosenNode, "temp"}
00277         };
00278         event.statusChosenNode = NodeInfo::StatusNode::InChain;
00279         event.indexBackArrow = {this->linkedList->getSize(), 0};
00280         event.lines = {9};
00281 
00282         events.emplace_back(event);
00283     }
00284 
00285     return events;
00286 }
```

7.7.3.2 deleteModeEvents()

```
std::vector< EventAnimation > CLLScene::deleteModeEvents (
              int chosenNode )
```

Definition at line 288 of file CLLScene.cpp.

```
00288                                                                    {
00289     this->linkedList->resetEvents();
00290     if (chosenNode < 0 || chosenNode >= this->linkedList->getSize())
00291         return {};
00292 
```

```
    this->linkedList->initHighlighter(
            constants::Highlighter::SLL::CODES_PATH[1].second,
            constants::Highlighter::SLL::CODES_PATH[1].first
    );

    std::vector<EventAnimation> events;
    EventAnimation event;

    if (!chosenNode) {
        event.titleNodes.emplace_back(chosenNode, "head|temp");
        event.colorNodes.push_back(chosenNode);
        event.statusChosenNode = NodeInfo::StatusNode::InChain;
        event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
        event.lines = {0, 1};

        events.emplace_back(event);

        if (this->linkedList->getSize() > 1) {
            event.reset();
            event.titleNodes = {
                    {chosenNode, "temp"},
                    {1, "head"}
            };
            event.colorNodes.push_back(1);
            event.colorArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
            event.isPrintNormal = true;
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.indexBackArrow = {this->linkedList->getSize() - 1, 1};
            event.lines = {2};

            events.emplace_back(event);
        }

        event.reset();
        event.titleNodes.emplace_back(1, "head");
        event.statusChosenNode = NodeInfo::StatusNode::Visible;
        event.indexBackArrow = {this->linkedList->getSize() - 1, 1};
        event.lines = {3};

        events.emplace_back(event);
    } else {
        event.reset();
        event.titleNodes.emplace_back(0, "head|current");
        event.colorNodes.push_back(0);
        event.statusChosenNode = NodeInfo::StatusNode::InChain;
        event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
        event.lines = {5};

        events.emplace_back(event);

        for (int i = 0; i < chosenNode; ++i) {
            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {i, "current"}
            };
            event.colorNodes.push_back(i);
            event.statusChosenNode = NodeInfo::StatusNode::InChain;
            event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
            event.lines = {6};

            events.emplace_back(event);

            if (i == chosenNode - 1) break;

            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {i, "current"}
            };
            event.colorNodes.push_back(i);
            event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::InChain;
            event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
            event.lines = {7};

            events.emplace_back(event);
        }

        event.reset();
        event.titleNodes = {
                {0, "head"},
                {chosenNode, "temp"},
                {chosenNode - 1, "current"}
        };
        event.colorNodes.push_back(chosenNode);
        event.colorArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
        event.statusChosenNode = NodeInfo::StatusNode::InChain;
        event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
        event.lines = {8};

        events.emplace_back(event);

        if (chosenNode != this->linkedList->getSize() - 1) {
            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {chosenNode, "temp"},
                    {chosenNode - 1, "current"}
            };
            event.colorNodes.push_back(chosenNode);
            event.colorArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
            event.isPrintNormal = true;
            event.lines = {9};

            events.emplace_back(event);

            event.reset();
            event.titleNodes.emplace_back(0, "head");
            event.statusChosenNode = NodeInfo::StatusNode::Visible;
            event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
            event.lines = {10};

            events.emplace_back(event);
        } else {
            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {chosenNode, "temp"},
                    {chosenNode - 1, "current"}
            };
            event.colorNodes.push_back(chosenNode);
            event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.indexBackArrow = {chosenNode - 1, 0};
            event.lines = {9};

            events.emplace_back(event);

            event.reset();
            event.titleNodes.emplace_back(0, "head");
            event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::Visible;
            event.indexBackArrow = {chosenNode - 1, 0};
            event.lines = {10};

            events.emplace_back(event);
        }
    }

    return events;
}
```

7.7.3.3 pollEvent()

```
void CLLScene::pollEvent (
            sf::Event event,
            sf::Vector2f mousePosView )  [override], [virtual]
```

Implements BaseScene.

Definition at line 11 of file CLLScene.cpp.

```
                                                                {
    if (this->isMenuOpen)
        this->menu->pollEvents(event, mousePosView);

    this->controlMenu->pollEvents(event, mousePosView);
}
```

7.7.3.4 render()

```
void CLLScene::render ( )  [override], [virtual]
```

Implements BaseScene.

Definition at line 123 of file CLLScene.cpp.

```
                        {
    if (this->isMenuOpen)
        this->menu->render();

    if (this->isDemoCodeOpen)
        this->linkedList->renderHighlighter();

    this->controlMenu->render();
    this->linkedList->render();
}
```

7.7.3.5 reset()

```
void CLLScene::reset ( )
```

Definition at line 139 of file CLLScene.cpp.

```
                       {
    this->menu->resetActiveOptionMenu();
}
```

7.7.3.6 searchModeEvents()

```
std::vector< EventAnimation > CLLScene::searchModeEvents (
               int chosenNode )
```

Definition at line 506 of file CLLScene.cpp.

```
                                                                   {
    this->linkedList->resetEvents();
    this->linkedList->initHighlighter(
            constants::Highlighter::SLL::CODES_PATH[3].second,
            constants::Highlighter::SLL::CODES_PATH[3].first
    );

    std::vector<EventAnimation> events;
    EventAnimation event;

    event.titleNodes.emplace_back(0, "head|current");
    event.colorNodes.push_back(0);
    event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
    event.lines = {0};

    events.emplace_back(event);

    for (int i = 0; i <= chosenNode; ++i) {
        if (i == chosenNode && chosenNode == this->linkedList->getSize())
            break;

        event.reset();
        event.titleNodes = {
                {0, "head"},
                {i, "current"}
        };
        event.colorNodes.push_back(i);
        event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
        event.lines = {1};

        events.emplace_back(event);

        if (i == chosenNode) break;
```

```
00539
00540         event.reset();
00541         event.titleNodes = {
00542                 {0, "head"},
00543                 {i, "current"}
00544         };
00545         event.colorNodes.push_back(i);
00546         event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
00547         event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
00548         event.lines = {4};
00549
00550         events.emplace_back(event);
00551     }
00552
00553     if (chosenNode == this->linkedList->getSize()) {
00554         event.reset();
00555         event.titleNodes.emplace_back(0, "head");
00556         event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
00557         event.lines = {5};
00558
00559         events.emplace_back(event);
00560     } else {
00561         event.reset();
00562         event.titleNodes = {
00563                 {0, "head"},
00564                 {chosenNode, "current"}
00565         };
00566         event.colorNodes.push_back(chosenNode);
00567         event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
00568         event.lines = {2, 3};
00569
00570         events.emplace_back(event);
00571     }
00572
00573     return events;
00574 }
```

7.7.3.7 update()

```
void CLLScene::update ( ) [override], [virtual]
```

Implements BaseScene.

Definition at line 18 of file CLLScene.cpp.

```
00018                       {
00019     if (this->isMenuOpen) {
00020         this->menu->update();
00021
00022         constants::MenuLinkedList::Button status = this->menu->getActiveOptionMenu();
00023         constants::MenuLinkedList::CreateMode::Button createMode;
00024         switch (status){
00025             case constants::MenuLinkedList::Button::CREATE_BUTTON:
00026                 createMode = this->menu->getActiveCreateMode();
00027                 if (createMode == constants::MenuLinkedList::CreateMode::Button::RANDOM_BUTTON) {
00028                     if (this->menu->createModeValue[0] == "None")
00029                         break;
00030                     if (this->menu->createModeValue[0].empty())
00031                         this->menu->createModeValue[0] = "0";
00032                     int size = std::stoi(this->menu->createModeValue[0]);
00033                     this->linkedList->createLinkedList(size);
00034                 } else if (createMode ==
      constants::MenuLinkedList::CreateMode::Button::DEFINED_LIST_BUTTON) {
00035                     if (this->menu->createModeValue[1] == "None")
00036                         break;
00037                     std::vector<std::string> values;
00038                     std::string value = this->menu->createModeValue[1];
00039                     std::stringstream ss(value);
00040                     std::string token;
00041                     while (std::getline(ss, token, ',')) {
00042                         values.push_back(token);
00043                     }
00044                     this->linkedList->createLinkedList(values);
00045                 } else if (createMode == constants::MenuLinkedList::CreateMode::Button::FILE_BUTTON) {
00046                     if (this->menu->createModeValue[2] == "None")
00047                         break;
00048                     std::vector<std::string> values;
00049                     std::string value = this->menu->createModeValue[2];
00050                     std::stringstream ss(value);
```

```
                    std::string token;
                    while (std::getline(ss, token, ','))
                        values.push_back(token);
                    this->linkedList->createLinkedList(values);
                    this->menu->createModeValue[2] = "None";
                }
                this->controlMenu->reset();
                break;
            case constants::MenuLinkedList::Button::ADD_BUTTON:
                if (this->menu->addModeValue[0] == "None" || this->menu->addModeValue[1] == "None" ||
this->menu->addModeValue[0].empty())
                    break;

                this->linkedList->addNode(
                        std::stoi(this->menu->addModeValue[0]),
                        this->menu->addModeValue[1],
                        this->addModeEvents(std::stoi(this->menu->addModeValue[0]))
                );

                std::cout « "Add: " « this->menu->addModeValue[0] « " " « this->menu->addModeValue[1]
« std::endl;
                this->menu->addModeValue[0] = this->menu->addModeValue[1] = "None";
                this->controlMenu->reset();
                break;
            case constants::MenuLinkedList::Button::DELETE_BUTTON:
                if (this->menu->deleteModeValue == "None" || this->menu->deleteModeValue.empty())
                    break;

                this->linkedList->deleteNode(
                        std::stoi(this->menu->deleteModeValue),
                        this->deleteModeEvents(std::stoi(this->menu->deleteModeValue))
                );

                std::cout « "Delete: " « this->menu->deleteModeValue « std::endl;
                this->menu->deleteModeValue = "None";
                this->controlMenu->reset();
                break;
            case constants::MenuLinkedList::Button::UPDATE_BUTTON:
                if (this->menu->updateModeValue[0] == "None" || this->menu->updateModeValue[1] ==
"None" || this->menu->updateModeValue[0].empty())
                    break;

                this->linkedList->updateNode(
                        std::stoi(this->menu->updateModeValue[0]),
                        this->menu->updateModeValue[1],
                        this->updateModeEvents(std::stoi(this->menu->updateModeValue[0]))
                );

                std::cout « "Update: " « this->menu->updateModeValue[0] « " " «
this->menu->updateModeValue[1] « std::endl;
                this->menu->updateModeValue[0] = this->menu->updateModeValue[1] = "None";
                this->controlMenu->reset();
                break;
            case constants::MenuLinkedList::Button::SEARCH_BUTTON:
                if (this->menu->searchModeValue == "None" || this->menu->searchModeValue.empty())
                    break;

                this->linkedList->searchNode(

this->searchModeEvents(this->linkedList->findValue(this->menu->searchModeValue))
                );

                std::cout « "Search: " « this->menu->searchModeValue « std::endl;
                this->menu->searchModeValue = "None";
                this->controlMenu->reset();
                break;
        }
    }

    this->controlMenu->update();

    this->linkedList->processControlMenu(this->controlMenu->getStatus());
    this->linkedList->setSpeed(this->controlMenu->getSpeed());

    this->linkedList->update();
}
```

7.7.3.8 updateModeEvents()

```
std::vector< EventAnimation > CLLScene::updateModeEvents (
        int chosenNode )
```

Definition at line 438 of file CLLScene.cpp.

```
                                                                {
    this->linkedList->resetEvents();
    if (chosenNode < 0 || chosenNode >= this->linkedList->getSize())
        return {};

    this->linkedList->initHighlighter(
            constants::Highlighter::SLL::CODES_PATH[2].second,
            constants::Highlighter::SLL::CODES_PATH[2].first
    );

    std::vector<EventAnimation> events;
    EventAnimation event;

    event.titleNodes.emplace_back(0, "head|current");
    event.colorNodes.push_back(0);
    event.isPrintPreVal = true;
    event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
    event.lines = {0};

    events.emplace_back(event);

    if (chosenNode) {
        for (int i = 0; i <= chosenNode; ++i) {
            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {i, "current"}
            };
            event.colorNodes.push_back(i);
            event.isPrintPreVal = true;
            event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
            event.lines = {1};

            events.emplace_back(event);

            if (i == chosenNode) break;

            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {i, "current"}
            };
            event.colorNodes.push_back(i);
            event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
            event.isPrintPreVal = true;
            event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
            event.lines = {2};

            events.emplace_back(event);
        }
    }

    event.reset();
    if (chosenNode == 0)
        event.titleNodes.emplace_back(0, "head|current");
    else
        event.titleNodes = {
                {0, "head"},
                {chosenNode, "current"}
        };
    event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
    event.lines = {3};

    events.emplace_back(event);

    return events;
}
```

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

- include/libScene/CLLScene.hpp
- include/libScene/CLLScene.cpp

7.8 ControlMenu Class Reference

```
#include <ControlMenu.hpp>
```

Public Types

- enum class StatusCode {
  PREVIOUS , PAUSE , PLAY , NEXT ,
  None }

Public Member Functions

- ControlMenu (sf::RenderWindow ∗window)
- ∼ControlMenu ()=default
- void pollEvents (sf::Event event, sf::Vector2f mousePosView)
- void update ()
- void render ()
- void reset ()
- ControlMenu::StatusCode getStatus ()
- float getSpeed () const

Public Attributes

- enum ControlMenu::StatusCode status

7.8.1 Detailed Description

Definition at line 13 of file ControlMenu.hpp.

7.8.2 Member Enumeration Documentation

7.8.2.1 StatusCode

```
enum class ControlMenu::StatusCode  [strong]
```

Enumerator

PREVIOUS	
PAUSE	
PLAY	
NEXT	
None	

Definition at line 23 of file ControlMenu.hpp.

```
                                {
        PREVIOUS,
        PAUSE,
        PLAY,
        NEXT,
        None
    } status;
```

7.8.3 Constructor & Destructor Documentation

7.8.3.1 ControlMenu()

```
ControlMenu::ControlMenu (
            sf::RenderWindow * window )  [explicit]
```

Definition at line 7 of file ControlMenu.cpp.

```
                                            {
    this->window = window;

    for (int i = 0; i < constants::ControlMenu::BUTTON_COUNT; ++i) {
        buttons[i] = new Button(
                this->window,
                constants::ControlMenu::buttonPos[i],
                constants::ControlMenu::buttonSize,
                constants::ControlMenu::BUTTON_NAMES[i],
                constants::ControlMenu::BUTTON_NAMES[i],
                constants::ControlMenu::BUTTON_NAME_SIZE,
                sf::Color::Black,
                constants::normalGray,
                constants::hoverGray,
                constants::clickGray
                );
    }

    this->font.loadFromFile(constants::fontPath);
    this->textSpeed.setFont(font);
    this->textSpeed.setString(to_string_with_precision(this->speed));
    this->textSpeed.setCharacterSize(constants::ControlMenu::TEXT_SIZE);
    this->textSpeed.setFillColor(sf::Color::Black);
    this->textSpeed.setOrigin(
            this->textSpeed.getLocalBounds().width / 2.0f,
            this->textSpeed.getLocalBounds().height / 2.0f
            );
    this->textSpeed.setPosition(
            constants::ControlMenu::buttonPos[3].x + constants::ControlMenu::buttonSize.x * 2,
            constants::ControlMenu::buttonPos[3].y + constants::ControlMenu::buttonSize.y / 2.0f
            );

    this->status = StatusCode::None;
    this->speed = 1;
}
```

7.8.3.2 ∼ControlMenu()

```
ControlMenu::∼ControlMenu ( )  [default]
```

7.8.4 Member Function Documentation

7.8.4.1 getSpeed()

```
float ControlMenu::getSpeed ( ) const
```

Definition at line 96 of file ControlMenu.cpp.

```
                                  {
    return this->speed;
}
```

7.8.4.2 getStatus()

```
ControlMenu::StatusCode ControlMenu::getStatus ( )
```

Definition at line 89 of file ControlMenu.cpp.

```
00089                                          {
00090     ControlMenu::StatusCode temp = this->status;
00091     if (this->status == StatusCode::PREVIOUS || this->status == StatusCode::NEXT)
00092         this->status = StatusCode::PAUSE;
00093     return temp;
00094 }
```

7.8.4.3 pollEvents()

```
void ControlMenu::pollEvents (
              sf::Event event,
              sf::Vector2f mousePosView )
```

Definition at line 43 of file ControlMenu.cpp.

```
00043                                                                               {
00044     for (int i = 0; i < constants::ControlMenu::BUTTON_COUNT; ++i) {
00045         if (buttons[i]->pollEvent(mousePosView)) {
00046             switch (i) {
00047                 case 0:
00048                     this->status = StatusCode::PREVIOUS;
00049                     break;
00050                 case 1:
00051                     if (this->status == StatusCode::PLAY)
00052                         this->status = StatusCode::PAUSE;
00053                     else
00054                         this->status = StatusCode::PLAY;
00055                     break;
00056                 case 2:
00057                     this->status = StatusCode::NEXT;
00058                     break;
00059                 case 3:
00060                     if (this->speed > 0.25)
00061                         this->speed -= 0.25;
00062                     break;
00063                 case 4:
00064                     if (this->speed < 2)
00065                         this->speed += 0.25;
00066                     break;
00067                 default:
00068                     this->status = StatusCode::None;
00069                     break;
00070             }
00071         }
00072     }
00073 }
```

7.8.4.4 render()

```
void ControlMenu::render ( )
```

Definition at line 82 of file ControlMenu.cpp.

```
00082                        {
00083     for (auto &button : buttons) {
00084         button->render();
00085     }
00086     this->window->draw(this->textSpeed);
00087 }
```

7.8.4.5 reset()

```
void ControlMenu::reset ( )
```

Definition at line 100 of file ControlMenu.cpp.

```
                              {
    this->status = StatusCode::None;
}
```

7.8.4.6 update()

```
void ControlMenu::update ( )
```

Definition at line 75 of file ControlMenu.cpp.

```
                               {
    for (auto &button : buttons) {
        button->update();
    }
    this->textSpeed.setString(to_string_with_precision(this->speed));
}
```

7.8.5 Member Data Documentation

7.8.5.1 status

```
enum ControlMenu::StatusCode ControlMenu::status
```

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

- include/libScene/ControlMenu.hpp
- include/libScene/ControlMenu.cpp

7.9 CustomTextbox Class Reference

```
#include <CustomTextbox.hpp>
```

Public Member Functions

- CustomTextbox (sf::RenderWindow ∗window, sf::Vector2f position, int size, std::string titleString, int max←Length)
- ∼CustomTextbox ()=default
- void pollEvent (sf::Event event, sf::Vector2f mousePosView)
- void update ()
- void render ()
- std::string getTextString ()
- void resetInput ()

7.9.1 Detailed Description

Definition at line 11 of file CustomTextbox.hpp.

7.9.2 Constructor & Destructor Documentation

7.9.2.1 CustomTextbox()

```
CustomTextbox::CustomTextbox (
              sf::RenderWindow * window,
              sf::Vector2f position,
              int size,
              std::string titleString,
              int maxLength )
```

Definition at line 7 of file CustomTextbox.cpp.

```
                                                                            {
    this->window = window;
    this->titleString = std::move(_titleString);
    this->position = position;

    font.loadFromFile(constants::fontPath);
    this->title.setFont(font);
    this->title.setCharacterSize(size);
    this->title.setFillColor(sf::Color::Black);
    this->title.setString(this->titleString);
    this->title.setPosition(this->position);

    float width = this->title.findCharacterPos(this->titleString.size() - 1).x -
      this->title.findCharacterPos(0).x;

//    std::cout « width « ' ' « this->title.getString().getSize() « std::endl;

    this->maxLength = maxLength;

    this->textbox = new TextBox(
        this->window,
        sf::Vector2f(this->position.x + width + 10, this->position.y),
        20,
        sf::Color::Black,
        sf::Color::White,
        this->maxLength
        );

    this->goButton = new Button(
        this->window,
        sf::Vector2f(this->textbox->getBox().getPosition().x + this->textbox->getBox().getSize().x +
      10, this->position.y),
        constants::goButtonSize,
        "Go",
        "Go",
        20,
        sf::Color::Black,
        constants::normalGray,
        constants::hoverGray,
        constants::clickGray
        );

    this->isGoButtonClicked = false;
}
```

7.9.2.2 ∼CustomTextbox()

```
CustomTextbox::∼CustomTextbox ( )  [default]
```

7.9.3 Member Function Documentation

7.9.3.1 getTextString()

```
std::string CustomTextbox::getTextString ( )
```

Definition at line 70 of file CustomTextbox.cpp.

```
                                        {
    if (this->isGoButtonClicked) {
        this->isGoButtonClicked = false;
        return this->textbox->getTextString();
    }
    return "None";
}
```

7.9.3.2 pollEvent()

```
void CustomTextbox::pollEvent (
               sf::Event event,
               sf::Vector2f mousePosView )
```

Definition at line 51 of file CustomTextbox.cpp.

```
                                                                           {
    this->textbox->pollEvent(event);
    if (this->goButton->pollEvent(mousePosView)) {
        this->isGoButtonClicked = true;
//        std::cout « "Go button clicked!\n";
    }
}
```

7.9.3.3 render()

```
void CustomTextbox::render ( )
```

Definition at line 64 of file CustomTextbox.cpp.

```
                             {
    this->window->draw(this->title);
    this->textbox->render();
    this->goButton->render();
}
```

7.9.3.4 resetInput()

```
void CustomTextbox::resetInput ( )
```

Definition at line 78 of file CustomTextbox.cpp.

```
                                 {
    this->textbox->resetInput();
}
```

7.9.3.5 update()

```
void CustomTextbox::update ( )
```

Definition at line 59 of file CustomTextbox.cpp.

```
                         {
    this->textbox->update();
    this->goButton->update();
}
```

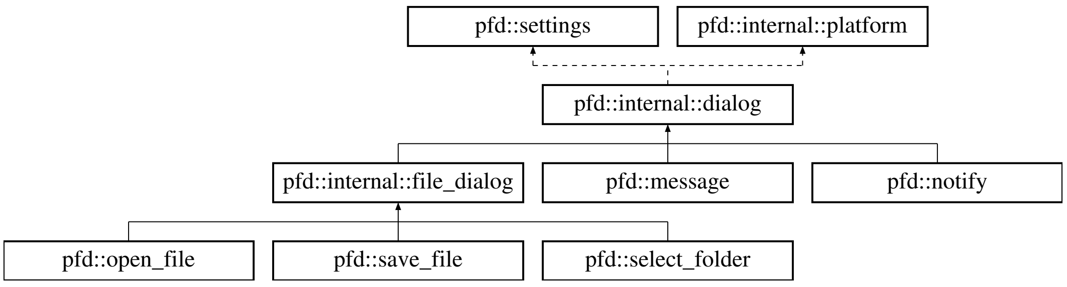
The documentation for this class was generated from the following files:

- include/stuff/CustomTextbox.hpp
- include/stuff/CustomTextbox.cpp

7.10 pfd::internal::dialog Class Reference

```
#include <FileDialog.h>
```

Inheritance diagram for pfd::internal::dialog:



Public Member Functions

- bool ready (int timeout=default_wait_timeout) const
- bool kill () const

Protected Member Functions

- dialog ()
- std::vector< std::string > desktop_helper () const
- std::string powershell_quote (std::string const &str) const
- std::string osascript_quote (std::string const &str) const
- std::string shell_quote (std::string const &str) const

Protected Member Functions inherited from pfd::settings

- settings (bool resync=false)
- bool check_program (std::string const &program)
- bool is_osascript () const
- bool is_zenity () const
- bool is_kdialog () const
- bool const & flags (flag in_flag) const
- bool & flags (flag in_flag)

Static Protected Member Functions

- static std::string buttons_to_name (choice _choice)
- static std::string get_icon_name (icon _icon)

Static Protected Member Functions inherited from pfd::settings

- static bool available ()
- static void verbose (bool value)
- static void rescan ()

Protected Attributes

- std::shared_ptr< executor > m_async

Additional Inherited Members

Protected Types inherited from pfd::settings

- enum class flag {
  is_scanned = 0 , is_verbose , has_zenity , has_matedialog ,
  has_qarma , has_kdialog , is_vista , max_flag }

7.10.1 Detailed Description

Definition at line 265 of file FileDialog.h.

7.10.2 Constructor & Destructor Documentation

7.10.2.1 dialog()

```
pfd::internal::dialog::dialog ( )  [inline], [explicit], [protected]
```

Definition at line 975 of file FileDialog.h.

```
00976             : m_async(std::make_shared<executor>())
00977     {
00978     }
```

7.10.3 Member Function Documentation

7.10.3.1 buttons_to_name()

```
std::string pfd::internal::dialog::buttons_to_name (
            choice _choice )  [inline], [static], [protected]
```

Definition at line 993 of file FileDialog.h.

```
    {
        switch (_choice)
        {
            case choice::ok_cancel: return "okcancel";
            case choice::yes_no: return "yesno";
            case choice::yes_no_cancel: return "yesnocancel";
            case choice::retry_cancel: return "retrycancel";
            case choice::abort_retry_ignore: return "abortretryignore";
                /* case choice::ok: */ default: return "ok";
        }
    }
```

7.10.3.2 desktop_helper()

```
std::vector< std::string > pfd::internal::dialog::desktop_helper ( ) const  [inline], [protected]
```

Definition at line 980 of file FileDialog.h.

```
    {
#if __APPLE__
        return { "osascript" };
#else
        return { flags(flag::has_zenity) ? "zenity"
                                        : flags(flag::has_matedialog) ? "matedialog"
                                                                      : flags(flag::has_qarma) ?
"qarma"
                                                                                                :
flags(flag::has_kdialog) ? "kdialog"
: "echo" };
#endif
    }
```

7.10.3.3 get_icon_name()

```
std::string pfd::internal::dialog::get_icon_name (
            icon _icon )  [inline], [static], [protected]
```

Definition at line 1006 of file FileDialog.h.

```
    {
        switch (_icon)
        {
            case icon::warning: return "warning";
            case icon::error: return "error";
            case icon::question: return "question";
                // Zenity wants "information" but WinForms wants "info"
                /* case icon::info: */ default:
#if _WIN32
                return "info";
#else
                return "information";
#endif
        }
    }
```

7.10.3.4 kill()

```
bool pfd::internal::dialog::kill ( ) const  [inline]
```

Definition at line 970 of file FileDialog.h.

```
    {
        return m_async->kill();
    }
```

7.10.3.5 osascript_quote()

```
std::string pfd::internal::dialog::osascript_quote (
              std::string const & str ) const  [inline], [protected]
```

Definition at line 1044 of file FileDialog.h.

```
    {
        return "\"" + std::regex_replace(str, std::regex("[\\\\\"]"), "\\$&") + "\"";
    }
```

7.10.3.6 powershell_quote()

```
std::string pfd::internal::dialog::powershell_quote (
              std::string const & str ) const  [inline], [protected]
```

Definition at line 1036 of file FileDialog.h.

```
    {
        return "'" + std::regex_replace(str, std::regex("['\"]"), "$&$&") + "'";
    }
```

7.10.3.7 ready()

```
bool pfd::internal::dialog::ready (
              int timeout = default_wait_timeout ) const  [inline]
```

Definition at line 965 of file FileDialog.h.

```
    {
        return m_async->ready(timeout);
    }
```

7.10.3.8 shell_quote()

```
std::string pfd::internal::dialog::shell_quote (
              std::string const & str ) const  [inline], [protected]
```

Definition at line 1051 of file FileDialog.h.

```
    {
        return "'" + std::regex_replace(str, std::regex("'"), "'\\''") + "'";
    }
```

7.10.4 Member Data Documentation

7.10.4.1 m_async

```
std::shared_ptr<executor> pfd::internal::dialog::m_async  [protected]
```

Definition at line 283 of file FileDialog.h.

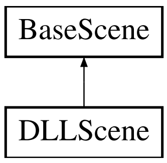
The documentation for this class was generated from the following file:

- include/core/FileDialog.h

7.11 DLLScene Class Reference

```
#include <DLLScene.hpp>
```

Inheritance diagram for DLLScene:



Public Member Functions

- DLLScene (sf::RenderWindow ∗window)
- void reset ()
- void pollEvent (sf::Event event, sf::Vector2f mousePosView) override
- void update () override
- void render () override
- std::vector< EventAnimation > addModeEvents (int chosenNode)
- std::vector< EventAnimation > deleteModeEvents (int chosenNode)
- std::vector< EventAnimation > updateModeEvents (int chosenNode)
- std::vector< EventAnimation > searchModeEvents (int chosenNode)

Public Member Functions inherited from BaseScene

- BaseScene (sf::RenderWindow ∗window)
- void createModeButton (sf::Vector2f position, std::string textString)
- virtual void pollEvent (sf::Event event, sf::Vector2f mousePosView)=0
- virtual void update ()=0
- virtual void render ()=0

Additional Inherited Members

Public Attributes inherited from BaseScene

- Button ∗ modeButton {}
- bool isMenuOpen {}
- bool isDemoCodeOpen {}

Protected Member Functions inherited from BaseScene

- void setWindow (sf::RenderWindow ∗window)

Protected Attributes inherited from BaseScene

- sf::RenderWindow ∗ window {}
- ControlMenu ∗ controlMenu

7.11.1 Detailed Description

Definition at line 12 of file DLLScene.hpp.

7.11.2 Constructor & Destructor Documentation

7.11.2.1 DLLScene()

```
DLLScene::DLLScene (
            sf::RenderWindow * window )  [explicit]
```

Definition at line 7 of file DLLScene.cpp.

```
                                          : BaseScene(window) {
    this->init();
}
```

7.11.3 Member Function Documentation

7.11.3.1 addModeEvents()

```
std::vector< EventAnimation > DLLScene::addModeEvents (
              int chosenNode )
```

Definition at line 143 of file DLLScene.cpp.

```
00143                                                                  {
00144     this->linkedList->resetEvents();
00145     if (chosenNode < 0 || chosenNode > this->linkedList->getSize())
00146         return {};
00147 
00148     std::vector<EventAnimation> events;
00149     EventAnimation event;
00150     int size = this->linkedList->getSize();
00151 
00152     if (chosenNode == 0) {
00153         this->linkedList->initHighlighter(
00154                 constants::Highlighter::DLL::CODES_PATH[0].second,
00155                 constants::Highlighter::DLL::CODES_PATH[0].first
00156         );
00157 
00158         event.titleNodes.emplace_back(chosenNode, "temp");
00159         if (size == 1)
00160             event.titleNodes.emplace_back(1, "head|tail");
00161         else if (size > 1){
00162             event.titleNodes.emplace_back(1, "head");
00163             event.titleNodes.emplace_back(size, "tail");
00164         }
00165         if (size)
00166             event.hiddenArrows.emplace_back(1, NodeInfo::ArrowType::LEFT);
00167         event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00168         event.colorNodes.emplace_back(chosenNode);
00169         event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00170         event.lines = {0, 1};
00171 
00172         events.emplace_back(event);
00173 
00174         event.reset();
00175 
00176         event.titleNodes.emplace_back(chosenNode, "temp");
00177         if (size == 1)
00178             event.titleNodes.emplace_back(1, "head|tail");
00179         else if (size > 1){
00180             event.titleNodes.emplace_back(1, "head");
00181             event.titleNodes.emplace_back(size, "tail");
00182         }
00183         if (size)
00184             event.hiddenArrows.emplace_back(1, NodeInfo::ArrowType::LEFT);
00185         event.colorArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00186         event.colorNodes.emplace_back(chosenNode);
00187         event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00188         event.isPrintNormal = true;
00189         event.lines = {2};
00190 
00191         events.emplace_back(event);
00192 
00193         if (size) {
00194             event.reset();
00195             event.titleNodes.emplace_back(chosenNode, "temp");
00196             if (size == 1)
00197                 event.titleNodes.emplace_back(1, "head|tail");
00198             else if (size > 1){
00199                 event.titleNodes.emplace_back(1, "head");
00200                 event.titleNodes.emplace_back(size, "tail");
00201             }
00202             event.colorArrows = {
00203 //                      {chosenNode, NodeInfo::ArrowType::RIGHT},
00204                     {chosenNode + 1, NodeInfo::ArrowType::LEFT}
00205             };
00206             event.colorNodes.emplace_back(chosenNode + 1);
00207             event.statusChosenNode = NodeInfo::StatusNode::InChain;
00208             event.lines = {3, 4};
00209 
00210             events.emplace_back(event);
00211         }
00212 
00213         event.reset();
00214         if (size) {
00215             event.titleNodes = {
00216                     {chosenNode, "head"},
00217                     {size,       "tail"}
00218             };
00219             event.lines = {7};
00220         }
```

```
        else {
            event.titleNodes.emplace_back(chosenNode, "head|tail");
            event.lines = {5, 6, 7};
        }
        event.colorNodes = {chosenNode};

        events.emplace_back(event);
    }
    else if (chosenNode == size) {
        this->linkedList->initHighlighter(
                constants::Highlighter::DLL::CODES_PATH[1].second,
                constants::Highlighter::DLL::CODES_PATH[1].first
        );

        event.titleNodes.emplace_back(chosenNode, "temp");
        if (size == 1)
            event.titleNodes.emplace_back(0, "head|tail");
        else if (size > 1){
            event.titleNodes.emplace_back(0, "head");
            event.titleNodes.emplace_back(size - 1, "tail");
        }
        event.hiddenArrows.emplace_back(size - 1, NodeInfo::ArrowType::RIGHT);
        event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::LEFT);
        event.colorNodes.emplace_back(chosenNode);
        event.statusChosenNode = NodeInfo::StatusNode::OutChain;
        event.lines = {0, 1};

        events.emplace_back(event);

        event.reset();

        event.titleNodes.emplace_back(chosenNode, "temp");
        if (size == 1)
            event.titleNodes.emplace_back(0, "head|tail");
        else if (size > 1){
            event.titleNodes.emplace_back(0, "head");
            event.titleNodes.emplace_back(size - 1, "tail");
        }
        event.hiddenArrows.emplace_back(size - 1, NodeInfo::ArrowType::RIGHT);
        event.colorArrows.emplace_back(chosenNode, NodeInfo::ArrowType::LEFT);
        event.colorNodes.emplace_back(chosenNode);
        event.statusChosenNode = NodeInfo::StatusNode::OutChain;
        event.lines = {2};

        events.emplace_back(event);

        event.reset();
        event.titleNodes.emplace_back(chosenNode, "temp");
        if (size == 1)
            event.titleNodes.emplace_back(0, "head|tail");
        else if (size > 1){
            event.titleNodes.emplace_back(0, "head");
            event.titleNodes.emplace_back(size - 1, "tail");
        }
        event.colorArrows = {
//                  {chosenNode, NodeInfo::ArrowType::LEFT},
                {chosenNode - 1, NodeInfo::ArrowType::RIGHT}
        };
        event.colorNodes.emplace_back(chosenNode - 1);
        event.statusChosenNode = NodeInfo::StatusNode::InChain;
        event.lines = {3};

        events.emplace_back(event);

        event.reset();
        event.titleNodes = {
                {chosenNode, "tail"},
                {0, "head"}
        };
        event.colorNodes = {chosenNode};
        event.lines = {4};

        events.emplace_back(event);
    }
    else {
        this->linkedList->initHighlighter(
                constants::Highlighter::DLL::CODES_PATH[2].second,
                constants::Highlighter::DLL::CODES_PATH[2].first
        );

        event.titleNodes = {
                {chosenNode, "temp"},
                {0,          "head"},
                {size,       "tail"}
        };
        event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
        event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::LEFT);
        event.colorNodes.emplace_back(chosenNode);
        event.statusChosenNode = NodeInfo::StatusNode::OutChain;
        event.lines = {0, 1};

        events.emplace_back(event);

        event.reset();
        event.titleNodes = {
                {chosenNode, "temp"},
                {0,          "head|current"},
                {size,       "tail"}
        };
        event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
        event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::LEFT);
        event.colorNodes.emplace_back(0);
        event.statusChosenNode = NodeInfo::StatusNode::OutChain;
        event.lines = {2};

        events.emplace_back(event);

        for (int i = 0; i < chosenNode; ++i) {
            event.reset();
            event.titleNodes = {
                    {chosenNode, "temp"},
                    {0,          "head"},
                    {size,       "tail"},
                    {i, "current"}
            };
            event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
            event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::LEFT);
            event.colorNodes.emplace_back(i);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.lines = {3};

            events.emplace_back(event);

            if (i == chosenNode - 1)
                break;

            event.reset();
            event.titleNodes = {
                    {chosenNode, "temp"},
                    {0,          "head"},
                    {size,       "tail"},
                    {i, "current"}
            };
            event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
            event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::LEFT);
            event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
//              event.colorArrows.emplace_back(i + 1 + (i + 1 == chosenNode),
NodeInfo::ArrowType::LEFT);
            event.colorNodes.emplace_back(i);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.lines = {4};

            events.emplace_back(event);
        }

        event.reset();
        event.titleNodes = {
                {chosenNode, "temp"},
                {0,          "head"},
                {size,       "tail"},
                {chosenNode - 1, "current"}
        };
        event.colorArrows = {
                {chosenNode, NodeInfo::ArrowType::RIGHT},
                {chosenNode, NodeInfo::ArrowType::LEFT}
        };
        event.colorNodes.emplace_back(chosenNode);
        event.statusChosenNode = NodeInfo::StatusNode::OutChain;
        event.isPrintNormal = true;
        event.lines = {5, 6};

        events.emplace_back(event);

        event.reset();
        event.titleNodes = {
                {chosenNode, "temp"},
                {0,          "head"},
                {size,       "tail"}
        };
        event.colorNodes.emplace_back(chosenNode);
        event.statusChosenNode = NodeInfo::StatusNode::InChain;
        event.lines = {7, 8};

        events.emplace_back(event);
```

```
    }

    return events;
}
```

7.11.3.2 deleteModeEvents()

```
std::vector< EventAnimation > DLLScene::deleteModeEvents (
            int chosenNode )
```

Definition at line 399 of file DLLScene.cpp.

```
                                                                    {
    this->linkedList->resetEvents();
    if (chosenNode < 0 || chosenNode >= this->linkedList->getSize())
        return {};

    std::vector<EventAnimation> events;
    EventAnimation event;
    int size = this->linkedList->getSize();

    if (chosenNode == 0) {
        this->linkedList->initHighlighter(
                constants::Highlighter::DLL::CODES_PATH[3].second,
                constants::Highlighter::DLL::CODES_PATH[3].first
        );

        if (size == 1) {
            event.titleNodes.emplace_back(chosenNode, "head|tail|temp");
            event.colorNodes.emplace_back(chosenNode);
            event.statusChosenNode = NodeInfo::StatusNode::InChain;
            event.lines = {0, 1, 2};

            events.emplace_back(event);

            event.reset();
            event.statusChosenNode = NodeInfo::StatusNode::Visible;
            event.lines = {5, 6, 7};

            events.emplace_back(event);
        }
        else {
            event.titleNodes = {
                    {chosenNode, "head|temp"},
                    {size - 1, "tail"}
            };
            event.colorNodes.emplace_back(chosenNode);
            event.lines = {0, 1};

            events.emplace_back(event);

            event.reset();
            if (size == 2)
                event.titleNodes.emplace_back(size - 1, "head|tail");
            else
                event.titleNodes = {
                    {size - 1, "tail"},
                    {chosenNode + 1, "head" }
            };
            event.titleNodes.emplace_back(chosenNode, "temp");
            event.colorNodes.emplace_back(chosenNode + 1);
//            event.isPrintNormal = true;
//            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.lines = {2};

            events.emplace_back(event);

            event.reset();
            if (size == 2)
                event.titleNodes.emplace_back(size - 1, "head|tail");
            else
                event.titleNodes = {
                        {size - 1, "tail"},
                        {chosenNode + 1, "head" }
                };
            event.titleNodes.emplace_back(chosenNode, "temp");
            event.colorNodes.emplace_back(chosenNode);
            event.hiddenArrows = {
//                    {chosenNode, NodeInfo::ArrowType::RIGHT},
                    {chosenNode + 1, NodeInfo::ArrowType::LEFT}
            };
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.isPrintNormal = true;
            event.lines = {3, 4};

            events.emplace_back(event);

            event.reset();
            if (size == 2)
                event.titleNodes.emplace_back(size - 1, "head|tail");
            else
                event.titleNodes = {
                        {size - 1, "tail"},
                        {chosenNode + 1, "head" }
                };
            event.hiddenArrows = {
                    {chosenNode + 1, NodeInfo::ArrowType::LEFT}
            };
            event.statusChosenNode = NodeInfo::StatusNode::Visible;
            event.lines = {7};

            events.emplace_back(event);
        }
    }
    else if (chosenNode == size - 1) {
        this->linkedList->initHighlighter(
                constants::Highlighter::DLL::CODES_PATH[4].second,
                constants::Highlighter::DLL::CODES_PATH[4].first
                );

        event.titleNodes = {
                {0, "head"},
                {chosenNode, "tail|temp"}
        };
        event.colorNodes.emplace_back(chosenNode);
        event.lines = {0, 1};

        events.emplace_back(event);

        event.reset();
        if (size == 2)
            event.titleNodes.emplace_back(0, "head|tail");
        else
            event.titleNodes = {
                    {chosenNode - 1, "tail"},
                    {0, "head" }
            };
        event.titleNodes.emplace_back(chosenNode, "temp");
        event.colorNodes.emplace_back(chosenNode - 1);
        event.lines = {2};

        events.emplace_back(event);

        event.reset();
        if (size == 2)
            event.titleNodes.emplace_back(0, "head|tail");
        else
            event.titleNodes = {
                    {chosenNode - 1, "tail"},
                    {0, "head" }
            };
        event.titleNodes.emplace_back(chosenNode, "temp");
        event.colorNodes.emplace_back(chosenNode);
        event.hiddenArrows = {
                {chosenNode - 1, NodeInfo::ArrowType::RIGHT}
        };
        event.statusChosenNode = NodeInfo::StatusNode::OutChain;
        event.isPrintNormal = true;
        event.lines = {3};

        events.emplace_back(event);

        event.reset();
        if (size == 2)
            event.titleNodes.emplace_back(0, "head|tail");
        else
            event.titleNodes = {
                    {chosenNode - 1, "tail"},
                    {0, "head" }
            };
        event.hiddenArrows = {
                {chosenNode - 1, NodeInfo::ArrowType::RIGHT}
        };
        event.statusChosenNode = NodeInfo::StatusNode::Visible;
        event.lines = {4};

```

```
            events.emplace_back(event);
        }
        else {
            this->linkedList->initHighlighter(
                    constants::Highlighter::DLL::CODES_PATH[5].second,
                    constants::Highlighter::DLL::CODES_PATH[5].first
            );

            event.titleNodes = {
                    {0, "head|temp"},
                    {size - 1, "tail"}
            };
            event.colorNodes.emplace_back(0);
            event.lines = {0, 1};

            events.emplace_back(event);

            for (int i = 0; i <= chosenNode; ++i) {
                event.reset();
                event.titleNodes = {
                        {0, "head"},
                        {i, "temp"},
                        {size - 1, "tail"}
                };
                event.colorNodes.emplace_back(i);
                event.lines = {2};

                events.emplace_back(event);

                if (i == chosenNode)
                    break;

                event.reset();
                event.titleNodes = {
                        {0, "head"},
                        {i, "temp"},
                        {size - 1, "tail"}
                };
                event.colorNodes.emplace_back(i);
                event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
                event.lines = {3};

                events.emplace_back(event);
            }

            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {chosenNode, "temp"},
                    {size - 1, "tail"}
            };
            event.colorNodes.emplace_back(chosenNode);
            event.colorArrows = {
                    {chosenNode - 1, NodeInfo::ArrowType::RIGHT},
                    {chosenNode + 1, NodeInfo::ArrowType::LEFT}
            };
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.isPrintNormal = true;
            event.lines = {4, 5};

            events.emplace_back(event);

            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {size - 1, "tail"}
            };
            event.statusChosenNode = NodeInfo::StatusNode::Visible;
            event.lines = {6};

            events.emplace_back(event);
        }

        return events;
    }
```

7.11.3.3 pollEvent()

```
void DLLScene::pollEvent (
            sf::Event event,
            sf::Vector2f mousePosView )  [override], [virtual]
```

Implements BaseScene.

Definition at line 11 of file DLLScene.cpp.

```
                                                                 {
    if (this->isMenuOpen)
        this->menu->pollEvents(event, mousePosView);

    this->controlMenu->pollEvents(event, mousePosView);
}
```

7.11.3.4 render()

`void DLLScene::render ( ) [override], [virtual]`

Implements BaseScene.

Definition at line 123 of file DLLScene.cpp.

```
                            {
    if (this->isMenuOpen)
        this->menu->render();

    if (this->isDemoCodeOpen)
        this->linkedList->renderHighlighter();

    this->controlMenu->render();
    this->linkedList->render();
}
```

7.11.3.5 reset()

`void DLLScene::reset ( )`

Definition at line 139 of file DLLScene.cpp.

```
                           {
    this->menu->resetActiveOptionMenu();
}
```

7.11.3.6 searchModeEvents()

```
std::vector< EventAnimation > DLLScene::searchModeEvents (
               int chosenNode )
```

Definition at line 717 of file DLLScene.cpp.

```
                                                                     {
    this->linkedList->resetEvents();
    this->linkedList->initHighlighter(
            constants::Highlighter::DLL::CODES_PATH[7].second,
            constants::Highlighter::DLL::CODES_PATH[7].first
    );

    std::vector<EventAnimation> events;
    EventAnimation event;
    int size = this->linkedList->getSize();

    if (size > 1)
        event.titleNodes = {
                {0, "head|current"},
                {size - 1, "tail"}
        };
    else
        event.titleNodes = {
                {0, "head|tail|current"}
        };
    event.colorNodes.push_back(0);
    event.lines = {0};

    events.emplace_back(event);

    for (int i = 0; i <= chosenNode; ++i) {
        if (i == chosenNode && chosenNode == this->linkedList->getSize())
            break;

        event.reset();
        event.titleNodes = {
                {0, "head"},
                {size - 1, "tail"},
                {i, "current"}
        };
        event.colorNodes.push_back(i);
        event.lines = {1};

        events.emplace_back(event);

        if (i == chosenNode) break;

        event.reset();
        event.titleNodes = {
                {0, "head"},
                {size - 1, "tail"},
                {i, "current"}
        };
        event.colorNodes.push_back(i);
        event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
        event.lines = {4};

        events.emplace_back(event);
    }

    if (chosenNode == this->linkedList->getSize()) {
        event.reset();
        event.titleNodes.emplace_back(0, "head");
        event.titleNodes.emplace_back(size - 1, "tail");
        event.lines = {5};

        events.emplace_back(event);
    } else {
        event.reset();
        event.titleNodes = {
                {0, "head"},
                {size - 1, "tail"},
                {chosenNode, "current"}
        };
        event.colorNodes.push_back(chosenNode);
        event.lines = {2, 3};

        events.emplace_back(event);
    }

    return events;
}
```

7.11.3.7 update()

```
void DLLScene::update ( )  [override], [virtual]
```

Implements BaseScene.

Definition at line 18 of file DLLScene.cpp.

```
                        {
    if (this->isMenuOpen) {
        this->menu->update();

        constants::MenuLinkedList::Button status = this->menu->getActiveOptionMenu();
        constants::MenuLinkedList::CreateMode::Button createMode;
        switch (status){
            case constants::MenuLinkedList::Button::CREATE_BUTTON:
                createMode = this->menu->getActiveCreateMode();
                if (createMode == constants::MenuLinkedList::CreateMode::Button::RANDOM_BUTTON) {
                    if (this->menu->createModeValue[0] == "None")
                        break;
                    if (this->menu->createModeValue[0].empty())
                        this->menu->createModeValue[0] = "0";
                    int size = std::stoi(this->menu->createModeValue[0]);
                    this->linkedList->createLinkedList(size);
                } else if (createMode == constants::MenuLinkedList::CreateMode::Button::DEFINED_LIST_BUTTON) {
                    if (this->menu->createModeValue[1] == "None")
                        break;
                    std::vector<std::string> values;
                    std::string value = this->menu->createModeValue[1];
                    std::stringstream ss(value);
                    std::string token;
                    while (std::getline(ss, token, ',')) {
                        values.push_back(token);
                    }
                    this->linkedList->createLinkedList(values);
                } else if (createMode == constants::MenuLinkedList::CreateMode::Button::FILE_BUTTON) {
                    if (this->menu->createModeValue[2] == "None")
                        break;
                    std::vector<std::string> values;
                    std::string value = this->menu->createModeValue[2];
                    std::stringstream ss(value);
                    std::string token;
                    while (std::getline(ss, token, ','))
                        values.push_back(token);
                    this->linkedList->createLinkedList(values);
                    this->menu->createModeValue[2] = "None";
                }
                this->controlMenu->reset();
                break;
            case constants::MenuLinkedList::Button::ADD_BUTTON:
                if (this->menu->addModeValue[0] == "None" || this->menu->addModeValue[1] == "None" || this->menu->addModeValue[0].empty())
                    break;

                this->linkedList->addNode(
                        std::stoi(this->menu->addModeValue[0]),
                        this->menu->addModeValue[1],
                        this->addModeEvents(std::stoi(this->menu->addModeValue[0]))
                );

                std::cout « "Add: " « this->menu->addModeValue[0] « " " « this->menu->addModeValue[1] « std::endl;
                this->menu->addModeValue[0] = this->menu->addModeValue[1] = "None";
                this->controlMenu->reset();
                break;
            case constants::MenuLinkedList::Button::DELETE_BUTTON:
                if (this->menu->deleteModeValue == "None" || this->menu->deleteModeValue.empty())
                    break;

                this->linkedList->deleteNode(
                        std::stoi(this->menu->deleteModeValue),
                        this->deleteModeEvents(std::stoi(this->menu->deleteModeValue))
                );

                std::cout « "Delete: " « this->menu->deleteModeValue « std::endl;
                this->menu->deleteModeValue = "None";
                this->controlMenu->reset();
                break;
            case constants::MenuLinkedList::Button::UPDATE_BUTTON:
                if (this->menu->updateModeValue[0] == "None" || this->menu->updateModeValue[1] == "None" || this->menu->updateModeValue[0].empty())
                    break;

                this->linkedList->updateNode(
                        std::stoi(this->menu->updateModeValue[0]),
                        this->menu->updateModeValue[1],
                        this->updateModeEvents(std::stoi(this->menu->updateModeValue[0]))
                );

                std::cout « "Update: " « this->menu->updateModeValue[0] « " " « this->menu->updateModeValue[1] « std::endl;
                this->menu->updateModeValue[0] = this->menu->updateModeValue[1] = "None";
                this->controlMenu->reset();
                break;
            case constants::MenuLinkedList::Button::SEARCH_BUTTON:
                if (this->menu->searchModeValue == "None" || this->menu->searchModeValue.empty())
                    break;

                this->linkedList->searchNode(
                        this->searchModeEvents(this->linkedList->findValue(this->menu->searchModeValue))
                );

                std::cout « "Search: " « this->menu->searchModeValue « std::endl;
                this->menu->searchModeValue = "None";
                this->controlMenu->reset();
                break;
        }
    }

    this->controlMenu->update();

    this->linkedList->processControlMenu(this->controlMenu->getStatus());
    this->linkedList->setSpeed(this->controlMenu->getSpeed());

    this->linkedList->update();
}
```

7.11.3.8 updateModeEvents()

```
std::vector< EventAnimation > DLLScene::updateModeEvents (
              int chosenNode )
```

Definition at line 629 of file DLLScene.cpp.

```
                                                                      {
    this->linkedList->resetEvents();
    if (chosenNode < 0 || chosenNode >= this->linkedList->getSize())
        return {};

    this->linkedList->initHighlighter(
            constants::Highlighter::DLL::CODES_PATH[6].second,
            constants::Highlighter::DLL::CODES_PATH[6].first
    );

    std::vector<EventAnimation> events;
    EventAnimation event;
    int size = this->linkedList->getSize();

    if (size > 1)
        event.titleNodes = {
                {0, "head|current"},
                {size - 1, "tail"}
        };
    else
        event.titleNodes = {
                {0, "head|tail|current"}
        };
    event.colorNodes.push_back(0);
    event.isPrintPreVal = true;
    event.lines = {0};

    events.emplace_back(event);

    if (chosenNode) {
        for (int i = 0; i <= chosenNode; ++i) {
            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {size - 1, "tail"},
                    {i, "current"},
            };
            event.colorNodes.push_back(i);
            event.isPrintPreVal = true;
            event.lines = {1};

            events.emplace_back(event);

            if (i == chosenNode) break;

            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {i, "current"},
                    {size - 1, "tail"}
            };
            event.colorNodes.push_back(i);
            event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
            event.isPrintPreVal = true;
            event.lines = {2};

            events.emplace_back(event);
        }
    }

    event.reset();
    if (size == 1)
        event.titleNodes = {
                {0, "head|tail|current"}
        };
    else if (chosenNode == size - 1)
        event.titleNodes = {
                {0, "head"},
                {chosenNode, "current|tail"}
        };
    else if (chosenNode == 0)
        event.titleNodes = {
                {0, "head|current"},
                {size - 1, "tail"}
        };
    else
        event.titleNodes = {
                {0,           "head"},
                {chosenNode, "current"},
                {size - 1,    "tail"}
        };
    event.lines = {3};

    events.emplace_back(event);

    return events;
}
```

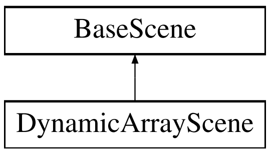
The documentation for this class was generated from the following files:

- include/libScene/DLLScene.hpp
- include/libScene/DLLScene.cpp

7.12 DynamicArrayScene Class Reference

```
#include <DynamicArrayScene.hpp>
```

Inheritance diagram for DynamicArrayScene:



Public Member Functions

- DynamicArrayScene (sf::RenderWindow ∗window)
- void reset ()
- void pollEvent (sf::Event event, sf::Vector2f mousePosView) override
- void update () override
- void render () override
- std::vector< EventAnimation > addModeEvents (int chosenNode)
- std::vector< EventAnimation > deleteModeEvents (int chosenNode)
- std::vector< EventAnimation > updateModeEvents (int chosenNode)
- std::vector< EventAnimation > searchModeEvents (int chosenNode)
- std::vector< EventAnimation > allocateModeEvents (int newSize)

Public Member Functions inherited from BaseScene

- BaseScene (sf::RenderWindow ∗window)
- void createModeButton (sf::Vector2f position, std::string textString)
- virtual void pollEvent (sf::Event event, sf::Vector2f mousePosView)=0
- virtual void update ()=0
- virtual void render ()=0

Additional Inherited Members

Public Attributes inherited from BaseScene

- Button ∗ modeButton {}
- bool isMenuOpen {}
- bool isDemoCodeOpen {}

Protected Member Functions inherited from BaseScene

- void setWindow (sf::RenderWindow ∗window)

Protected Attributes inherited from BaseScene

- sf::RenderWindow ∗ window {}
- ControlMenu ∗ controlMenu

7.12.1 Detailed Description

Definition at line 12 of file DynamicArrayScene.hpp.

7.12.2 Constructor & Destructor Documentation

7.12.2.1 DynamicArrayScene()

```
DynamicArrayScene::DynamicArrayScene (
            sf::RenderWindow * window )   [explicit]
```

Definition at line 7 of file DynamicArrayScene.cpp.

```
                                                    : BaseScene(window) {
    this->init();
}
```

7.12.3 Member Function Documentation

7.12.3.1 addModeEvents()

```
std::vector< EventAnimation > DynamicArrayScene::addModeEvents (
              int chosenNode )
```

Definition at line 156 of file DynamicArrayScene.cpp.

```
                                                                              {
    this->array->resetEvents();
    if (chosenNode < 0 || chosenNode > this->array->getSize())
        return {};

    // init highlighter
    // ...

    int size = this->array->getSize() + 1,
        squaresSize = this->array->getSquaresSize();
    std::vector<EventAnimation> events;
    EventAnimation event;

    if (size > squaresSize) {
        ++squaresSize;
        event.eventSquares.assign(squaresSize, EventSquare());
        event.eventSquaresTemp.assign(squaresSize, EventSquare());
        for (auto &square : event.eventSquares) {
            square.status = Square::Status::active;
            square.isPrintPreVal = true;
        }
        event.eventSquares.back().status = Square::Status::hidden;
        if (size > 1)
            event.eventSquares[size - 2].title = "n";
        for (auto &square : event.eventSquaresTemp) {
            square.status = Square::Status::inactive;
            square.isPrintPreVal = true;
        }

        events.emplace_back(event);

        for (int i = 0; i < size - 1; ++i) {
            event = EventAnimation();
            event.eventSquares.assign(squaresSize, EventSquare());
            event.eventSquaresTemp.assign(squaresSize, EventSquare());
            for (auto &square : event.eventSquares) {
                square.status = Square::Status::active;
                square.isPrintPreVal = true;
            }
            event.eventSquares.back().status = Square::Status::hidden;
            if (size > 1)
                event.eventSquares[size - 2].title = "n";
            for (auto &square : event.eventSquaresTemp) {
                square.status = Square::Status::inactive;
                square.isPrintPreVal = true;
            }
            for (int j = 0; j < i; ++j) {
                event.eventSquaresTemp[j].status = Square::Status::active;
                event.eventSquaresTemp[j].isPrintPreVal = false;
            }
            event.eventSquaresTemp[i].status = Square::Status::chosen;
            event.eventSquaresTemp[i].title = "m";

            events.emplace_back(event);

            event.eventSquaresTemp[i].isPrintPreVal = false;
            event.eventSquares[i].status = Square::Status::chosen;

            events.emplace_back(event);
        }
    }

    event = EventAnimation();
    event.eventSquares.assign(squaresSize, EventSquare());
    event.eventSquaresTemp.assign(squaresSize, EventSquare());
    for (auto &square : event.eventSquares) {
        square.status = Square::Status::active;
        square.isPrintPreVal = true;
    }
    for (int i = size - 1; i < squaresSize; ++i)
        event.eventSquares[i].status = Square::Status::inactive;
    if (size > 1)
        event.eventSquares[size - 2].title = "n";
    for (auto &square : event.eventSquaresTemp) {
        square.status = Square::Status::hidden;
    }

    events.emplace_back(event);
```

```
00234
00235     event = EventAnimation();
00236     event.eventSquares.assign(squaresSize, EventSquare());
00237     event.eventSquaresTemp.assign(squaresSize, EventSquare());
00238     for (auto &square : event.eventSquares) {
00239         square.status = Square::Status::active;
00240         square.isPrintPreVal = true;
00241     }
00242     for (int i = size; i < squaresSize; ++i)
00243         event.eventSquares[i].status = Square::Status::inactive;
00244     event.eventSquares[size - 1].title = "n";
00245     for (auto &square : event.eventSquaresTemp)
00246         square.status = Square::Status::hidden;
00247
00248     events.emplace_back(event);
00249
00250     for (int i = size - 1; i >= chosenNode; --i) {
00251         event = EventAnimation();
00252         event.eventSquares.assign(squaresSize, EventSquare());
00253         event.eventSquaresTemp.assign(squaresSize, EventSquare());
00254         for (auto &square: event.eventSquares) {
00255             square.status = Square::Status::active;
00256             square.isPrintPreVal = true;
00257         }
00258         for (int j = size; j < squaresSize; ++j)
00259             event.eventSquares[j].status = Square::Status::inactive;
00260         event.eventSquares[size - 1].title = "n";
00261         for (int j = size - 1; j > i; --j)
00262             event.eventSquares[j].isPrintPreVal = false;
00263         event.eventSquares[i].status = Square::Status::chosen;
00264         for (auto &square : event.eventSquaresTemp)
00265             square.status = Square::Status::hidden;
00266
00267         events.emplace_back(event);
00268
00269         event.eventSquares[i].isPrintPreVal = false;
00270         if (i > chosenNode)
00271             event.eventSquares[i - 1].status = Square::Status::chosen;
00272
00273         events.emplace_back(event);
00274     }
00275
00276     return events;
00277 }
```

7.12.3.2 allocateModeEvents()

```
std::vector< EventAnimation > DynamicArrayScene::allocateModeEvents (
               int newSize )
```

Definition at line 402 of file DynamicArrayScene.cpp.

```
00402                                                                       {
00403     this->array->resetEvents();
00404
00405     // init highlighter
00406     // ...
00407
00408     int size = this->array->getSize(),
00409         oldSize = this->array->getSquaresSize(),
00410         squaresSize = std::max(oldSize, newSize);
00411
00412     std::vector<EventAnimation> events;
00413     EventAnimation event;
00414
00415     event.eventSquares.assign(squaresSize, EventSquare());
00416     event.eventSquaresTemp.assign(newSize, EventSquare());
00417     for (int i = 0; i < size; ++i) {
00418         event.eventSquares[i].status = Square::Status::active;
00419         if (i == size - 1)
00420             event.eventSquares[i].title = "n";
00421     }
00422     for (int i = size; i < oldSize; ++i) {
00423         event.eventSquares[i].status = Square::Status::inactive;
00424     }
00425     for (int i = oldSize; i < newSize; ++i) {
00426         event.eventSquares[i].status = Square::Status::hidden;
00427     }
00428     for (auto &square : event.eventSquaresTemp) {
```

```
00429         square.status = Square::Status::inactive;
00430         square.isPrintPreVal = true;
00431     }
00432 
00433     events.emplace_back(event);
00434 
00435     for (int i = 0; i < std::min(size, newSize); ++i) {
00436         event = EventAnimation();
00437         event.eventSquares.assign(squaresSize, EventSquare());
00438         event.eventSquaresTemp.assign(newSize, EventSquare());
00439         for (int j = 0; j < size; ++j) {
00440             event.eventSquares[j].status = Square::Status::active;
00441             if (j == size - 1)
00442                 event.eventSquares[j].title = "n";
00443         }
00444         for (int j = size; j < oldSize; ++j) {
00445             event.eventSquares[j].status = Square::Status::inactive;
00446         }
00447         for (int j = oldSize; j < newSize; ++j) {
00448             event.eventSquares[j].status = Square::Status::hidden;
00449         }
00450         for (auto &square : event.eventSquaresTemp) {
00451             square.status = Square::Status::inactive;
00452             square.isPrintPreVal = true;
00453         }
00454         for (int j = 0; j < i; ++j) {
00455             event.eventSquaresTemp[j].status = Square::Status::active;
00456             event.eventSquaresTemp[j].isPrintPreVal = false;
00457         }
00458         event.eventSquaresTemp[i].title = "m";
00459         event.eventSquaresTemp[i].status = Square::Status::chosen;
00460 
00461         events.emplace_back(event);
00462 
00463         event.eventSquaresTemp[i].isPrintPreVal = false;
00464         event.eventSquares[i].status = Square::Status::chosen;
00465 
00466         events.emplace_back(event);
00467     }
00468 
00469     event = EventAnimation();
00470     event.eventSquares.assign(squaresSize, EventSquare());
00471     event.eventSquaresTemp.assign(newSize, EventSquare());
00472 
00473     for (int i = 0; i < std::min(size, newSize); ++i) {
00474         event.eventSquares[i].status = Square::Status::active;
00475         if (i == std::min(size, newSize) - 1)
00476             event.eventSquares[i].title = "n";
00477     }
00478     for (int i = size; i < newSize; ++i) {
00479         event.eventSquares[i].status = Square::Status::inactive;
00480     }
00481     for (int i = newSize; i < oldSize; ++i) {
00482         event.eventSquares[i].status = Square::Status::hidden;
00483     }
00484     for (auto &square : event.eventSquaresTemp) {
00485         square.status = Square::Status::hidden;
00486     }
00487 
00488     events.emplace_back(event);
00489 
00490     return events;
00491 }
```

7.12.3.3 deleteModeEvents()

```
std::vector< EventAnimation > DynamicArrayScene::deleteModeEvents (
               int chosenNode )
```

Definition at line 279 of file DynamicArrayScene.cpp.

```
00279                                                                            {
00280     this->array->resetEvents();
00281     if (chosenNode < 0 || chosenNode >= this->array->getSize())
00282         return {};
00283 
00284     // init highlighter
00285     // ...
00286 
```

```
    int size = this->array->getSize(),
        squaresSize = this->array->getSquaresSize();
    std::vector<EventAnimation> events;
    EventAnimation event;

    for (int i = chosenNode; i < size - 1; ++i) {
        event = EventAnimation();
        event.eventSquares.assign(squaresSize, EventSquare());
        for (auto &square : event.eventSquares) {
            square.status = Square::Status::active;
            square.isPrintPreVal = true;
        }
        for (int j = size; j < squaresSize; ++j)
            event.eventSquares[j].status = Square::Status::inactive;
        for (int j = 0; j < i; ++j)
            event.eventSquares[j].isPrintPreVal = false;
        event.eventSquares[i].status = Square::Status::chosen;
        for (auto &square : event.eventSquaresTemp)
            square.status = Square::Status::hidden;
        event.eventSquares[size - 1].title = "n";

        events.emplace_back(event);

        event.eventSquares[i].isPrintPreVal = false;
        event.eventSquares[i + 1].status = Square::Status::chosen;

        events.emplace_back(event);
    }

    event = EventAnimation();
    event.eventSquares.assign(squaresSize, EventSquare());
    for (int i = 0; i < size - 1; ++i) {
        event.eventSquares[i].status = Square::Status::active;
        if (i == size - 2)
            event.eventSquares[i].title = "n";
    }
    for (int i = size - 1; i < squaresSize; ++i)
        event.eventSquares[i].status = Square::Status::inactive;

    events.emplace_back(event);

    return events;
}
```

7.12.3.4 pollEvent()

```
void DynamicArrayScene::pollEvent (
            sf::Event event,
            sf::Vector2f mousePosView )  [override], [virtual]
```

Implements BaseScene.

Definition at line 140 of file DynamicArrayScene.cpp.

```
                                                                    {
    if (this->isMenuOpen)
        this->menu->pollEvents(event, mousePosView);

    this->controlMenu->pollEvents(event, mousePosView);
}
```

7.12.3.5 render()

```
void DynamicArrayScene::render ( )  [override], [virtual]
```

Implements BaseScene.

Definition at line 129 of file DynamicArrayScene.cpp.

```
00129                               {
00130     if (this->isMenuOpen)
00131         this->menu->render();
00132 
00133     if (this->isDemoCodeOpen)
00134         this->array->renderHighlighter();
00135 
00136     this->controlMenu->render();
00137     this->array->render();
00138 }
```

7.12.3.6 reset()

```
void DynamicArrayScene::reset ( )
```

Definition at line 152 of file DynamicArrayScene.cpp.

```
00152                              {
00153     this->menu->resetActiveOptionMenu();
00154 }
```

7.12.3.7 searchModeEvents()

```
std::vector< EventAnimation > DynamicArrayScene::searchModeEvents (
               int chosenNode )
```

Definition at line 361 of file DynamicArrayScene.cpp.

```
00361                                                                                  {
00362     this->array->resetEvents();
00363 
00364     // init highlighter
00365     // ...
00366 
00367     int size = this->array->getSize(),
00368         squaresSize = this->array->getSquaresSize();
00369     std::vector<EventAnimation> events;
00370     EventAnimation event;
00371 
00372     for (int i = 0; i <= chosenNode; ++i) {
00373         if (i == size) break;
00374 
00375         event = EventAnimation();
00376         event.eventSquares.assign(squaresSize, EventSquare());
00377         for (int j = 0; j < size; ++j) {
00378             event.eventSquares[j].status = Square::Status::active;
00379             if (j == size - 1)
00380                 event.eventSquares[size - 1].title = "n";
00381         }
00382         event.eventSquares[i].status = Square::Status::chosen;
00383 
00384         events.emplace_back(event);
00385     }
00386 
00387     if (chosenNode == size) {
00388         event = EventAnimation();
00389         event.eventSquares.assign(squaresSize, EventSquare());
00390         for (int j = 0; j < size; ++j) {
00391             event.eventSquares[j].status = Square::Status::active;
00392             if (j == size - 1)
00393                 event.eventSquares[size - 1].title = "n";
00394         }
00395 
00396         events.emplace_back(event);
00397     }
00398 
00399     return events;
00400 }
```

7.12.3.8 update()

```
void DynamicArrayScene::update ( ) [override], [virtual]
```

Implements BaseScene.

Definition at line 11 of file DynamicArrayScene.cpp.

```
                                {
    if (this->isMenuOpen) {
        this->menu->update();

        constants::MenuArray::Button status = this->menu->getActiveOptionMenu();
        constants::MenuArray::CreateMode::Button createMode;
        switch (status){
            case constants::MenuArray::Button::CREATE_BUTTON:
                createMode = this->menu->getActiveCreateMode();
                if (createMode == constants::MenuArray::CreateMode::Button::RANDOM_BUTTON) {
                    if (this->menu->createModeValue[0] == "None")
                        break;
                    if (this->menu->createModeValue[0].empty())
                        this->menu->createModeValue[0] = "0";
                    int size = std::stoi(this->menu->createModeValue[0]);
                    this->array->createArray(size);
                } else if (createMode == constants::MenuArray::CreateMode::Button::DEFINED_LIST_BUTTON) {
                    if (this->menu->createModeValue[1] == "None")
                        break;
                    std::vector<std::string> values;
                    std::string value = this->menu->createModeValue[1];
                    std::stringstream ss(value);
                    std::string token;
                    while (std::getline(ss, token, ',')) {
                        values.push_back(token);
                    }
                    this->array->createArray(values);
                } else if (createMode == constants::MenuArray::CreateMode::Button::FILE_BUTTON) {
                    if (this->menu->createModeValue[2] == "None")
                        break;
                    std::vector<std::string> values;
                    std::string value = this->menu->createModeValue[2];
                    std::stringstream ss(value);
                    std::string token;
                    while (std::getline(ss, token, ','))
                        values.push_back(token);
                    this->array->createArray(values);
                    this->menu->createModeValue[2] = "None";
                }
                this->controlMenu->reset();
                break;
            case constants::MenuArray::Button::ADD_BUTTON:
                if (this->menu->addModeValue[0] == "None" || this->menu->addModeValue[1] == "None" || this->menu->addModeValue[0].empty())
                    break;

                this->array->addSquare(
                        std::stoi(this->menu->addModeValue[0]),
                        this->menu->addModeValue[1],
                        this->addModeEvents(std::stoi(this->menu->addModeValue[0]))
                );

                std::cout « "Add: " « this->menu->addModeValue[0] « " " « this->menu->addModeValue[1] « std::endl;
                this->menu->addModeValue[0] = this->menu->addModeValue[1] = "None";
                this->controlMenu->reset();
                break;
            case constants::MenuArray::Button::DELETE_BUTTON:
                if (this->menu->deleteModeValue == "None" || this->menu->deleteModeValue.empty())
                    break;

                this->array->deleteSquare(
                        std::stoi(this->menu->deleteModeValue),
                        this->deleteModeEvents(std::stoi(this->menu->deleteModeValue))
                );

                std::cout « "Delete: " « this->menu->deleteModeValue « std::endl;
                this->menu->deleteModeValue = "None";
                this->controlMenu->reset();
                break;
            case constants::MenuArray::Button::UPDATE_BUTTON:
                if (this->menu->updateModeValue[0] == "None" || this->menu->updateModeValue[1] == "None" || this->menu->updateModeValue[0].empty())
                    break;

```

```
00083                 this->array->updateSquare(
00084                         std::stoi(this->menu->updateModeValue[0]),
00085                         this->menu->updateModeValue[1],
00086                         this->updateModeEvents(std::stoi(this->menu->updateModeValue[0]))
00087                 );
00088 
00089                 std::cout « "Update: " « this->menu->updateModeValue[0] « " " «
      this->menu->updateModeValue[1] « std::endl;
00090                 this->menu->updateModeValue[0] = this->menu->updateModeValue[1] = "None";
00091                 this->controlMenu->reset();
00092                 break;
00093             case constants::MenuArray::Button::SEARCH_BUTTON:
00094                 if (this->menu->searchModeValue == "None" || this->menu->searchModeValue.empty())
00095                     break;
00096 
00097                 this->array->searchSquare(
00098                         this->searchModeEvents(this->array->findValue(this->menu->searchModeValue))
00099                 );
00100 
00101                 std::cout « "Search: " « this->menu->searchModeValue « std::endl;
00102                 this->menu->searchModeValue = "None";
00103                 this->controlMenu->reset();
00104                 break;
00105             case constants::MenuArray::Button::ALLOCATE_BUTTON:
00106                 if (this->menu->allocateModeValue == "None" || this->menu->allocateModeValue.empty())
00107                     break;
00108 
00109                 this->array->allocateSquare(
00110                         std::stoi(this->menu->allocateModeValue),
00111                         this->allocateModeEvents(std::stoi(this->menu->allocateModeValue))
00112                 );
00113 
00114                 std::cout « "Allocate: " « this->menu->allocateModeValue « std::endl;
00115                 this->menu->allocateModeValue = "None";
00116                 this->controlMenu->reset();
00117                 break;
00118         }
00119     }
00120 
00121     this->controlMenu->update();
00122 
00123     this->array->processControlMenu(this->controlMenu->getStatus());
00124     this->array->setSpeed(this->controlMenu->getSpeed());
00125 
00126     this->array->update();
00127 }
```

7.12.3.9 updateModeEvents()

```
std::vector< EventAnimation > DynamicArrayScene::updateModeEvents (
              int chosenNode )
```

Definition at line 331 of file DynamicArrayScene.cpp.

```
00331                                                                                 {
00332     this->array->resetEvents();
00333     if (chosenNode < 0 || chosenNode >= this->array->getSize())
00334         return {};
00335 
00336     // init highlighter
00337     // ...
00338 
00339     std::vector<EventAnimation> events;
00340     EventAnimation event;
00341 
00342     event = EventAnimation();
00343     event.eventSquares.assign(this->array->getSquaresSize(), EventSquare());
00344     for (int i = 0; i < this->array->getSize(); ++i) {
00345         event.eventSquares[i].status = Square::Status::active;
00346         if (i == this->array->getSize() - 1)
00347             event.eventSquares[this->array->getSize() - 1].title = "n";
00348     }
00349     event.eventSquares[chosenNode].status = Square::Status::chosen;
00350     event.eventSquares[chosenNode].isPrintPreVal = true;
00351 
00352     events.emplace_back(event);
00353 
00354     event.eventSquares[chosenNode].isPrintPreVal = false;
00355 
```

```
    events.emplace_back(event);

    return events;
}
```

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

- include/libScene/DynamicArrayScene.hpp
- include/libScene/DynamicArrayScene.cpp

7.13 EventAnimation Class Reference

```
#include <EventAnimation.hpp>
```

Public Member Functions

- EventAnimation ()
- ∼EventAnimation ()
- EventAnimation & operator= (const EventAnimation &other)
- void reset ()

Public Attributes

- std::vector< std::pair< int, std::string > > titleNodes
- std::vector< std::pair< int, NodeInfo::ArrowType > > colorArrows
- std::vector< std::pair< int, NodeInfo::ArrowType > > hiddenArrows
- std::vector< int > colorNodes
- NodeInfo::StatusNode statusChosenNode
- bool isPrintPreVal
- bool isPrintNormal
- bool isShowBackArrow
- std::pair< int, int > indexBackArrow
- std::vector< EventSquare > eventSquares {}
- std::vector< EventSquare > eventSquaresTemp {}
- std::vector< int > lines

7.13.1 Detailed Description

Definition at line 20 of file EventAnimation.hpp.

7.13.2 Constructor & Destructor Documentation

7.13.2.1 EventAnimation()

```
EventAnimation::EventAnimation ( )
```

Definition at line 7 of file EventAnimation.cpp.

```
00007                                  {
00008     this->statusChosenNode = NodeInfo::StatusNode::InChain;
00009     this->isPrintPreVal = this->isPrintNormal = this->isShowBackArrow = false;
00010     this->indexBackArrow = {-1, -1};
00011 
00012     this->titleNodes = {};
00013     this->colorArrows = {};
00014     this->hiddenArrows = {};
00015     this->colorNodes = {};
00016     this->lines = {};
00017 
00018     this->eventSquares = {};
00019     this->eventSquaresTemp = {};
00020 }
```

7.13.2.2 ∼EventAnimation()

```
EventAnimation::∼EventAnimation ( )
```

Definition at line 37 of file EventAnimation.cpp.

```
00037                                   {
00038     this->titleNodes = {};
00039     this->colorArrows = {};
00040     this->hiddenArrows = {};
00041     this->colorNodes = {};
00042     this->lines = {};
00043 
00044     this->eventSquares = {};
00045     this->eventSquaresTemp = {};
00046 }
```

7.13.3 Member Function Documentation

7.13.3.1 operator=()

```
EventAnimation & EventAnimation::operator= (
              const EventAnimation & other )   [default]
```

7.13.3.2 reset()

```
void EventAnimation::reset ( )
```

Definition at line 22 of file EventAnimation.cpp.

```
00022                               {
00023     this->titleNodes.clear();
00024     this->colorArrows.clear();
00025     this->hiddenArrows.clear();
00026     this->colorNodes.clear();
00027     this->lines.clear();
00028 
00029     this->statusChosenNode = NodeInfo::StatusNode::InChain;
00030     this->isPrintPreVal = this->isPrintNormal = this->isShowBackArrow = false;
00031     this->indexBackArrow = {-1, -1};
00032 
00033     this->eventSquares.clear();
00034     this->eventSquaresTemp.clear();
00035 }
```

7.13.4 Member Data Documentation

7.13.4.1 colorArrows

```
std::vector<std::pair<int, NodeInfo::ArrowType> > EventAnimation::colorArrows
```

Definition at line 24 of file EventAnimation.hpp.

7.13.4.2 colorNodes

```
std::vector<int> EventAnimation::colorNodes
```

Definition at line 26 of file EventAnimation.hpp.

7.13.4.3 eventSquares

```
std::vector<EventSquare> EventAnimation::eventSquares {}
```

Definition at line 32 of file EventAnimation.hpp.

7.13.4.4 eventSquaresTemp

```
std::vector<EventSquare> EventAnimation::eventSquaresTemp {}
```

Definition at line 32 of file EventAnimation.hpp.

7.13.4.5 hiddenArrows

```
std::vector<std::pair<int, NodeInfo::ArrowType> > EventAnimation::hiddenArrows
```

Definition at line 25 of file EventAnimation.hpp.

7.13.4.6 indexBackArrow

```
std::pair<int, int> EventAnimation::indexBackArrow
```

Definition at line 29 of file EventAnimation.hpp.

7.13.4.7 isPrintNormal

```
bool EventAnimation::isPrintNormal
```

Definition at line 28 of file EventAnimation.hpp.

7.13.4.8 isPrintPreVal

```
bool EventAnimation::isPrintPreVal
```

Definition at line 28 of file EventAnimation.hpp.

7.13.4.9 isShowBackArrow

```
bool EventAnimation::isShowBackArrow
```

Definition at line 28 of file EventAnimation.hpp.

7.13.4.10 lines

```
std::vector<int> EventAnimation::lines
```

Definition at line 34 of file EventAnimation.hpp.

7.13.4.11 statusChosenNode

```
NodeInfo::StatusNode EventAnimation::statusChosenNode
```

Definition at line 27 of file EventAnimation.hpp.

7.13.4.12 titleNodes

```
std::vector<std::pair<int, std::string> > EventAnimation::titleNodes
```

Definition at line 23 of file EventAnimation.hpp.

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

- include/core/EventAnimation.hpp
- include/core/EventAnimation.cpp

7.14 EventSquare Struct Reference

```
#include <EventAnimation.hpp>
```

Public Member Functions

- EventSquare ()=default
- ∼EventSquare ()=default

Public Attributes

- Square::Status status = Square::Status::inactive
- bool isPrintPreVal = false
- std::string title {}

7.14.1 Detailed Description

Definition at line 11 of file EventAnimation.hpp.

7.14.2 Constructor & Destructor Documentation

7.14.2.1 EventSquare()

```
EventSquare::EventSquare ( )  [default]
```

7.14.2.2 ∼EventSquare()

```
EventSquare::∼EventSquare ( )  [default]
```

7.14.3 Member Data Documentation

7.14.3.1 isPrintPreVal

```
bool EventSquare::isPrintPreVal = false
```

Definition at line 13 of file EventAnimation.hpp.

7.14.3.2 status

```
Square::Status EventSquare::status = Square::Status::inactive
```

Definition at line 12 of file EventAnimation.hpp.

7.14.3.3 title

```
std::string EventSquare::title {}
```

Definition at line 14 of file EventAnimation.hpp.

The documentation for this struct was generated from the following file:

- include/core/EventAnimation.hpp

7.15 pfd::internal::executor Class Reference

```
#include <FileDialog.h>
```

Public Member Functions

- std::string result (int ∗exit_code=nullptr)
- bool kill ()
- void start_process (std::vector< std::string > const &command)
- ∼executor ()

Protected Member Functions

- bool ready (int timeout=default_wait_timeout)
- void stop ()

Friends

- class dialog

7.15.1 Detailed Description

Definition at line 166 of file FileDialog.h.

7.15.2 Constructor & Destructor Documentation

7.15.2.1 ∼executor()

```
pfd::internal::executor::∼executor ( )  [inline]
```

Definition at line 809 of file FileDialog.h.

```
    {
        stop();
    }
```

7.15.3 Member Function Documentation

7.15.3.1 kill()

```
bool pfd::internal::executor::kill ( )  [inline]
```

Definition at line 698 of file FileDialog.h.

```
    {
#if _WIN32
        if (m_future.valid())
    {
        // Close all windows that werent open when we started the future
        auto previous_windows = m_windows;
        EnumWindows(&enum_windows_callback, (LPARAM)this);
        for (auto hwnd : m_windows)
            if (previous_windows.find(hwnd) == previous_windows.end())
            {
                SendMessage(hwnd, WM_CLOSE, 0, 0);
                // Also send IDNO in case of a Yes/No or Abort/Retry/Ignore messagebox
                SendMessage(hwnd, WM_COMMAND, IDNO, 0);
            }
    }
#elif __EMSCRIPTEN__ || __NX__
        // FIXME: do something
    return false; // cannot kill
#else
        ::kill(m_pid, SIGKILL);
#endif
        stop();
        return true;
    }
```

7.15.3.2 ready()

```
bool pfd::internal::executor::ready (
            int timeout = default_wait_timeout )  [inline], [protected]
```

Definition at line 814 of file FileDialog.h.

```
    {
        if (!m_running)
            return true;

#if _WIN32
        if (m_future.valid())
    {
        auto status = m_future.wait_for(std::chrono::milliseconds(timeout));
        if (status != std::future_status::ready)
        {
            // On Windows, we need to run the message pump. If the async
            // thread uses a Windows API dialog, it may be attached to the
            // main thread and waiting for messages that only we can dispatch.
            MSG msg;
            while (PeekMessage(&msg, nullptr, 0, 0, PM_REMOVE))
            {
                TranslateMessage(&msg);
                DispatchMessage(&msg);
            }
            return false;
        }

        m_stdout = m_future.get();
    }
#elif __EMSCRIPTEN__ || __NX__
        // FIXME: do something
    (void)timeout;
#else
        char buf[BUFSIZ];
        ssize_t received = read(m_fd, buf, BUFSIZ); // Flawfinder: ignore
        if (received > 0)
        {
            m_stdout += std::string(buf, received);
            return false;
        }

        // Reap child process if it is dead. It is possible that the system has already reaped it
        // (this happens when the calling application handles or ignores SIG_CHLD) and results in
        // waitpid() failing with ECHILD. Otherwise we assume the child is running and we sleep for
        // a little while.
        int status;
        pid_t child = waitpid(m_pid, &status, WNOHANG);
        if (child != m_pid && (child >= 0 || errno != ECHILD))
        {
            // FIXME: this happens almost always at first iteration
            std::this_thread::sleep_for(std::chrono::milliseconds(timeout));
            return false;
        }

        close(m_fd);
        m_exit_code = WEXITSTATUS(status);
#endif

        m_running = false;
        return true;
    }
```

7.15.3.3 result()

```
std::string pfd::internal::executor::result (
            int * exit_code = nullptr )  [inline]
```

Definition at line 690 of file FileDialog.h.

```
    {
        stop();
        if (exit_code)
            *exit_code = m_exit_code;
        return m_stdout;
    }
```

7.15.3.4 start_process()

```
void pfd::internal::executor::start_process (
            std::vector< std::string > const & command )  [inline]
```

Definition at line 764 of file FileDialog.h.

```
    {
        stop();
        m_stdout.clear();
        m_exit_code = -1;

        int in[2], out[2];
        if (pipe(in) != 0 || pipe(out) != 0)
            return;

        m_pid = fork();
        if (m_pid < 0)
            return;

        close(in[m_pid ? 0 : 1]);
        close(out[m_pid ? 1 : 0]);

        if (m_pid == 0)
        {
            dup2(in[0], STDIN_FILENO);
            dup2(out[1], STDOUT_FILENO);

            // Ignore stderr so that it doesnt pollute the console (e.g. GTK+ errors from zenity)
            int fd = open("/dev/null", O_WRONLY);
            dup2(fd, STDERR_FILENO);
            close(fd);

            std::vector<char *> args;
            std::transform(command.cbegin(), command.cend(), std::back_inserter(args),
                           [](std::string const &s) { return const_cast<char *>(s.c_str()); });
            args.push_back(nullptr); // null-terminate argv[]

            execvp(args[0], args.data());
            exit(1);
        }

        close(in[1]);
        m_fd = out[0];
        auto flags = fcntl(m_fd, F_GETFL);
        fcntl(m_fd, F_SETFL, flags | O_NONBLOCK);

        m_running = true;
    }
```

7.15.3.5 stop()

```
void pfd::internal::executor::stop ( )  [inline], [protected]
```

Definition at line 872 of file FileDialog.h.

```
    {
        // Loop until the user closes the dialog
        while (!ready())
            ;
    }
```

7.15.4 Friends And Related Function Documentation

7.15.4.1 dialog

```
friend class dialog  [friend]
```

Definition at line 168 of file FileDialog.h.

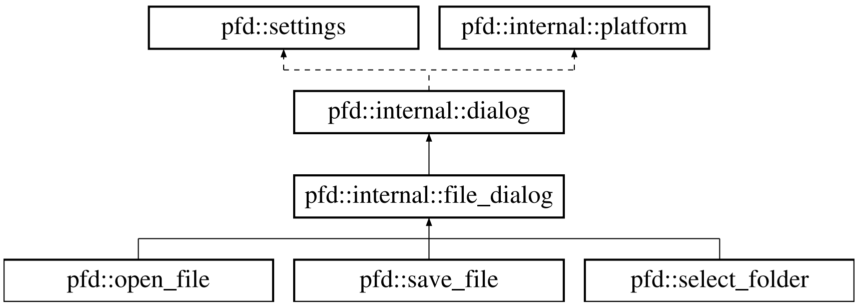
The documentation for this class was generated from the following file:

- include/core/FileDialog.h

7.16 pfd::internal::file_dialog Class Reference

```
#include <FileDialog.h>
```

Inheritance diagram for pfd::internal::file_dialog:



Protected Types

- enum type { open , save , folder }

Protected Types inherited from pfd::settings

- enum class flag {
  is_scanned = 0 , is_verbose , has_zenity , has_matedialog ,
  has_qarma , has_kdialog , is_vista , max_flag }

Protected Member Functions

- file_dialog (type in_type, std::string const &title, std::string const &default_path="", std::vector< std::string > const &filters={}, opt options=opt::none)
- std::string string_result ()
- std::vector< std::string > vector_result ()

Protected Member Functions inherited from pfd::internal::dialog

- dialog ()
- std::vector< std::string > desktop_helper () const
- std::string powershell_quote (std::string const &str) const
- std::string osascript_quote (std::string const &str) const
- std::string shell_quote (std::string const &str) const

Protected Member Functions inherited from pfd::settings

- settings (bool resync=false)
- bool check_program (std::string const &program)
- bool is_osascript () const
- bool is_zenity () const
- bool is_kdialog () const
- bool const & flags (flag in_flag) const
- bool & flags (flag in_flag)

Additional Inherited Members

Public Member Functions inherited from pfd::internal::dialog

- bool ready (int timeout=default_wait_timeout) const
- bool kill () const

Static Protected Member Functions inherited from pfd::internal::dialog

- static std::string buttons_to_name (choice _choice)
- static std::string get_icon_name (icon _icon)

Static Protected Member Functions inherited from pfd::settings

- static bool available ()
- static void verbose (bool value)
- static void rescan ()

Protected Attributes inherited from pfd::internal::dialog

- std::shared_ptr< executor > m_async

7.16.1 Detailed Description

Definition at line 286 of file FileDialog.h.

7.16.2 Member Enumeration Documentation

7.16.2.1 type

```
enum pfd::internal::file_dialog::type  [protected]
```

Enumerator

open	
save	
folder	

Definition at line 289 of file FileDialog.h.

```
            {
                open,
                save,
                folder,
            };
```

7.16.3 Constructor & Destructor Documentation

7.16.3.1 file_dialog()

```
pfd::internal::file_dialog::file_dialog (
            type in_type,
            std::string const & title,
            std::string const & default_path = "",
            std::vector< std::string > const & filters = {},
            opt options = opt::none )  [inline], [protected]
```

Definition at line 1058 of file FileDialog.h.

```
    {
#if _WIN32
        std::string filter_list;
    std::regex whitespace("  *");
    for (size_t i = 0; i + 1 < filters.size(); i += 2)
    {
        filter_list += filters[i] + '\0';
        filter_list += std::regex_replace(filters[i + 1], whitespace, ";") + '\0';
    }
    filter_list += '\0';

    m_async->start_func([this, in_type, title, default_path, filter_list,
                         options](int *exit_code) -> std::string
    {
        (void)exit_code;
        m_wtitle = internal::str2wstr(title);
        m_wdefault_path = internal::str2wstr(default_path);
        auto wfilter_list = internal::str2wstr(filter_list);

        // Initialise COM. This is required for the new folder selection window,
        // (see https://github.com/samhocevar/portable-file-dialogs/pull/21)
        // and to avoid random crashes with GetOpenFileNameW() (see
        // https://github.com/samhocevar/portable-file-dialogs/issues/51)
        ole32_dll ole32;

        // Folder selection uses a different method
        if (in_type == type::folder)
        {
#if PFD_HAS_IFILEDIALOG
            if (flags(flag::is_vista))
            {
                // On Vista and higher we should be able to use IFileDialog for folder selection
                IFileDialog *ifd;
                HRESULT hr = dll::proc<HRESULT WINAPI (REFCLSID, LPUNKNOWN, DWORD, REFIID, LPVOID
*)>(ole32, "CoCreateInstance")
                                 (CLSID_FileOpenDialog, nullptr, CLSCTX_INPROC_SERVER,
IID_PPV_ARGS(&ifd));

                // In case CoCreateInstance fails (which it should not), try legacy approach
                if (SUCCEEDED(hr))
                    return select_folder_vista(ifd, options & opt::force_path);
            }
#endif

            BROWSEINFOW bi;
            memset(&bi, 0, sizeof(bi));

            bi.lpfn = &bffcallback;
            bi.lParam = (LPARAM)this;

            if (flags(flag::is_vista))
            {
                if (ole32.is_initialized())
                    bi.ulFlags |= BIF_NEWDIALOGSTYLE;
                bi.ulFlags |= BIF_EDITBOX;
                bi.ulFlags |= BIF_STATUSTEXT;
            }

            auto *list = SHBrowseForFolderW(&bi);
            std::string ret;
            if (list)
            {
                auto buffer = new wchar_t[MAX_PATH];
                SHGetPathFromIDListW(list, buffer);
                dll::proc<void WINAPI (LPVOID)>(ole32, "CoTaskMemFree")(list);
                ret = internal::wstr2str(buffer);
                delete[] buffer;
            }
            return ret;
        }

        OPENFILENAMEW ofn;
        memset(&ofn, 0, sizeof(ofn));
        ofn.lStructSize = sizeof(OPENFILENAMEW);
        ofn.hwndOwner = GetActiveWindow();

        ofn.lpstrFilter = wfilter_list.c_str();

        auto woutput = std::wstring(MAX_PATH * 256, L'\0');
        ofn.lpstrFile = (LPWSTR)woutput.data();
        ofn.nMaxFile = (DWORD)woutput.size();
        if (!m_wdefault_path.empty())
        {
            // If a directory was provided, use it as the initial directory. If
            // a valid path was provided, use it as the initial file. Otherwise,
            // let the Windows API decide.
            auto path_attr = GetFileAttributesW(m_wdefault_path.c_str());
            if (path_attr != INVALID_FILE_ATTRIBUTES && (path_attr & FILE_ATTRIBUTE_DIRECTORY))
                ofn.lpstrInitialDir = m_wdefault_path.c_str();
            else if (m_wdefault_path.size() <= woutput.size())
                //second argument is size of buffer, not length of string
                StringCchCopyW(ofn.lpstrFile, MAX_PATH*256+1, m_wdefault_path.c_str());
            else
            {
                ofn.lpstrFileTitle = (LPWSTR)m_wdefault_path.data();
                ofn.nMaxFileTitle = (DWORD)m_wdefault_path.size();
            }
        }
        ofn.lpstrTitle = m_wtitle.c_str();
        ofn.Flags = OFN_NOCHANGEDIR | OFN_EXPLORER;

        dll comdlg32("comdlg32.dll");

        // Apply new visual style (required for windows XP)
        new_style_context ctx;

        if (in_type == type::save)
        {
            if (!(options & opt::force_overwrite))
                ofn.Flags |= OFN_OVERWRITEPROMPT;

            dll::proc<BOOL WINAPI (LPOPENFILENAMEW)> get_save_file_name(comdlg32, "GetSaveFileNameW");
            if (get_save_file_name(&ofn) == 0)
                return "";
            return internal::wstr2str(woutput.c_str());
        }
        else
        {
            if (options & opt::multiselect)
                ofn.Flags |= OFN_ALLOWMULTISELECT;
            ofn.Flags |= OFN_PATHMUSTEXIST;

            dll::proc<BOOL WINAPI (LPOPENFILENAMEW)> get_open_file_name(comdlg32, "GetOpenFileNameW");
            if (get_open_file_name(&ofn) == 0)
                return "";
        }

        std::string prefix;
        for (wchar_t const *p = woutput.c_str(); *p; )
        {
            auto filename = internal::wstr2str(p);
            p += wcslen(p);
            // In multiselect mode, we advance p one wchar further and
            // check for another filename. If there is one and the
            // prefix is empty, it means we just read the prefix.
            if ((options & opt::multiselect) && *++p && prefix.empty())
            {
                prefix = filename + "/";
                continue;
            }

            m_vector_result.push_back(prefix + filename);
        }

        return "";
    });
#elif __EMSCRIPTEN__
        // FIXME: do something
    (void)in_type;
    (void)title;
    (void)default_path;
    (void)filters;
    (void)options;
#else
        auto command = desktop_helper();

        if (is_osascript())
        {
            std::string script = "set ret to choose";
            switch (in_type)
            {
                case type::save:
                    script += " file name";
                    break;
                case type::open: default:
                    script += " file";
                    if (options & opt::multiselect)
                        script += " with multiple selections allowed";
                    break;
                case type::folder:
                    script += " folder";
                    break;
            }

            if (default_path.size())
            {
                if (in_type == type::folder || is_directory(default_path))
                    script += " default location ";
                else
                    script += " default name ";
                script += osascript_quote(default_path);
            }

            script += " with prompt " + osascript_quote(title);

            if (in_type == type::open)
            {
                // Concatenate all user-provided filter patterns
                std::string patterns;
                for (size_t i = 0; i < filters.size() / 2; ++i)
                    patterns += " " + filters[2 * i + 1];

                // Split the pattern list to check whether "*" is in there; if it
                // is, we have to disable filters because there is no mechanism in
                // OS X for the user to override the filter.
                std::regex sep("\\s+");
                std::string filter_list;
                bool has_filter = true;
                std::sregex_token_iterator iter(patterns.begin(), patterns.end(), sep, -1);
                std::sregex_token_iterator end;
                for ( ; iter != end; ++iter)
                {
                    auto pat = iter->str();
                    if (pat == "*" || pat == "*.*")
                        has_filter = false;
                    else if (internal::starts_with(pat, "*."))
                        filter_list += "," + osascript_quote(pat.substr(2, pat.size() - 2));
                }

                if (has_filter && filter_list.size() > 0)
                {
                    // There is a weird AppleScript bug where file extensions of length != 3 are
                    // ignored, e.g. type{"txt"} works, but type{"json"} does not. Fortunately if
                    // the whole list starts with a 3-character extension, everything works again.
                    // We use "///" for such an extension because we are sure it cannot appear in
                    // an actual filename.
                    script += " of type {\"///\"" + filter_list + "}";
                }
            }

            if (in_type == type::open && (options & opt::multiselect))
            {
                script += "\nset s to \"\"";
                script += "\nrepeat with i in ret";
                script += "\n  set s to s & (POSIX path of i) & \"\\n\"";
                script += "\nend repeat";
                script += "\ncopy s to stdout";
            }
            else
            {
                script += "\nPOSIX path of ret";
            }

            command.push_back("-e");
            command.push_back(script);
        }
        else if (is_zenity())
        {
            command.push_back("--file-selection");

            // If the default path is a directory, make sure it ends with "/" otherwise zenity will
            // open the file dialog in the parent directory.
            auto filename_arg = "--filename=" + default_path;
            if (in_type != type::folder && !ends_with(default_path, "/") &&
internal::is_directory(default_path))
                filename_arg += "/";
            command.push_back(filename_arg);

            command.push_back("--title");
            command.push_back(title);
            command.push_back("--separator=\n");

            for (size_t i = 0; i < filters.size() / 2; ++i)
            {
                command.push_back("--file-filter");
                command.push_back(filters[2 * i] + "|" + filters[2 * i + 1]);
            }

            if (in_type == type::save)
                command.push_back("--save");
            if (in_type == type::folder)
                command.push_back("--directory");
            if (!(options & opt::force_overwrite))
                command.push_back("--confirm-overwrite");
            if (options & opt::multiselect)
                command.push_back("--multiple");
        }
        else if (is_kdialog())
        {
            switch (in_type)
            {
                case type::save: command.push_back("--getsavefilename"); break;
                case type::open: command.push_back("--getopenfilename"); break;
                case type::folder: command.push_back("--getexistingdirectory"); break;
            }
            if (options & opt::multiselect)
            {
                command.push_back("--multiple");
                command.push_back("--separate-output");
            }

            command.push_back(default_path);

            std::string filter;
            for (size_t i = 0; i < filters.size() / 2; ++i)
                filter += (i == 0 ? "" : " | ") + filters[2 * i] + "(" + filters[2 * i + 1] + ")";
            command.push_back(filter);

            command.push_back("--title");
            command.push_back(title);
        }

        if (flags(flag::is_verbose))
            std::cerr « "pfd: " « command « std::endl;

        m_async->start_process(command);
#endif
    }
```

7.16.4 Member Function Documentation

7.16.4.1 string_result()

`std::string pfd::internal::file_dialog::string_result ( ) [inline], [protected]`

Definition at line 1359 of file FileDialog.h.

```
    {
#if _WIN32
        return m_async->result();
#else
        auto ret = m_async->result();
        // Strip potential trailing newline (zenity). Also strip trailing slash
        // added by osascript for consistency with other backends.
        while (!ret.empty() && (ret.back() == '\n' || ret.back() == '/'))
            ret.pop_back();
        return ret;
#endif
    }
```

7.16.4.2 vector_result()

`std::vector< std::string > pfd::internal::file_dialog::vector_result ( ) [inline], [protected]`

Definition at line 1373 of file FileDialog.h.

```
    {
#if _WIN32
        m_async->result();
    return m_vector_result;
#else
        std::vector<std::string> ret;
        auto result = m_async->result();
        for (;;)
        {
            // Split result along newline characters
            auto i = result.find('\n');
            if (i == 0 || i == std::string::npos)
                break;
            ret.push_back(result.substr(0, i));
            result = result.substr(i + 1, result.size());
        }
        return ret;
#endif
    }
```

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

- include/core/FileDialog.h

7.17 Highlighter Class Reference

`#include <Highlighter.hpp>`

Public Member Functions

- Highlighter (sf::RenderWindow ∗window, int linesCount, const char ∗codePath)
- void toggle (std::vector< int > lines)
- void resetToggle ()
- void render ()

7.17.1 Detailed Description

Definition at line 11 of file Highlighter.hpp.

7.17.2 Constructor & Destructor Documentation

7.17.2.1 Highlighter()

```
Highlighter::Highlighter (
            sf::RenderWindow * window,
            int linesCount,
            const char * codePath )
```

Definition at line 7 of file Highlighter.cpp.

```
                                                                          {
    this->window = window;
    this->linesCount = linesCount;

    this->codeTexture.loadFromFile(codePath);
    this->codeSprite.setTexture(this->codeTexture);
    this->codeSprite.setScale(constants::Highlighter::codeScale);

    this->codeSprite.setOrigin(
            this->codeSprite.getLocalBounds().width,
            this->codeSprite.getLocalBounds().height
            );

    this->codeSprite.setPosition(constants::Highlighter::codePos);

    float heightTop = 43;

    this->rectSize = sf::Vector2f(
            this->codeSprite.getGlobalBounds().width,
            ((this->codeSprite.getLocalBounds().height - heightTop * 2) /
      static_cast<float>(this->linesCount)) * constants::Highlighter::codeScale.y
            );

    for (int i = 0; i < this->linesCount; ++i) {
        sf::RectangleShape rect(this->rectSize);
        rect.setOrigin(rect.getLocalBounds().width, rect.getLocalBounds().height);
        rect.setFillColor(constants::transparentGreen);
        rect.setPosition(
                this->codeSprite.getPosition().x,
                this->codeSprite.getPosition().y - (heightTop * constants::Highlighter::codeScale.y) -
      static_cast<float>(this->linesCount - 1 - i) * this->rectSize.y
                );
        this->lines.push_back(rect);
    }
}
```

7.17.3 Member Function Documentation

7.17.3.1 render()

```
void Highlighter::render ( )
```

Definition at line 45 of file Highlighter.cpp.

```
                        {
    this->window->draw(this->codeSprite);

    for (auto &i : this->toggleLines) {
        this->window->draw(this->lines[i]);
    }
}
```

7.17.3.2 resetToggle()

```
void Highlighter::resetToggle ( )
```

Definition at line 53 of file Highlighter.cpp.

```
                                {
    this->toggleLines.clear();
}
```

7.17.3.3 toggle()

```
void Highlighter::toggle (
            std::vector< int > lines )
```

Definition at line 41 of file Highlighter.cpp.

```
                                                {
    this->toggleLines = std::move(linesList);
}
```

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

- include/libScene/Highlighter.hpp
- include/libScene/Highlighter.cpp

7.18 LinkedList Class Reference

```
#include <LinkedList.hpp>
```

Public Types

- enum class TypeLinkedList { SINGLY , DOUBLY , CIRCULAR }

Public Member Functions

- LinkedList (sf::RenderWindow ∗window, TypeLinkedList typeLinkedList)
- LinkedList (sf::RenderWindow ∗window, TypeLinkedList typeLinkedList, int size)
- LinkedList (sf::RenderWindow ∗window, TypeLinkedList typeLinkedList, std::vector< std::string > values)
- void setSpeed (float speed)
- int findValue (const std::string &value)
- sf::Vector2f getPosNode (int position)
- int getSize () const
- void update ()
- void updateAnimation ()
- void render ()
- void renderHighlighter ()
- void resetEvents ()
- void calculateEffectivePositions ()
- void clear ()
- void processControlMenu (ControlMenu::StatusCode status)
- void initHighlighter (int linesCount, const char ∗codePath)
- void toggleLines (std::vector< int > lines)
- void createLinkedList (int size)
- void createLinkedList (std::vector< std::string > values)
- void addNode (int position, std::string value, const std::vector< EventAnimation > &listEvents)
- void deleteNode (int position, const std::vector< EventAnimation > &listEvents)
- void updateNode (int position, std::string value, const std::vector< EventAnimation > &listEvents)
- void searchNode (const std::vector< EventAnimation > &listEvents)

7.18.1 Detailed Description

Definition at line 16 of file LinkedList.hpp.

7.18.2 Member Enumeration Documentation

7.18.2.1 TypeLinkedList

```
enum class LinkedList::TypeLinkedList [strong]
```

Enumerator

SINGLY	
DOUBLY	
CIRCULAR	

Definition at line 18 of file LinkedList.hpp.

```
                          {
    SINGLY,
    DOUBLY,
    CIRCULAR
};
```

7.18.3 Constructor & Destructor Documentation

7.18.3.1 LinkedList() [1/3]

```
LinkedList::LinkedList (
        sf::RenderWindow * window,
        TypeLinkedList typeLinkedList ) [explicit]
```

Definition at line 7 of file LinkedList.cpp.

```
                                                                          {
    this->window = window;
    this->typeLinkedList = typeLinkedList;
    this->highlighter = nullptr;
    this->delayTime = constants::LinkedList::DELAY_TIME;
    this->backArrow = new BackArrow(this->window, {0, 0}, {0, 0});

    if (this->typeLinkedList == TypeLinkedList::CIRCULAR)
        this->backArrow->show();
    else
        this->backArrow->hide();

    this->createLinkedList(0);
}
```

7.18.3.2 LinkedList() [2/3]

```
LinkedList::LinkedList (
            sf::RenderWindow * window,
            TypeLinkedList typeLinkedList,
            int size )
```

Definition at line 39 of file LinkedList.cpp.

```
00039                                                                                           {
00040     this->window = window;
00041     this->typeLinkedList = typeLinkedList;
00042     this->highlighter = nullptr;
00043     this->delayTime = constants::LinkedList::DELAY_TIME;
00044     this->backArrow = new BackArrow(this->window, {0, 0}, {0, 0});
00045 
00046     if (this->typeLinkedList == TypeLinkedList::CIRCULAR)
00047         this->backArrow->show();
00048     else
00049         this->backArrow->hide();
00050 
00051     this->createLinkedList(size);
00052 }
```

7.18.3.3 LinkedList() [3/3]

```
LinkedList::LinkedList (
            sf::RenderWindow * window,
            TypeLinkedList typeLinkedList,
            std::vector< std::string > values )
```

Definition at line 54 of file LinkedList.cpp.

```
00054
      {
00055     this->window = window;
00056     this->typeLinkedList = typeLinkedList;
00057     this->highlighter = nullptr;
00058     this->delayTime = constants::LinkedList::DELAY_TIME;
00059     this->backArrow = new BackArrow(this->window, {0, 0}, {0, 0});
00060 
00061     if (this->typeLinkedList == TypeLinkedList::CIRCULAR)
00062         this->backArrow->show();
00063     else
00064         this->backArrow->hide();
00065 
00066     this->createLinkedList(std::move(values));
00067 }
```

7.18.4 Member Function Documentation

7.18.4.1 addNode()

```
void LinkedList::addNode (
            int position,
            std::string value,
            const std::vector< EventAnimation > & listEvents )
```

Definition at line 320 of file LinkedList.cpp.

```
00320                                                                                                      {
00321     if (position < 0 || position > this->size) return;
```

```

    sf::Vector2f newPosition(
            constants::NodeInfo::originNode.x + static_cast<float>(this->nodes.size()) *
constants::NodeInfo::offsetX,
            constants::NodeInfo::originNode.y
    );
    if (this->size) {
        this->nodes.back()->initArrow(
                NodeInfo::ArrowType::RIGHT,
                this->nodes.back()->getPosition(),
                newPosition
        );
    }
    this->nodes.push_back(new NodeInfo(
            this->window,
            "10",
            newPosition,
            this->typeLinkedList == TypeLinkedList::DOUBLY
    ));
    ++this->size;
    if (this->typeLinkedList == TypeLinkedList::DOUBLY && this->size > 1)
        this->nodes.back()->initArrow(
                NodeInfo::ArrowType::LEFT,
                this->nodes.back()->getPosition(),
                this->nodes[this->nodes.size() - 2]->getPosition()
        );
    this->backArrow->setPosition(newPosition, this->nodes[0]->getPosition());
    for (int i = this->size - 1; i > position; --i) {
        this->nodes[i]->setValue(this->nodes[i - 1]->getValue());
        this->nodes[i]->reInitPreVal();
    }
    this->nodes[position]->setValue(std::move(value));
//    std::cout « "add node to the current list " « position « " " « this->nodes[position]->getValue()
« std::endl;

    this->chosenNode = position;
    this->currentEvent = 0;

    for (auto &e : listEvents)
        this->events.emplace_back(e);
}
```

7.18.4.2 calculateEffectivePositions()

void LinkedList::calculateEffectivePositions ( )

Definition at line 147 of file LinkedList.cpp.

```
                                              {
    if (this->size < 2) return;

    int lastInChain = 0;
    if (this->nodes[lastInChain]->getStatusNode() != NodeInfo::StatusNode::InChain){
        lastInChain++;
    }

    this->nodes[lastInChain]->setEffectivePosition(
            sf::Vector2f(
                    constants::NodeInfo::originNode.x,
                    constants::NodeInfo::originNode.y
            )
    );

    for (int i = lastInChain + 1; i < this->size; i++){
        if (this->nodes[i]->getStatusNode() == NodeInfo::StatusNode::InChain){
            this->nodes[i]->setEffectivePosition(
                    sf::Vector2f(
                            this->nodes[lastInChain]->getPosition().x + constants::NodeInfo::offsetX,
                            this->nodes[lastInChain]->getPosition().y
                    )
            );
            lastInChain = i;
        }
    }
}
```

7.18.4.3 clear()

```
void LinkedList::clear ( )
```

Definition at line 22 of file LinkedList.cpp.

```
                            {
    for (auto &node : this->nodes)
        delete node;
    this->nodes.clear();
    this->size = 0;
}
```

7.18.4.4 createLinkedList() [1/2]

```
void LinkedList::createLinkedList (
              int size )
```

Definition at line 201 of file LinkedList.cpp.

```
                                            {
    this->resetEvents();
    this->size = _size;
    for (auto &node : this->nodes)
        delete node;
    this->nodes.resize(_size);
    for (int i = 0; i < size; i++){
        this->nodes[i] = new NodeInfo(
                this->window,
                std::to_string(Random::randomInt(0, 99)),
                sf::Vector2f(
                        constants::NodeInfo::originNode.x + static_cast<float>(i) *
 constants::NodeInfo::offsetX,
                        constants::NodeInfo::originNode.y
                ),
                this->typeLinkedList == TypeLinkedList::DOUBLY
        );
        if (i > 0){
            this->nodes[i - 1]->initArrow(
                    NodeInfo::ArrowType::RIGHT,
                    this->nodes[i - 1]->getPosition(),
                    this->nodes[i]->getPosition()
            );
            if (this->typeLinkedList == TypeLinkedList::DOUBLY)
                this->nodes[i]->initArrow(
                        NodeInfo::ArrowType::LEFT,
                        this->nodes[i]->getPosition(),
                        this->nodes[i - 1]->getPosition()
                );
        }
    }
    if (this->size > 1)
        this->backArrow->setPosition(this->nodes.back()->getPosition(),
 this->nodes[0]->getPosition());
}
```

7.18.4.5 createLinkedList() [2/2]

```
void LinkedList::createLinkedList (
              std::vector< std::string > values )
```

Definition at line 235 of file LinkedList.cpp.

```
                                                              {
    this->resetEvents();
    this->size = static_cast<int>(values.size());
    for (auto &node : this->nodes)
        delete node;
    this->nodes.resize(this->size);
```

```
00241     for (int i = 0; i < this->size; i++){
00242         this->nodes[i] = new NodeInfo(
00243                 this->window,
00244                 values[i],
00245                 sf::Vector2f(
00246                         constants::NodeInfo::originNode.x + static_cast<float>(i) *
      constants::NodeInfo::offsetX,
00247                         constants::NodeInfo::originNode.y
00248                 ),
00249                 this->typeLinkedList == TypeLinkedList::DOUBLY
00250         );
00251         if (i > 0){
00252             this->nodes[i - 1]->initArrow(
00253                     NodeInfo::ArrowType::RIGHT,
00254                     this->nodes[i - 1]->getPosition(),
00255                     this->nodes[i]->getPosition()
00256             );
00257             if (this->typeLinkedList == TypeLinkedList::DOUBLY)
00258                 this->nodes[i]->initArrow(
00259                         NodeInfo::ArrowType::LEFT,
00260                         this->nodes[i]->getPosition(),
00261                         this->nodes[i - 1]->getPosition()
00262                 );
00263         }
00264     }
00265     if (this->size > 1)
00266         this->backArrow->setPosition(this->nodes.back()->getPosition(),
      this->nodes[0]->getPosition());
00267 }
```

7.18.4.6 deleteNode()

```
void LinkedList::deleteNode (
              int position,
              const std::vector< EventAnimation > & listEvents )
```

Definition at line 362 of file LinkedList.cpp.

```
00362                                                                                           {
00363     if (position < 0 || position >= this->size) return;
00364 
00365     this->deletedNode = position;
00366     this->chosenNode = position;
00367     this->currentEvent = 0;
00368 
00369     for (auto &e : listEvents)
00370         this->events.emplace_back(e);
00371 }
```

7.18.4.7 findValue()

```
int LinkedList::findValue (
              const std::string & value )
```

Definition at line 392 of file LinkedList.cpp.

```
00392                                                     {
00393     for (int i = 0; i < this->size; ++i)
00394         if (this->nodes[i]->getValue() == value)
00395             return i;
00396     return this->size;
00397 }
```

7.18.4.8 getPosNode()

```
sf::Vector2f LinkedList::getPosNode (
             int position )
```

Definition at line 399 of file LinkedList.cpp.

```
                                         {
    if (position < 0 || position >= this->size) return {};
    return this->nodes[position]->getPosition();
}
```

7.18.4.9 getSize()

```
int LinkedList::getSize ( ) const
```

Definition at line 316 of file LinkedList.cpp.

```
                            {
    return this->size;
}
```

7.18.4.10 initHighlighter()

```
void LinkedList::initHighlighter (
             int linesCount,
             const char * codePath )
```

Definition at line 269 of file LinkedList.cpp.

```
                                                                     {
    delete this->highlighter;
    this->highlighter = new Highlighter(
            this->window,
            linesCount,
            codePath
    );
}
```

7.18.4.11 processControlMenu()

```
void LinkedList::processControlMenu (
             ControlMenu::StatusCode status )
```

Definition at line 287 of file LinkedList.cpp.

```
                                                                {
    if (this->clock.getElapsedTime().asSeconds() < this->delayTime / this->speed)
        return;
    switch (status){
        case ControlMenu::StatusCode::PREVIOUS:
            if (this->currentEvent > 0)
                --this->currentEvent;
            break;
        case ControlMenu::StatusCode::PAUSE:
//              std::cout « "PAUSE" « std::endl;
            break;
        case ControlMenu::StatusCode::PLAY:
            if (this->currentEvent + 1 < this->events.size()) {
                this->isDelay = true;
                this->clock.restart();
            }
        case ControlMenu::StatusCode::NEXT:
            if (this->currentEvent + 1 < this->events.size())
                ++this->currentEvent;
            break;
        default:
            break;
    }
}
```

7.18.4.12 render()

```
void LinkedList::render ( )
```

Definition at line 29 of file LinkedList.cpp.

```
                              {
    if (this->size > 1) {
//        this->backArrow->toggleActiveColorNode();
        this->backArrow->render();
    }
    for (auto &node : this->nodes){
        node->render();
    }
}
```

7.18.4.13 renderHighlighter()

```
void LinkedList::renderHighlighter ( )
```

Definition at line 282 of file LinkedList.cpp.

```
                                         {
    if (this->highlighter)
        this->highlighter->render();
}
```

7.18.4.14 resetEvents()

```
void LinkedList::resetEvents ( )
```

Definition at line 175 of file LinkedList.cpp.

```
                                   {
    delete this->highlighter;
    this->highlighter = nullptr;
    this->currentEvent = 0;
    this->events.clear();
    this->chosenNode = 0;

    if (this->deletedNode != -1){
        this->nodes.erase(this->nodes.begin() + this->deletedNode);
        --this->size;
        if (this->size && this->deletedNode == this->size)
            this->nodes.back()->destroyArrow(NodeInfo::ArrowType::RIGHT);
        if (this->size && this->deletedNode == 0)
            this->nodes[0]->destroyArrow(NodeInfo::ArrowType::LEFT);
    }
    this->deletedNode = -1;

    for (int i = 0;  i < this->size; i++){
        this->nodes[i]->reset();
        this->nodes[i]->reInitPos(i);
        this->nodes[i]->reInitPreVal();
    }
    if (this->size > 1)
        this->backArrow->setPosition(this->nodes.back()->getPosition(),
  this->nodes[0]->getPosition());
}
```

7.18.4.15 searchNode()

```
void LinkedList::searchNode (
            const std::vector< EventAnimation > & listEvents )
```

Definition at line 384 of file LinkedList.cpp.

```
                                                                          {
    this->chosenNode = 0;
    this->currentEvent = 0;

    for (auto &e : listEvents)
        this->events.emplace_back(e);
}
```

7.18.4.16 setSpeed()

```
void LinkedList::setSpeed (
            float speed )
```

Definition at line 312 of file LinkedList.cpp.

```
                                         {
    this->speed = _speed;
}
```

7.18.4.17 toggleLines()

```
void LinkedList::toggleLines (
            std::vector< int > lines )
```

Definition at line 278 of file LinkedList.cpp.

```
                                                    {
    this->highlighter->toggle(std::move(lines));
}
```

7.18.4.18 update()

```
void LinkedList::update ( )
```

Definition at line 69 of file LinkedList.cpp.

```
                              {
    if ((int)this->events.size() && (this->isDelay or this->clock.getElapsedTime().asSeconds() >
      this->delayTime / this->speed))
        this->updateAnimation();
    this->isDelay = false;
}
```

7.18.4.19 updateAnimation()

```
void LinkedList::updateAnimation ( )
```

Definition at line 75 of file LinkedList.cpp.

```
00075                                 {
00076     if (this->nodes.empty())
00077         return;
00078 
00079     // reset events of list
00080     for (auto &node : this->nodes){
00081         node->reset();
00082     }
00083 
00084     if (this->typeLinkedList == TypeLinkedList::CIRCULAR)
00085         this->backArrow->show();
00086 
00087     EventAnimation &event = this->events[this->currentEvent];
00088     for (auto &arrow: event.colorArrows)
00089         this->nodes[arrow.first]->toggleActiveColorArrow(arrow.second);
00090     for (auto &arrow : event.hiddenArrows)
00091         this->nodes[arrow.first]->hide(arrow.second);
00092     for (auto node : event.colorNodes)
00093         this->nodes[node]->toggleActiveColorNode();
00094     switch (event.statusChosenNode) {
00095         case NodeInfo::StatusNode::InChain:
00096             this->nodes[this->chosenNode]->setNodeInChain();
00097             break;
00098         case NodeInfo::StatusNode::OutChain:
00099             this->nodes[this->chosenNode]->setNodeOutside();
00100             break;
00101         case NodeInfo::StatusNode::Visible:
00102             this->nodes[this->chosenNode]->setNodeVisible();
00103             break;
00104     }
00105     if (event.isPrintPreVal)
00106         this->nodes[this->chosenNode]->setPrintPreVal();
00107     if (this->chosenNode < this->size - 1 && event.isPrintNormal)
00108         this->nodes[this->chosenNode + 1]->setPrintNormal();
00109 
00110     if (this->highlighter)
00111         this->highlighter->toggle(event.lines);
00112 
00113     this->calculateEffectivePositions();
00114 
00115     for (auto &node : this->nodes){
00116         node->updateNode();
00117     }
00118 
00119     for (auto &i : event.titleNodes) {
00120         this->nodes[i.first]->setTitle(i.second);
00121     }
00122 
00123     if (this->chosenNode < this->size - 1)
00124         this->nodes[this->chosenNode]->updateArrows(NodeInfo::ArrowType::RIGHT,
      this->nodes[this->chosenNode + 1]->getPosition());
00125     if (this->chosenNode > 0)
00126         this->nodes[this->chosenNode]->updateArrows(NodeInfo::ArrowType::LEFT,
      this->nodes[this->chosenNode - 1]->getPosition());
00127 
00128     if (event.indexBackArrow.first != -1 and event.indexBackArrow.second != -1)
00129         this->backArrow->setPosition(
00130                 this->nodes[event.indexBackArrow.first]->getPosition(),
00131                 this->nodes[event.indexBackArrow.second]->getPosition()
00132                 );
00133 
00134     int lastInChain = 0;
00135     if (this->nodes[lastInChain]->getStatusNode() != NodeInfo::StatusNode::InChain){
00136         lastInChain++;
00137     }
00138     for (int i = lastInChain + 1; i < this->size; i++){
00139         if (this->nodes[i]->getStatusNode() == NodeInfo::StatusNode::InChain) {
00140             this->nodes[lastInChain]->updateArrows(NodeInfo::ArrowType::RIGHT,
      this->nodes[i]->getPosition());
00141             this->nodes[i]->updateArrows(NodeInfo::ArrowType::LEFT,
      this->nodes[lastInChain]->getPosition());
00142             lastInChain = i;
00143         }
00144     }
00145 }
```

7.18.4.20 updateNode()

```
void LinkedList::updateNode (
            int position,
            std::string value,
            const std::vector< EventAnimation > & listEvents )
```

Definition at line 373 of file LinkedList.cpp.

```
{
    if (position < 0 || position >= this->size) return;

    this->nodes[position]->setValue(std::move(value));
    this->chosenNode = position;
    this->currentEvent = 0;

    for (auto &e : listEvents)
        this->events.emplace_back(e);
}
```

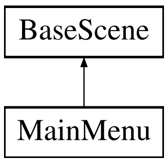
The documentation for this class was generated from the following files:

- include/core/LinkedList.hpp
- include/core/LinkedList.cpp

7.19 MainMenu Class Reference

`#include <MainMenu.hpp>`

Inheritance diagram for MainMenu:



Public Member Functions

- MainMenu (sf::RenderWindow ∗window)
- void pollEvent (sf::Event event, sf::Vector2f mousePosView) override
- void update () override
- void render () override

Public Member Functions inherited from BaseScene

- BaseScene (sf::RenderWindow ∗window)
- void createModeButton (sf::Vector2f position, std::string textString)
- virtual void pollEvent (sf::Event event, sf::Vector2f mousePosView)=0
- virtual void update ()=0
- virtual void render ()=0

Additional Inherited Members

Public Attributes inherited from BaseScene

- Button ∗ modeButton {}
- bool isMenuOpen {}
- bool isDemoCodeOpen {}

Protected Member Functions inherited from BaseScene

- void setWindow (sf::RenderWindow ∗window)

Protected Attributes inherited from BaseScene

- sf::RenderWindow ∗ window {}
- ControlMenu ∗ controlMenu

7.19.1 Detailed Description

Definition at line 10 of file MainMenu.hpp.

7.19.2 Constructor & Destructor Documentation

7.19.2.1 MainMenu()

```
MainMenu::MainMenu (
            sf::RenderWindow * window ) [explicit]
```

Definition at line 7 of file MainMenu.cpp.

```
00007                                                   : BaseScene(window) {
00008     this->modeButton = new Button;
00009 }
```

7.19.3 Member Function Documentation

7.19.3.1 pollEvent()

```
void MainMenu::pollEvent (
            sf::Event event,
            sf::Vector2f mousePosView ) [override], [virtual]
```

Implements BaseScene.

Definition at line 11 of file MainMenu.cpp.

```
00011                                                                       {
00012
00013 }
```

7.19.3.2 render()

```
void MainMenu::render ( )  [override], [virtual]
```

Implements BaseScene.

Definition at line 19 of file MainMenu.cpp.

```
00019                         {
00020
00021 }
```

7.19.3.3 update()

```
void MainMenu::update ( )  [override], [virtual]
```

Implements BaseScene.

Definition at line 15 of file MainMenu.cpp.

```
00015                         {
00016
00017 }
```

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

- include/libScene/MainMenu.hpp
- include/libScene/MainMenu.cpp

7.20 MenuArray Class Reference

```
#include <MenuArray.hpp>
```

Public Member Functions

- constants::MenuArray::CreateMode::Button getActiveCreateMode ()
- MenuArray (sf::RenderWindow ∗window, constants::MenuArray::Type _typeArray)
- ∼MenuArray ()=default
- void resetActiveOptionMenu ()
- void pollEvents (sf::Event event, sf::Vector2f mousePosView)
- void update ()
- void render ()
- Button ∗ getButton (int index)
- constants::MenuArray::Button getActiveOptionMenu ()

Public Attributes

- std::string createModeValue [constants::MenuArray::CreateMode::BUTTON_COUNT]
- std::string addModeValue [constants::MenuArray::AddMode::TEXTBOX_COUNT]
- std::string deleteModeValue
- std::string updateModeValue [constants::MenuArray::UpdateMode::TEXTBOX_COUNT]
- std::string searchModeValue
- std::string allocateModeValue

7.20.1 Detailed Description

Definition at line 14 of file MenuArray.hpp.

7.20.2 Constructor & Destructor Documentation

7.20.2.1 MenuArray()

```
MenuArray::MenuArray (
            sf::RenderWindow * window,
            constants::MenuArray::Type _typeArray )  [explicit]
```

Definition at line 7 of file MenuArray.cpp.

```
                                                                 {
    this->window = window;
    this->typeArray = _typeArray;
    this->init();
}
```

7.20.2.2 ∼MenuArray()

```
MenuArray::∼MenuArray ( )  [default]
```

7.20.3 Member Function Documentation

7.20.3.1 getActiveCreateMode()

```
constants::MenuArray::CreateMode::Button MenuArray::getActiveCreateMode ( )
```

Definition at line 158 of file MenuArray.cpp.

```
                                                         {
    return this->activeCreateMode;
}
```

7.20.3.2 getActiveOptionMenu()

```
constants::MenuArray::Button MenuArray::getActiveOptionMenu ( )
```

Definition at line 154 of file MenuArray.cpp.

```
                                                  {
    return this->activeOptionMenu;
}
```

7.20.3.3 getButton()

```
Button * MenuArray::getButton (
              int index )
```

Definition at line 150 of file MenuArray.cpp.

```
                                    {
    return this->buttons[index];
}
```

7.20.3.4 pollEvents()

```
void MenuArray::pollEvents (
              sf::Event event,
              sf::Vector2f mousePosView )
```

Definition at line 53 of file MenuArray.cpp.

```
                                                                   {
    if (this->activeOptionMenu != constants::MenuArray::Button::NONE)
        this->buttons[this->activeOptionMenu]->setColor(constants::normalGray);

    for (int i = 0; i < constants::MenuArray::BUTTON_COUNT; ++i) {
        if (this->buttons[i]->pollEvent(mousePosView)) {
            std::cout « "Button " « i « " is clicked" « std::endl;
            this->activeOptionMenu = static_cast<constants::MenuArray::Button>(i);
            this->activeAddMode = constants::MenuArray::AddMode::Textbox::NONE;
        }
    }

    switch (this->activeOptionMenu) {
        case constants::MenuArray::Button::CREATE_BUTTON:
            this->pollEventCreateMode(event, mousePosView);
            break;
        case constants::MenuArray::Button::ADD_BUTTON:
            this->pollEventAddMode(event, mousePosView);
            break;
        case constants::MenuArray::Button::DELETE_BUTTON:
            this->pollEventDeleteMode(event, mousePosView);
            break;
        case constants::MenuArray::Button::UPDATE_BUTTON:
            this->pollEventUpdateMode(event, mousePosView);
            break;
        case constants::MenuArray::Button::SEARCH_BUTTON:
            this->pollEventSearchMode(event, mousePosView);
            break;
        case constants::MenuArray::Button::ALLOCATE_BUTTON:
            this->pollEventAllocateMode(event, mousePosView);
            break;
        case constants::MenuArray::Button::NONE:
            break;
    }
}
```

7.20.3.5 render()

```
void MenuArray::render ( )
```

Definition at line 121 of file MenuArray.cpp.

```
                         {
    for (Button* button : this->buttons) {
        button->render();
    }

    switch (this->activeOptionMenu) {
        case constants::MenuArray::Button::CREATE_BUTTON:
            this->renderCreateMode();
```

```
00129             break;
00130         case constants::MenuArray::Button::ADD_BUTTON:
00131             this->renderAddMode();
00132             break;
00133         case constants::MenuArray::Button::DELETE_BUTTON:
00134             this->renderDeleteMode();
00135             break;
00136         case constants::MenuArray::Button::UPDATE_BUTTON:
00137             this->renderUpdateMode();
00138             break;
00139         case constants::MenuArray::Button::SEARCH_BUTTON:
00140             this->renderSearchMode();
00141             break;
00142         case constants::MenuArray::Button::ALLOCATE_BUTTON:
00143             this->renderAllocateMode();
00144             break;
00145         case constants::MenuArray::Button::NONE:
00146             break;
00147     }
00148 }
```

7.20.3.6 resetActiveOptionMenu()

```
void MenuArray::resetActiveOptionMenu ( )
```

Definition at line 48 of file MenuArray.cpp.

```
00048                                         {
00049     this->activeOptionMenu = constants::MenuArray::Button::NONE;
00050     this->activeCreateMode = constants::MenuArray::CreateMode::Button::NONE;
00051 }
```

7.20.3.7 update()

```
void MenuArray::update ( )
```

Definition at line 89 of file MenuArray.cpp.

```
00089                          {
00090     if (this->activeOptionMenu != constants::MenuArray::Button::NONE)
00091         this->buttons[this->activeOptionMenu]->setColor(constants::clickGreen);
00092 
00093     for (Button* button : this->buttons) {
00094         button->update();
00095     }
00096 
00097     switch (this->activeOptionMenu) {
00098         case constants::MenuArray::Button::CREATE_BUTTON:
00099             this->updateCreateMode();
00100             break;
00101         case constants::MenuArray::Button::ADD_BUTTON:
00102             this->updateAddMode();
00103             break;
00104         case constants::MenuArray::Button::DELETE_BUTTON:
00105             this->updateDeleteMode();
00106             break;
00107         case constants::MenuArray::Button::UPDATE_BUTTON:
00108             this->updateUpdateMode();
00109             break;
00110         case constants::MenuArray::Button::SEARCH_BUTTON:
00111             this->updateSearchMode();
00112             break;
00113         case constants::MenuArray::Button::ALLOCATE_BUTTON:
00114             this->updateAllocateMode();
00115             break;
00116         case constants::MenuArray::Button::NONE:
00117             break;
00118     }
00119 }
```

7.20.4 Member Data Documentation

7.20.4.1 addModeValue

`std::string MenuArray::addModeValue[constants::MenuArray::AddMode::TEXTBOX_COUNT]`

Definition at line 84 of file MenuArray.hpp.

7.20.4.2 allocateModeValue

`std::string MenuArray::allocateModeValue`

Definition at line 96 of file MenuArray.hpp.

7.20.4.3 createModeValue

`std::string MenuArray::createModeValue[constants::MenuArray::CreateMode::BUTTON_COUNT]`

Definition at line 80 of file MenuArray.hpp.

7.20.4.4 deleteModeValue

`std::string MenuArray::deleteModeValue`

Definition at line 87 of file MenuArray.hpp.

7.20.4.5 searchModeValue

`std::string MenuArray::searchModeValue`

Definition at line 93 of file MenuArray.hpp.

7.20.4.6 updateModeValue

```
std::string MenuArray::updateModeValue[constants::MenuArray::UpdateMode::TEXTBOX_COUNT]
```

Definition at line 90 of file MenuArray.hpp.

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

- include/libScene/MenuArray.hpp
- include/libScene/MenuArray.cpp

7.21 MenuDataStructure Class Reference

```
#include <MenuDataStructure.hpp>
```

Public Member Functions

- constants::MenuDataStructure::CreateMode::Button getActiveCreateMode ()
- MenuDataStructure (sf::RenderWindow ∗window)
- ∼MenuDataStructure ()=default
- void resetActiveOptionMenu ()
- void resetActiveOptionMenuOnly ()
- void pollEvents (sf::Event event, sf::Vector2f mousePosView)
- void update ()
- void render ()
- Button ∗ getButton (int index)
- constants::MenuDataStructure::Button getActiveOptionMenu ()

Public Attributes

- std::string createModeValue [constants::MenuDataStructure::CreateMode::BUTTON_COUNT]
- std::string pushModeValue

7.21.1 Detailed Description

Definition at line 14 of file MenuDataStructure.hpp.

7.21.2 Constructor & Destructor Documentation

7.21.2.1 MenuDataStructure()

```
MenuDataStructure::MenuDataStructure (
              sf::RenderWindow * window )  [explicit]
```

Definition at line 7 of file MenuDataStructure.cpp.

```
{
    this->window = window;
    this->init();
}
```

7.21.2.2 ∼MenuDataStructure()

```
MenuDataStructure::∼MenuDataStructure ( )  [default]
```

7.21.3 Member Function Documentation

7.21.3.1 getActiveCreateMode()

```
constants::MenuDataStructure::CreateMode::Button MenuDataStructure::getActiveCreateMode ( )
```

Definition at line 94 of file MenuDataStructure.cpp.

```
{
    return this->activeCreateMode;
}
```

7.21.3.2 getActiveOptionMenu()

```
constants::MenuDataStructure::Button MenuDataStructure::getActiveOptionMenu ( )
```

Definition at line 90 of file MenuDataStructure.cpp.

```
{
    return this->activeOptionMenu;
}
```

7.21.3.3 getButton()

```
Button * MenuDataStructure::getButton (
              int index )
```

Definition at line 86 of file MenuDataStructure.cpp.

```
{
    return this->buttons[index];
}
```

7.21.3.4 pollEvents()

```
void MenuDataStructure::pollEvents (
            sf::Event event,
            sf::Vector2f mousePosView )
```

Definition at line 41 of file MenuDataStructure.cpp.

```
                                                                     {
    if (this->activeOptionMenu != constants::MenuDataStructure::Button::NONE)
        this->buttons[this->activeOptionMenu]->setColor(constants::normalGray);

    for (int i = 0; i < constants::MenuDataStructure::BUTTON_COUNT; i++) {
        if (this->buttons[i]->pollEvent(mousePosView)) {
            std::cout « "Button " « i « " is clicked" « std::endl;
            this->activeOptionMenu = static_cast<constants::MenuDataStructure::Button>(i);
        }
    }

    if (this->activeOptionMenu == constants::MenuDataStructure::Button::CREATE_BUTTON) {
        this->pollEventCreateMode(event, mousePosView);
    } else if (this->activeOptionMenu == constants::MenuDataStructure::Button::PUSH_BUTTON) {
        this->pollEventPushMode(event, mousePosView);
    }
}
```

7.21.3.5 render()

```
void MenuDataStructure::render ( )
```

Definition at line 74 of file MenuDataStructure.cpp.

```
                               {
    for (Button* button : this->buttons) {
        button->render();
    }

    if (this->activeOptionMenu == constants::MenuDataStructure::Button::CREATE_BUTTON) {
        this->renderCreateMode();
    } else if (this->activeOptionMenu == constants::MenuDataStructure::Button::PUSH_BUTTON) {
        this->renderPushMode();
    }
}
```

7.21.3.6 resetActiveOptionMenu()

```
void MenuDataStructure::resetActiveOptionMenu ( )
```

Definition at line 239 of file MenuDataStructure.cpp.

```
                                            {
    this->activeOptionMenu = constants::MenuDataStructure::Button::NONE;
    this->activeCreateMode = constants::MenuDataStructure::CreateMode::Button::NONE;
}
```

7.21.3.7 resetActiveOptionMenuOnly()

```
void MenuDataStructure::resetActiveOptionMenuOnly ( )
```

Definition at line 244 of file MenuDataStructure.cpp.

```
                                                {
    this->activeOptionMenu = constants::MenuDataStructure::Button::NONE;
}
```

7.21.3.8 update()

```
void MenuDataStructure::update ( )
```

Definition at line 59 of file MenuDataStructure.cpp.

```
                           {
    if (this->activeOptionMenu < constants::MenuDataStructure::Button::POP_BUTTON)
        this->buttons[this->activeOptionMenu]->setColor(constants::clickGreen);

    for (Button* button : this->buttons) {
        button->update();
    }

    if (this->activeOptionMenu == constants::MenuDataStructure::Button::CREATE_BUTTON) {
        this->updateCreateMode();
    } else if (this->activeOptionMenu == constants::MenuDataStructure::Button::PUSH_BUTTON) {
        this->updatePushMode();
    }
}
```

7.21.4 Member Data Documentation

7.21.4.1 createModeValue

```
std::string MenuDataStructure::createModeValue[constants::MenuDataStructure::CreateMode::BUTTON_COUNT]
```

Definition at line 45 of file MenuDataStructure.hpp.

7.21.4.2 pushModeValue

```
std::string MenuDataStructure::pushModeValue
```

Definition at line 49 of file MenuDataStructure.hpp.

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

- include/libScene/MenuDataStructure.hpp
- include/libScene/MenuDataStructure.cpp

7.22 MenuLinkedList Class Reference

```
#include <MenuLinkedList.hpp>
```

Public Member Functions

- constants::MenuLinkedList::CreateMode::Button getActiveCreateMode ()
- MenuLinkedList (sf::RenderWindow ∗window)
- ∼MenuLinkedList ()=default
- void resetActiveOptionMenu ()
- void pollEvents (sf::Event event, sf::Vector2f mousePosView)
- void update ()
- void render ()
- Button ∗ getButton (int index)
- constants::MenuLinkedList::Button getActiveOptionMenu ()

Public Attributes

- std::string createModeValue [constants::MenuLinkedList::CreateMode::BUTTON_COUNT]
- std::string addModeValue [constants::MenuLinkedList::AddMode::TEXTBOX_COUNT]
- std::string deleteModeValue
- std::string updateModeValue [constants::MenuLinkedList::UpdateMode::TEXTBOX_COUNT]
- std::string searchModeValue

Protected Member Functions

- void initCreateMode ()
- void pollEventCreateMode (sf::Event event, sf::Vector2f mousePosView)
- void updateCreateMode ()
- void renderCreateMode ()
- void initAddMode ()
- void pollEventAddMode (sf::Event event, sf::Vector2f mousePosView)
- void updateAddMode ()
- void renderAddMode ()
- void initDeleteMode ()
- void pollEventDeleteMode (sf::Event event, sf::Vector2f mousePosView)
- void updateDeleteMode ()
- void renderDeleteMode ()
- void initUpdateMode ()
- void pollEventUpdateMode (sf::Event event, sf::Vector2f mousePosView)
- void updateUpdateMode ()
- void renderUpdateMode ()
- void initSearchMode ()
- void pollEventSearchMode (sf::Event event, sf::Vector2f mousePosView)
- void updateSearchMode ()
- void renderSearchMode ()
- void init ()
- void initButtons ()

Protected Attributes

- sf::RenderWindow ∗ window
- Button ∗ buttons [constants::MenuLinkedList::BUTTON_COUNT]
- constants::MenuLinkedList::Button activeOptionMenu
- Button ∗ subCreateMode [constants::MenuLinkedList::CreateMode::BUTTON_COUNT]
- CustomTextbox ∗ createTextbox [constants::MenuLinkedList::CreateMode::BUTTON_COUNT]
- constants::MenuLinkedList::CreateMode::Button activeCreateMode
- bool isOpenFileDialog = false
- CustomTextbox ∗ addTextbox [constants::MenuLinkedList::AddMode::TEXTBOX_COUNT]
- constants::MenuLinkedList::AddMode::Textbox activeAddMode
- CustomTextbox ∗ deleteTextbox
- CustomTextbox ∗ updateTextbox [constants::MenuLinkedList::UpdateMode::TEXTBOX_COUNT]
- constants::MenuLinkedList::UpdateMode::Textbox activeUpdateMode
- CustomTextbox ∗ searchTextbox

7.22.1 Detailed Description

Definition at line 16 of file MenuLinkedList.hpp.

7.22.2 Constructor & Destructor Documentation

7.22.2.1 MenuLinkedList()

```
MenuLinkedList::MenuLinkedList (
            sf::RenderWindow * window )  [explicit]
```

Definition at line 39 of file MenuLinkedList.cpp.

```
                                                          {
    this->window = window;
    this->init();
}
```

7.22.2.2 ∼MenuLinkedList()

```
MenuLinkedList::∼MenuLinkedList ( )  [default]
```

7.22.3 Member Function Documentation

7.22.3.1 getActiveCreateMode()

`constants::MenuLinkedList::CreateMode::Button MenuLinkedList::getActiveCreateMode ( )`

Definition at line 242 of file MenuLinkedList.cpp.

```
                                                                      {
    return this->activeCreateMode;
}
```

7.22.3.2 getActiveOptionMenu()

`constants::MenuLinkedList::Button MenuLinkedList::getActiveOptionMenu ( )`

Definition at line 246 of file MenuLinkedList.cpp.

```
                                                             {
    return this->activeOptionMenu;
}
```

7.22.3.3 getButton()

```
Button * MenuLinkedList::getButton (
              int index )
```

Definition at line 132 of file MenuLinkedList.cpp.

```
                                              {
    return this->buttons[index];
}
```

7.22.3.4 init()

`void MenuLinkedList::init ( ) [protected]`

Definition at line 7 of file MenuLinkedList.cpp.

```
                          {
    this->initButtons();
    this->initCreateMode();
    this->initAddMode();
    this->initDeleteMode();
    this->initUpdateMode();
    this->initSearchMode();

    this->activeOptionMenu = constants::MenuLinkedList::Button::NONE;
}
```

7.22.3.5 initAddMode()

```
void MenuLinkedList::initAddMode ( )  [protected]
```

Definition at line 250 of file MenuLinkedList.cpp.

```
00250                              {
00251     //init stuff for add mode
00252     this->activeAddMode = constants::MenuLinkedList::AddMode::Textbox::NONE;
00253     for (int i = 0; i < constants::MenuLinkedList::AddMode::TEXTBOX_COUNT; i++) {
00254         sf::Vector2f position = sf::Vector2f(
00255                 this->buttons[1]->getPosition().x + (constants::optionButtonSize.x +
      constants::distance2ModeButtons),
00256                 this->buttons[1]->getPosition().y
00257         );
00258         this->addTextbox[i] = new CustomTextbox{
00259                 this->window,
00260                 position,
00261                 20,
00262                 constants::MenuLinkedList::AddMode::TEXTBOX_NAMES[i],
00263                 constants::MenuLinkedList::AddMode::TEXTBOX_LENGTH[i],
00264         };
00265         this->addModeValue[i] = "None";
00266     }
00267 }
```

7.22.3.6 initButtons()

```
void MenuLinkedList::initButtons ( )  [protected]
```

Definition at line 18 of file MenuLinkedList.cpp.

```
00018                              {
00019     for (int i = 0; i < constants::MenuLinkedList::BUTTON_COUNT; i++) {
00020         sf::Vector2f position = sf::Vector2f(
00021                 constants::sideButtonSize.x + constants::distance2ModeButtons,
00022                 constants::submenuButtonPos.y + (constants::optionButtonSize.y +
      constants::distance2ModeButtons / 10) * static_cast<float>(i)
00023                 );
00024         this->buttons[i] = new Button(
00025                 this->window,
00026                 position,
00027                 constants::optionButtonSize,
00028                 constants::MenuLinkedList::BUTTON_NAMES[i],
00029                 constants::MenuLinkedList::BUTTON_NAMES[i],
00030                 constants::MenuLinkedList::BUTTON_NAME_SIZE,
00031                 sf::Color::Black,
00032                 constants::normalGray,
00033                 constants::hoverGray,
00034                 constants::clickGray
00035         );
00036     }
00037 }
```

7.22.3.7 initCreateMode()

```
void MenuLinkedList::initCreateMode ( )  [protected]
```

Definition at line 141 of file MenuLinkedList.cpp.

```
00141                                 {
00142     // init stuff for create mode
00143     this->activeCreateMode = constants::MenuLinkedList::CreateMode::Button::NONE;
00144     for (int i = 0; i < constants::MenuLinkedList::CreateMode::BUTTON_COUNT; i++) {
00145         sf::Vector2f position = sf::Vector2f(
00146                 this->buttons[0]->getPosition().x + (constants::optionButtonSize.x +
      constants::distance2ModeButtons) * static_cast<float>(i + 1),
00147                 this->buttons[0]->getPosition().y
00148         );
00149         this->subCreateMode[i] = new Button(
00150                 this->window,
00151                 position,
00152                 constants::optionButtonSize,
00153                 constants::MenuLinkedList::CreateMode::BUTTON_NAMES[i],
00154                 constants::MenuLinkedList::CreateMode::BUTTON_NAMES[i],
00155                 constants::MenuLinkedList::CreateMode::NAME_SIZE,
00156                 sf::Color::Black,
00157                 constants::normalGray,
00158                 constants::hoverGray,
00159                 constants::clickGray
00160         );
00161         if (i < 2)
00162             this->createTextbox[i] = new CustomTextbox{
00163                     this->window,
00164                     sf::Vector2f(
00165                             this->subCreateMode[0]->getPosition().x,
00166                             this->subCreateMode[0]->getPosition().y + constants::optionButtonSize.y +
      constants::distance2ModeButtons
00167                     ),
00168                     20,
00169                     constants::MenuLinkedList::CreateMode::TEXTBOX_NAMES[i],
00170                     constants::MenuLinkedList::CreateMode::TEXTBOX_LENGTH[i],
00171             };
00172         this->createModeValue[i] = "None";
00173     }
00174     this->isOpenFileDialog = false;
00175 }
```

7.22.3.8 initDeleteMode()

```
void MenuLinkedList::initDeleteMode ( )  [protected]
```

Definition at line 297 of file MenuLinkedList.cpp.

```
00297                                     {
00298     sf::Vector2f position = sf::Vector2f(
00299             this->buttons[2]->getPosition().x + (constants::optionButtonSize.x +
      constants::distance2ModeButtons),
00300             this->buttons[2]->getPosition().y
00301     );
00302     this->deleteTextbox = new CustomTextbox{
00303             this->window,
00304             position,
00305             20,
00306             constants::MenuLinkedList::DeleteMode::TEXTBOX_NAME,
00307             constants::MenuLinkedList::DeleteMode::TEXTBOX_LENGTH,
00308     };
00309     this->deleteModeValue = "None";
00310 }
```

7.22.3.9 initSearchMode()

```
void MenuLinkedList::initSearchMode ( )  [protected]
```

Definition at line 380 of file MenuLinkedList.cpp.

```
00380                                     {
00381     sf::Vector2f position = sf::Vector2f(
00382             this->buttons[4]->getPosition().x + (constants::optionButtonSize.x +
      constants::distance2ModeButtons),
00383             this->buttons[4]->getPosition().y
00384     );
00385     this->searchTextbox = new CustomTextbox{
00386             this->window,
00387             position,
00388             20,
00389             constants::MenuLinkedList::SearchMode::TEXTBOX_NAME,
00390             constants::MenuLinkedList::SearchMode::TEXTBOX_LENGTH,
00391     };
00392     this->searchModeValue = "None";
00393 }
```

7.22.3.10 initUpdateMode()

```
void MenuLinkedList::initUpdateMode ( )  [protected]
```

Definition at line 333 of file MenuLinkedList.cpp.

```
                                     {
    // init stuff for update mode
    this->activeUpdateMode = constants::MenuLinkedList::UpdateMode::Textbox::NONE;
    for (int i = 0; i < constants::MenuLinkedList::UpdateMode::TEXTBOX_COUNT; i++) {
        sf::Vector2f position = sf::Vector2f(
                this->buttons[3]->getPosition().x + (constants::optionButtonSize.x +
      constants::distance2ModeButtons),
                this->buttons[3]->getPosition().y
        );
        this->updateTextbox[i] = new CustomTextbox{
                this->window,
                position,
                20,
                constants::MenuLinkedList::UpdateMode::TEXTBOX_NAMES[i],
                constants::MenuLinkedList::UpdateMode::TEXTBOX_LENGTH[i],
        };
        this->updateModeValue[i] = "None";
    }
}
```

7.22.3.11 pollEventAddMode()

```
void MenuLinkedList::pollEventAddMode (
            sf::Event event,
            sf::Vector2f mousePosView )  [protected]
```

Definition at line 268 of file MenuLinkedList.cpp.

```
                                                                             {
    if (this->activeAddMode == constants::MenuLinkedList::AddMode::NONE)
        this->activeAddMode = constants::MenuLinkedList::AddMode::POSITION_TEXTBOX;

    this->addTextbox[this->activeAddMode]->pollEvent(event, mousePosView);
}
```

7.22.3.12 pollEventCreateMode()

```
void MenuLinkedList::pollEventCreateMode (
            sf::Event event,
            sf::Vector2f mousePosView )  [protected]
```

Definition at line 176 of file MenuLinkedList.cpp.

```
                                                                                {
    if (this->activeCreateMode != constants::MenuLinkedList::CreateMode::Button::NONE)
        this->subCreateMode[this->activeCreateMode]->setColor(constants::normalGray);

    for (int i = 0; i < constants::MenuLinkedList::CreateMode::BUTTON_COUNT; i++) {
        if (this->subCreateMode[i]->pollEvent(mousePosView)) {
            this->activeCreateMode = static_cast<constants::MenuLinkedList::CreateMode::Button>(i);
            if (i == constants::MenuLinkedList::CreateMode::Button::FILE_BUTTON)
                this->isOpenFileDialog = true;
            std::cout « "Button " « i « " is clicked" « std::endl;
        }
    }

//    this->testTextbox->pollEvent(event);
    if (this->activeCreateMode < constants::MenuLinkedList::CreateMode::TEXTBOX_COUNT)
        this->createTextbox[this->activeCreateMode]->pollEvent(event, mousePosView);
}
```

7.22.3.13 pollEventDeleteMode()

```
void MenuLinkedList::pollEventDeleteMode (
            sf::Event event,
            sf::Vector2f mousePosView )  [protected]
```

Definition at line 311 of file MenuLinkedList.cpp.

```
                                                                                {
    this->deleteTextbox->pollEvent(event, mousePosView);
}
```

7.22.3.14 pollEvents()

```
void MenuLinkedList::pollEvents (
            sf::Event event,
            sf::Vector2f mousePosView )
```

Definition at line 44 of file MenuLinkedList.cpp.

```
                                                                        {
    if (this->activeOptionMenu != constants::MenuLinkedList::Button::NONE)
        this->buttons[this->activeOptionMenu]->setColor(constants::normalGray);

    for (int i = 0; i < constants::MenuLinkedList::BUTTON_COUNT; i++) {
        if (this->buttons[i]->pollEvent(mousePosView)) {
            std::cout « "Button " « i « " is clicked" « std::endl;
            this->activeOptionMenu = static_cast<constants::MenuLinkedList::Button>(i);
            this->activeAddMode = constants::MenuLinkedList::AddMode::Textbox::NONE;
        }
    }

    switch (this->activeOptionMenu) {
        case constants::MenuLinkedList::Button::CREATE_BUTTON:
            this->pollEventCreateMode(event, mousePosView);
            break;
        case constants::MenuLinkedList::Button::ADD_BUTTON:
            this->pollEventAddMode(event, mousePosView);
            break;
        case constants::MenuLinkedList::Button::DELETE_BUTTON:
            this->pollEventDeleteMode(event, mousePosView);
            break;
        case constants::MenuLinkedList::Button::UPDATE_BUTTON:
            this->pollEventUpdateMode(event, mousePosView);
            break;
        case constants::MenuLinkedList::Button::SEARCH_BUTTON:
            this->pollEventSearchMode(event, mousePosView);
            break;
        case constants::MenuLinkedList::Button::NONE:
            break;
    }
}
```

7.22.3.15 pollEventSearchMode()

```
void MenuLinkedList::pollEventSearchMode (
            sf::Event event,
            sf::Vector2f mousePosView )  [protected]
```

Definition at line 394 of file MenuLinkedList.cpp.

```
                                                                                {
    this->searchTextbox->pollEvent(event, mousePosView);
}
```

7.22.3.16 pollEventUpdateMode()

```
void MenuLinkedList::pollEventUpdateMode (
            sf::Event event,
            sf::Vector2f mousePosView )  [protected]
```

Definition at line 351 of file MenuLinkedList.cpp.

```
                                                                    {
    if (this->activeUpdateMode == constants::MenuLinkedList::UpdateMode::NONE)
        this->activeUpdateMode = constants::MenuLinkedList::UpdateMode::POSITION_TEXTBOX;

    this->updateTextbox[this->activeUpdateMode]->pollEvent(event, mousePosView);
}
```

7.22.3.17 render()

```
void MenuLinkedList::render ( )
```

Definition at line 106 of file MenuLinkedList.cpp.

```
                            {
    for (Button* button : this->buttons) {
        button->render();
    }

    switch (this->activeOptionMenu) {
        case constants::MenuLinkedList::Button::CREATE_BUTTON:
            this->renderCreateMode();
            break;
        case constants::MenuLinkedList::Button::ADD_BUTTON:
            this->renderAddMode();
            break;
        case constants::MenuLinkedList::Button::DELETE_BUTTON:
            this->renderDeleteMode();
            break;
        case constants::MenuLinkedList::Button::UPDATE_BUTTON:
            this->renderUpdateMode();
            break;
        case constants::MenuLinkedList::Button::SEARCH_BUTTON:
            this->renderSearchMode();
            break;
        case constants::MenuLinkedList::Button::NONE:
            break;
    }
}
```

7.22.3.18 renderAddMode()

```
void MenuLinkedList::renderAddMode ( )  [protected]
```

Definition at line 293 of file MenuLinkedList.cpp.

```
                                   {
    this->addTextbox[this->activeAddMode]->render();
}
```

7.22.3.19 renderCreateMode()

```
void MenuLinkedList::renderCreateMode ( )  [protected]
```

Definition at line 232 of file MenuLinkedList.cpp.

```
                                    {
    for (Button* button : this->subCreateMode) {
        button->render();
    }

//    this->testTextbox->render();
    if (this->activeCreateMode < constants::MenuLinkedList::CreateMode::TEXTBOX_COUNT)
        this->createTextbox[this->activeCreateMode]->render();
}
```

7.22.3.20 renderDeleteMode()

```
void MenuLinkedList::renderDeleteMode ( )  [protected]
```

Definition at line 329 of file MenuLinkedList.cpp.

```
                                    {
    this->deleteTextbox->render();
}
```

7.22.3.21 renderSearchMode()

```
void MenuLinkedList::renderSearchMode ( )  [protected]
```

Definition at line 412 of file MenuLinkedList.cpp.

```
                                    {
    this->searchTextbox->render();
}
```

7.22.3.22 renderUpdateMode()

```
void MenuLinkedList::renderUpdateMode ( )  [protected]
```

Definition at line 376 of file MenuLinkedList.cpp.

```
                                    {
    this->updateTextbox[this->activeUpdateMode]->render();
}
```

7.22.3.23 resetActiveOptionMenu()

```
void MenuLinkedList::resetActiveOptionMenu ( )
```

Definition at line 136 of file MenuLinkedList.cpp.

```
                                        {
    this->activeOptionMenu = constants::MenuLinkedList::Button::NONE;
    this->activeCreateMode = constants::MenuLinkedList::CreateMode::Button::NONE;
}
```

7.22.3.24 update()

```
void MenuLinkedList::update ( )
```

Definition at line 77 of file MenuLinkedList.cpp.

```
                              {
    if (this->activeOptionMenu != constants::MenuLinkedList::Button::NONE)
        this->buttons[this->activeOptionMenu]->setColor(constants::clickGreen);

    for (Button* button : this->buttons) {
        button->update();
    }

    switch (this->activeOptionMenu) {
        case constants::MenuLinkedList::Button::CREATE_BUTTON:
            this->updateCreateMode();
            break;
        case constants::MenuLinkedList::Button::ADD_BUTTON:
            this->updateAddMode();
            break;
        case constants::MenuLinkedList::Button::DELETE_BUTTON:
            this->updateDeleteMode();
            break;
        case constants::MenuLinkedList::Button::UPDATE_BUTTON:
            this->updateUpdateMode();
            break;
        case constants::MenuLinkedList::Button::SEARCH_BUTTON:
            this->updateSearchMode();
            break;
        case constants::MenuLinkedList::Button::NONE:
            break;
    }
}
```

7.22.3.25 updateAddMode()

```
void MenuLinkedList::updateAddMode ( ) [protected]
```

Definition at line 274 of file MenuLinkedList.cpp.

```
                                     {
    if (this->activeAddMode == constants::MenuLinkedList::AddMode::NONE)
        this->activeAddMode = constants::MenuLinkedList::AddMode::POSITION_TEXTBOX;

    this->addTextbox[this->activeAddMode]->update();

    std::string inputUser = this->addTextbox[this->activeAddMode]->getTextString();
    // check if input is number
    bool isValid = true;
    for (char i : inputUser)
        if (!std::isdigit(i))
            isValid = false;
    if (isValid && inputUser != "None") {
        this->addModeValue[this->activeAddMode] = inputUser;
        std::cout « inputUser « std::endl;
        this->addTextbox[this->activeAddMode]->resetInput();
        this->activeAddMode =
static_cast<constants::MenuLinkedList::AddMode::Textbox>(!this->activeAddMode);
    }
}
```

7.22.3.26 updateCreateMode()

```
void MenuLinkedList::updateCreateMode ( ) [protected]
```

Definition at line 193 of file MenuLinkedList.cpp.

```
                                        {
    if (this->activeCreateMode != constants::MenuLinkedList::CreateMode::Button::NONE)
```

```
        this->subCreateMode[this->activeCreateMode]->setColor(constants::clickGreen);

    for (Button* button : this->subCreateMode) {
        button->update();
    }

//    this->testTextbox->update();
    if (this->activeCreateMode < constants::MenuLinkedList::CreateMode::TEXTBOX_COUNT) {
        this->createTextbox[this->activeCreateMode]->update();
        std::string inputUser = this->createTextbox[this->activeCreateMode]->getTextString();
        if (inputUser != "None") {
            std::cout « inputUser « std::endl;
            this->createTextbox[this->activeCreateMode]->resetInput();
        }
        this->createModeValue[this->activeCreateMode] = inputUser;
    } else if (this->activeCreateMode == constants::MenuLinkedList::CreateMode::FILE_BUTTON) {
        if (this->isOpenFileDialog) {
            auto f = pfd::open_file("Choose files to read", pfd::path::home(),
                                    {"Text Files (.txt .text)", "*.txt *.text",
                                     "All Files", "*"});

            // wait for the user to select a file unless the window will be not responsive
            while (!f.ready(100)) {
                sf::Event event{};
                this->window->pollEvent(event);
            }

            if (!f.result().empty()) {
                std::ifstream file(f.result()[0]);
                std::string line;
                file » line;
                this->createModeValue[this->activeCreateMode] = line;
            }
        }
        this->isOpenFileDialog = false;
    }
}
```

7.22.3.27 updateDeleteMode()

`void MenuLinkedList::updateDeleteMode ( )  [protected]`

Definition at line 314 of file MenuLinkedList.cpp.

```
                                        {
    this->deleteTextbox->update();

    std::string inputUser = this->deleteTextbox->getTextString();
    // check if input is number
    bool isValid = true;
    for (char i : inputUser)
        if (!std::isdigit(i))
            isValid = false;
    if (isValid && inputUser != "None") {
        this->deleteModeValue = inputUser;
        std::cout « inputUser « std::endl;
        this->deleteTextbox->resetInput();
    }
}
```

7.22.3.28 updateSearchMode()

`void MenuLinkedList::updateSearchMode ( )  [protected]`

Definition at line 397 of file MenuLinkedList.cpp.

```
                                        {
    this->searchTextbox->update();

    std::string inputUser = this->searchTextbox->getTextString();
    // check if input is number
    bool isValid = true;
    for (char i : inputUser)
        if (!std::isdigit(i))
            isValid = false;
    if (isValid && inputUser != "None") {
        this->searchModeValue = inputUser;
        std::cout « inputUser « std::endl;
        this->searchTextbox->resetInput();
    }
}
```

7.22.3.29 updateUpdateMode()

`void MenuLinkedList::updateUpdateMode ( ) [protected]`

Definition at line 357 of file MenuLinkedList.cpp.

```
                                             {
    if (this->activeUpdateMode == constants::MenuLinkedList::UpdateMode::NONE)
        this->activeUpdateMode = constants::MenuLinkedList::UpdateMode::POSITION_TEXTBOX;

    this->updateTextbox[this->activeUpdateMode]->update();

    std::string inputUser = this->updateTextbox[this->activeUpdateMode]->getTextString();
    // check if input is number
    bool isValid = true;
    for (char i : inputUser)
        if (!std::isdigit(i))
            isValid = false;
    if (isValid && inputUser != "None") {
        this->updateModeValue[this->activeUpdateMode] = inputUser;
        std::cout « inputUser « std::endl;
        this->updateTextbox[this->activeUpdateMode]->resetInput();
        this->activeUpdateMode =
      static_cast<constants::MenuLinkedList::UpdateMode::Textbox>(!this->activeUpdateMode);
    }
}
```

7.22.4 Member Data Documentation

7.22.4.1 activeAddMode

`constants::MenuLinkedList::AddMode::Textbox MenuLinkedList::activeAddMode [protected]`

Definition at line 36 of file MenuLinkedList.hpp.

7.22.4.2 activeCreateMode

`constants::MenuLinkedList::CreateMode::Button MenuLinkedList::activeCreateMode [protected]`

Definition at line 26 of file MenuLinkedList.hpp.

7.22.4.3 activeOptionMenu

`constants::MenuLinkedList::Button MenuLinkedList::activeOptionMenu [protected]`

Definition at line 21 of file MenuLinkedList.hpp.

7.22.4.4 activeUpdateMode

`constants::MenuLinkedList::UpdateMode::Textbox MenuLinkedList::activeUpdateMode [protected]`

Definition at line 53 of file MenuLinkedList.hpp.

7.22.4.5 addModeValue

`std::string MenuLinkedList::addModeValue[constants::MenuLinkedList::AddMode::TEXTBOX_COUNT]`

Definition at line 77 of file MenuLinkedList.hpp.

7.22.4.6 addTextbox

`CustomTextbox* MenuLinkedList::addTextbox[constants::MenuLinkedList::AddMode::TEXTBOX_COUNT] [protected]`

Definition at line 35 of file MenuLinkedList.hpp.

7.22.4.7 buttons

`Button* MenuLinkedList::buttons[constants::MenuLinkedList::BUTTON_COUNT] [protected]`

Definition at line 19 of file MenuLinkedList.hpp.

7.22.4.8 createModeValue

`std::string MenuLinkedList::createModeValue[constants::MenuLinkedList::CreateMode::BUTTON_COUNT]`

Definition at line 73 of file MenuLinkedList.hpp.

7.22.4.9 createTextbox

```
CustomTextbox* MenuLinkedList::createTextbox[constants::MenuLinkedList::CreateMode::BUTTON_COUNT]
[protected]
```

Definition at line 25 of file MenuLinkedList.hpp.

7.22.4.10 deleteModeValue

```
std::string MenuLinkedList::deleteModeValue
```

Definition at line 80 of file MenuLinkedList.hpp.

7.22.4.11 deleteTextbox

```
CustomTextbox* MenuLinkedList::deleteTextbox  [protected]
```

Definition at line 44 of file MenuLinkedList.hpp.

7.22.4.12 isOpenFileDialog

```
bool MenuLinkedList::isOpenFileDialog = false  [protected]
```

Definition at line 27 of file MenuLinkedList.hpp.

7.22.4.13 searchModeValue

```
std::string MenuLinkedList::searchModeValue
```

Definition at line 86 of file MenuLinkedList.hpp.

7.22.4.14 searchTextbox

```
CustomTextbox* MenuLinkedList::searchTextbox  [protected]
```

Definition at line 61 of file MenuLinkedList.hpp.

7.22.4.15 subCreateMode

```
Button* MenuLinkedList::subCreateMode[constants::MenuLinkedList::CreateMode::BUTTON_COUNT]
[protected]
```

Definition at line 24 of file MenuLinkedList.hpp.

7.22.4.16 updateModeValue

```
std::string MenuLinkedList::updateModeValue[constants::MenuLinkedList::UpdateMode::TEXTBOX_COUNT]
```

Definition at line 83 of file MenuLinkedList.hpp.

7.22.4.17 updateTextbox

```
CustomTextbox* MenuLinkedList::updateTextbox[constants::MenuLinkedList::UpdateMode::TEXTBOX_COUNT]
[protected]
```

Definition at line 52 of file MenuLinkedList.hpp.

7.22.4.18 window

```
sf::RenderWindow* MenuLinkedList::window  [protected]
```

Definition at line 18 of file MenuLinkedList.hpp.

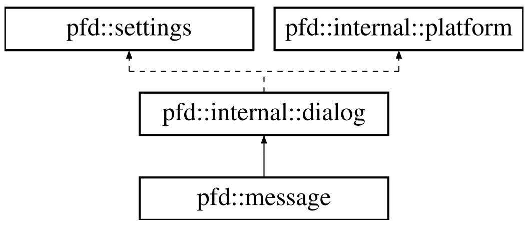
The documentation for this class was generated from the following files:

- include/libScene/MenuLinkedList.hpp
- include/libScene/MenuLinkedList.cpp

7.23 pfd::message Class Reference

```
#include <FileDialog.h>
```

Inheritance diagram for pfd::message:



Public Member Functions

- message (std::string const &title, std::string const &text, choice _choice=choice::ok_cancel, icon _↩ icon=icon::info)
- button result ()

Public Member Functions inherited from pfd::internal::dialog

- bool ready (int timeout=default_wait_timeout) const
- bool kill () const

Additional Inherited Members

Protected Types inherited from pfd::settings

- enum class flag {
  is_scanned = 0 , is_verbose , has_zenity , has_matedialog ,
  has_qarma , has_kdialog , is_vista , max_flag }

Protected Member Functions inherited from pfd::internal::dialog

- dialog ()
- std::vector< std::string > desktop_helper () const
- std::string powershell_quote (std::string const &str) const
- std::string osascript_quote (std::string const &str) const
- std::string shell_quote (std::string const &str) const

Protected Member Functions inherited from pfd::settings

- settings (bool resync=false)
- bool check_program (std::string const &program)
- bool is_osascript () const
- bool is_zenity () const
- bool is_kdialog () const
- bool const & flags (flag in_flag) const
- bool & flags (flag in_flag)

Static Protected Member Functions inherited from pfd::internal::dialog

- static std::string buttons_to_name (choice _choice)
- static std::string get_icon_name (icon _icon)

Static Protected Member Functions inherited from pfd::settings

- static bool available ()
- static void verbose (bool value)
- static void rescan ()

Protected Attributes inherited from pfd::internal::dialog

- std::shared_ptr< executor > m_async

7.23.1 Detailed Description

Definition at line 348 of file FileDialog.h.

7.23.2 Constructor & Destructor Documentation

7.23.2.1 message()

```
pfd::message::message (
              std::string const & title,
              std::string const & text,
              choice _choice = choice::ok_cancel,
              icon _icon = icon::info )  [inline]
```

Definition at line 1596 of file FileDialog.h.

```
    {
#if _WIN32
        // Use MB_SYSTEMMODAL rather than MB_TOPMOST to ensure the message window is brought
    // to front. See https://github.com/samhocevar/portable-file-dialogs/issues/52
    UINT style = MB_SYSTEMMODAL;
    switch (_icon)
    {
        case icon::warning: style |= MB_ICONWARNING; break;
        case icon::error: style |= MB_ICONERROR; break;
        case icon::question: style |= MB_ICONQUESTION; break;
        /* case icon::info: */ default: style |= MB_ICONINFORMATION; break;
    }

    switch (_choice)
    {
        case choice::ok_cancel: style |= MB_OKCANCEL; break;
        case choice::yes_no: style |= MB_YESNO; break;
        case choice::yes_no_cancel: style |= MB_YESNOCANCEL; break;
        case choice::retry_cancel: style |= MB_RETRYCANCEL; break;
        case choice::abort_retry_ignore: style |= MB_ABORTRETRYIGNORE; break;
        /* case choice::ok: */ default: style |= MB_OK; break;
    }

    m_mappings[IDCANCEL] = button::cancel;
    m_mappings[IDOK] = button::ok;
    m_mappings[IDYES] = button::yes;
    m_mappings[IDNO] = button::no;
    m_mappings[IDABORT] = button::abort;
    m_mappings[IDRETRY] = button::retry;
    m_mappings[IDIGNORE] = button::ignore;

    m_async->start_func([text, title, style](int* exit_code) -> std::string
    {
        auto wtext = internal::str2wstr(text);
        auto wtitle = internal::str2wstr(title);
        // Apply new visual style (required for all Windows versions)
        new_style_context ctx;
        *exit_code = MessageBoxW(GetActiveWindow(), wtext.c_str(), wtitle.c_str(), style);
        return "";
    });

#elif __EMSCRIPTEN__
        std::string full_message;
    switch (_icon)
    {
        case icon::warning: full_message = ""; break;
        case icon::error: full_message = ""; break;
        case icon::question: full_message = ""; break;
        /* case icon::info: */ default: full_message = ""; break;
    }

    full_message += ' ' + title + "\n\n" + text;

    // This does not really start an async task; it just passes the
    // EM_ASM_INT return value to a fake start() function.
    m_async->start(EM_ASM_INT(
    {
        if ($1)
            return window.confirm(UTF8ToString($0)) ? 0 : -1;
        alert(UTF8ToString($0));
        return 0;
    }, full_message.c_str(), _choice == choice::ok_cancel));
#else
        auto command = desktop_helper();

        if (is_osascript())
        {
            std::string script = "display dialog " + osascript_quote(text) +
                                 " with title " + osascript_quote(title);
            auto if_cancel = button::cancel;
            switch (_choice)
            {
                case choice::ok_cancel:
                    script += "buttons {\"OK\", \"Cancel\"}"
                              " default button \"OK\""
                              " cancel button \"Cancel\"";
                    break;
                case choice::yes_no:
                    script += "buttons {\"Yes\", \"No\"}"
                              " default button \"Yes\""
                              " cancel button \"No\"";
                    if_cancel = button::no;
                    break;
                case choice::yes_no_cancel:
                    script += "buttons {\"Yes\", \"No\", \"Cancel\"}"
                              " default button \"Yes\""
                              " cancel button \"Cancel\"";
                    break;
                case choice::retry_cancel:
                    script += "buttons {\"Retry\", \"Cancel\"}"
                              " default button \"Retry\""
                              " cancel button \"Cancel\"";
                    break;
                case choice::abort_retry_ignore:
                    script += "buttons {\"Abort\", \"Retry\", \"Ignore\"}"
                              " default button \"Abort\""
                              " cancel button \"Retry\"";
                    if_cancel = button::retry;
                    break;
                case choice::ok: default:
                    script += "buttons {\"OK\"}"
                              " default button \"OK\""
                              " cancel button \"OK\"";
                    if_cancel = button::ok;
                    break;
            }
            m_mappings[1] = if_cancel;
            m_mappings[256] = if_cancel; // XXX: I think this was never correct
            script += " with icon ";
            switch (_icon)
            {
#define PFD_OSX_ICON(n) "alias ((path to library folder from system domain) as text " \
                "& \"CoreServices:CoreTypes.bundle:Contents:Resources:" n ".icns\")"
                case icon::info: default: script += PFD_OSX_ICON("ToolBarInfo"); break;
                case icon::warning: script += "caution"; break;
                case icon::error: script += "stop"; break;
                case icon::question: script += PFD_OSX_ICON("GenericQuestionMarkIcon"); break;
#undef PFD_OSX_ICON
            }

            command.push_back("-e");
            command.push_back(script);
        }
        else if (is_zenity())
        {
            switch (_choice)
            {
                case choice::ok_cancel:
                    command.insert(command.end(), { "--question", "--cancel-label=Cancel",
"--ok-label=OK" }); break;
                case choice::yes_no:
                    // Do not use standard --question because it causes No to return -1,
                    // which is inconsistent with the Yes/No/Cancel mode below.
                    command.insert(command.end(), { "--question", "--switch", "--extra-button=No",
"--extra-button=Yes" }); break;
```

```
                case choice::yes_no_cancel:
                    command.insert(command.end(), { "--question", "--switch", "--extra-button=Cancel",
"--extra-button=No", "--extra-button=Yes" }); break;
                case choice::retry_cancel:
                    command.insert(command.end(), { "--question", "--switch", "--extra-button=Cancel",
"--extra-button=Retry" }); break;
                case choice::abort_retry_ignore:
                    command.insert(command.end(), { "--question", "--switch", "--extra-button=Ignore",
"--extra-button=Abort", "--extra-button=Retry" }); break;
                case choice::ok:
                default:
                    switch (_icon)
                    {
                        case icon::error: command.push_back("--error"); break;
                        case icon::warning: command.push_back("--warning"); break;
                        default: command.push_back("--info"); break;
                    }
            }

            command.insert(command.end(), { "--title", title,
                                            "--width=300", "--height=0", // sensible defaults
                                            "--no-markup", // do not interpret text as Pango markup
                                            "--text", text,
                                            "--icon-name=dialog-" + get_icon_name(_icon) });
        }
        else if (is_kdialog())
        {
            if (_choice == choice::ok)
            {
                switch (_icon)
                {
                    case icon::error: command.push_back("--error"); break;
                    case icon::warning: command.push_back("--sorry"); break;
                    default: command.push_back("--msgbox"); break;
                }
            }
            else
            {
                std::string flag = "--";
                if (_icon == icon::warning || _icon == icon::error)
                    flag += "warning";
                flag += "yesno";
                if (_choice == choice::yes_no_cancel)
                    flag += "cancel";
                command.push_back(flag);
                if (_choice == choice::yes_no || _choice == choice::yes_no_cancel)
                {
                    m_mappings[0] = button::yes;
                    m_mappings[256] = button::no;
                }
            }

            command.push_back(text);
            command.push_back("--title");
            command.push_back(title);

            // Must be after the above part
            if (_choice == choice::ok_cancel)
                command.insert(command.end(), { "--yes-label", "OK", "--no-label", "Cancel" });
        }

        if (flags(flag::is_verbose))
            std::cerr « "pfd: " « command « std::endl;

        m_async->start_process(command);
#endif
    }
```

7.23.3 Member Function Documentation

7.23.3.1 result()

```
button pfd::message::result ( )  [inline]
```

Definition at line 1798 of file FileDialog.h.

```
    {
        int exit_code;
        auto ret = m_async->result(&exit_code);
        // osascript will say "button returned:Cancel\n"
        // and others will just say "Cancel\n"
        if (internal::ends_with(ret, "Cancel\n"))
            return button::cancel;
        if (internal::ends_with(ret, "OK\n"))
            return button::ok;
        if (internal::ends_with(ret, "Yes\n"))
            return button::yes;
        if (internal::ends_with(ret, "No\n"))
            return button::no;
        if (internal::ends_with(ret, "Abort\n"))
            return button::abort;
        if (internal::ends_with(ret, "Retry\n"))
            return button::retry;
        if (internal::ends_with(ret, "Ignore\n"))
            return button::ignore;
        if (m_mappings.count(exit_code) != 0)
            return m_mappings[exit_code];
        return exit_code == 0 ? button::ok : button::cancel;
    }
```

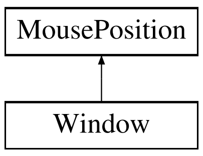
The documentation for this class was generated from the following file:

- include/core/FileDialog.h

7.24 MousePosition Class Reference

`#include <MousePosition.hpp>`

Inheritance diagram for MousePosition:



Public Member Functions

- void updateMousePosition ()

Protected Attributes

- sf::RenderWindow ∗ relativeWindow
- sf::Vector2i mousePos
- sf::Vector2f mousePosView

7.24.1 Detailed Description

Definition at line 10 of file MousePosition.hpp.

7.24.2 Member Function Documentation

7.24.2.1 updateMousePosition()

```
void MousePosition::updateMousePosition ( )
```

Definition at line 7 of file MousePosition.cpp.

```
                                          {
    this->mousePos = sf::Mouse::getPosition(*this->relativeWindow);
    this->mousePosView = this->relativeWindow->mapPixelToCoords(this->mousePos);
}
```

7.24.3 Member Data Documentation

7.24.3.1 mousePos

```
sf::Vector2i MousePosition::mousePos [protected]
```

Definition at line 14 of file MousePosition.hpp.

7.24.3.2 mousePosView

```
sf::Vector2f MousePosition::mousePosView [protected]
```

Definition at line 15 of file MousePosition.hpp.

7.24.3.3 relativeWindow

```
sf::RenderWindow* MousePosition::relativeWindow [protected]
```

Definition at line 12 of file MousePosition.hpp.

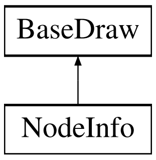
The documentation for this class was generated from the following files:

- include/MousePosition.hpp
- include/MousePosition.cpp

7.25 NodeInfo Class Reference

```
#include <NodeInfo.hpp>
```

Inheritance diagram for NodeInfo:



Public Types

- enum class ArrowType { LEFT , RIGHT }
- enum class StatusNode { InChain , OutChain , Visible }
- enum class TypeNode { Normal , Outside , Effective }

Public Member Functions

- NodeInfo (sf::RenderWindow ∗window, std::string value, sf::Vector2f position, bool _isDLL)
- ∼NodeInfo ()
- void updateNode ()
- void updateArrows (ArrowType type, sf::Vector2f end)
- void render () override
- void initArrow (ArrowType type, sf::Vector2f start, sf::Vector2f end)
- void destroyArrow (ArrowType type)
- void reInitPos (int index)
- void reInitPreVal ()
- void setEffectivePosition (sf::Vector2f start)
- void setArrows (ArrowType type, sf::Vector2f start, sf::Vector2f end)
- void setValue (std::string value)
- sf::Vector2f getPosition ()
- std::string getValue ()
- void toggleActiveColorNode ()
- void toggleActiveColorArrow (ArrowType type)
- void setPrintPreVal ()
- void setPrintNormal ()
- void setNodeInChain ()
- void setNodeOutside ()
- void setNodeVisible ()
- void setTitle (const std::string &title)
- void hide (ArrowType type)
- void show (ArrowType type)
- StatusNode getStatusNode ()
- void resetColorNode ()
- void resetColorArrow (ArrowType type)
- void resetTitle ()
- void reset ()

Public Member Functions inherited from BaseDraw

- BaseDraw (sf::RenderWindow ∗window)
- virtual void render ()=0

Additional Inherited Members

Protected Attributes inherited from BaseDraw

- sf::RenderWindow ∗ window

7.25.1 Detailed Description

Definition at line 12 of file NodeInfo.hpp.

7.25.2 Member Enumeration Documentation

7.25.2.1 ArrowType

`enum class NodeInfo::ArrowType [strong]`

Enumerator

LEFT	
RIGHT	

Definition at line 14 of file NodeInfo.hpp.

```
                        {
        LEFT,
        RIGHT
    };
```

7.25.2.2 StatusNode

`enum class NodeInfo::StatusNode [strong]`

Enumerator

InChain	
OutChain	
Visible	

Definition at line 19 of file NodeInfo.hpp.

```
                        {
        InChain,
        OutChain,
        Visible
    };
```

7.25.2.3 TypeNode

enum class NodeInfo::TypeNode [strong]

Enumerator

Normal	
Outside	
Effective	

Definition at line 25 of file NodeInfo.hpp.

```
                        {
        Normal,
        Outside,
        Effective
    };
```

7.25.3 Constructor & Destructor Documentation

7.25.3.1 NodeInfo()

```
NodeInfo::NodeInfo (
            sf::RenderWindow * window,
            std::string value,
            sf::Vector2f position,
            bool _isDLL )
```

Definition at line 7 of file NodeInfo.cpp.

```
                                                                                        :
      BaseDraw(window) {
    this->values[0] = value;
    this->values[1] = value;

    this->positions[(int)TypeNode::Normal] = position;
    this->positions[(int)TypeNode::Effective] = position;
    this->positions[(int)TypeNode::Outside] = sf::Vector2f(
            position.x,
            position.y + constants::NodeInfo::offsetY
            );

    this->isDLL = _isDLL;

    this->statusNode = StatusNode::InChain;

    this->node = new SingleNode(window, std::move(value), this->positions[(int)TypeNode::Normal]);

    for (auto &arrow : this->arrows)
        arrow[(int)ArrowType::LEFT] = arrow[(int)ArrowType::RIGHT] = nullptr;

    this->isPrintPreVal = this->isPrintNormal = false;

    this->title.setFont(this->node->font);
    this->title.setCharacterSize(constants::TitleNode::fontSize);
    this->title.setFillColor(constants::titleGreen);
    this->title.setString("");
}
```

7.25.3.2 ∼NodeInfo()

```
NodeInfo::∼NodeInfo ( )
```

Definition at line 193 of file NodeInfo.cpp.

```
                        {
    delete this->node;
    for (auto & arrow : this->arrows) {
        for (auto & j : arrow) {
            delete j;
        }
    }
}
```

7.25.4 Member Function Documentation

7.25.4.1 destroyArrow()

```
void NodeInfo::destroyArrow (
             NodeInfo::ArrowType type )
```

Definition at line 231 of file NodeInfo.cpp.

```
                                                     {
    if (this->arrows[0][(int)type])
        delete this->arrows[0][(int)type];
    if (this->arrows[1][(int)type])
        delete this->arrows[1][(int)type];
    this->arrows[0][(int)type] = nullptr;
    this->arrows[1][(int)type] = nullptr;
}
```

7.25.4.2 getPosition()

```
sf::Vector2f NodeInfo::getPosition ( )
```

Definition at line 94 of file NodeInfo.cpp.

```
                                   {
    this->updateNode(); // ?
    return this->node->getPosition();
}
```

7.25.4.3 getStatusNode()

```
NodeInfo::StatusNode NodeInfo::getStatusNode ( )
```

Definition at line 164 of file NodeInfo.cpp.

```
                                            {
    return this->statusNode;
}
```

7.25.4.4 getValue()

```
std::string NodeInfo::getValue ( )
```

Definition at line 206 of file NodeInfo.cpp.

```
                              {
    return this->values[0];
}
```

7.25.4.5 hide()

```
void NodeInfo::hide (
            NodeInfo::ArrowType type )
```

Definition at line 179 of file NodeInfo.cpp.

```
                                              {
    if (this->arrows[0][(int)type])
        this->arrows[0][(int)type]->hide();
    if (this->arrows[1][(int)type])
        this->arrows[1][(int)type]->hide();
}
```

7.25.4.6 initArrow()

```
void NodeInfo::initArrow (
            NodeInfo::ArrowType type,
            sf::Vector2f start,
            sf::Vector2f end )
```

Definition at line 54 of file NodeInfo.cpp.

```
                                                                                {
    this->arrows[1][(int)type] = new Arrow(this->window, start, end);
    this->arrows[1][(int)type]->setMid();
    this->arrows[0][(int)type] = new Arrow(this->window, start, end);
}
```

7.25.4.7 reInitPos()

```
void NodeInfo::reInitPos (
            int index )
```

Definition at line 99 of file NodeInfo.cpp.

```
                                  {
    this->positions[(int)TypeNode::Normal] = sf::Vector2f(
            constants::NodeInfo::originNode.x + static_cast<float>(index) *
 constants::NodeInfo::offsetX,
            constants::NodeInfo::originNode.y
    );
    this->positions[(int)TypeNode::Outside] = sf::Vector2f(
            this->positions[(int)TypeNode::Effective].x,
            this->positions[(int)TypeNode::Effective].y + constants::NodeInfo::offsetY
            );
}
```

7.25.4.8 reInitPreVal()

```
void NodeInfo::reInitPreVal ( )
```

Definition at line 160 of file NodeInfo.cpp.

```
                          {
    this->values[1] = this->values[0];
}
```

7.25.4.9 render()

```
void NodeInfo::render ( )  [override], [virtual]
```

Implements BaseDraw.

Definition at line 35 of file NodeInfo.cpp.

```
                    {
    if (this->statusNode == StatusNode::Visible)
        return;

    if (this->isDLL && this->statusNode == StatusNode::InChain){
        if (this->arrows[1][(int)ArrowType::LEFT])
            this->arrows[1][(int)ArrowType::LEFT]->render();
        if (this->arrows[1][(int)ArrowType::RIGHT])
            this->arrows[1][(int)ArrowType::RIGHT]->render();
    } else {
        if (this->arrows[0][(int)ArrowType::LEFT])
            this->arrows[0][(int)ArrowType::LEFT]->render();
        if (this->arrows[0][(int)ArrowType::RIGHT])
            this->arrows[0][(int)ArrowType::RIGHT]->render();
    }
    this->node->render();
    this->window->draw(this->title);
}
```

7.25.4.10 reset()

```
void NodeInfo::reset ( )
```

Definition at line 82 of file NodeInfo.cpp.

```
                   {
    this->resetColorNode();
    this->resetColorArrow(ArrowType::LEFT);
    this->resetColorArrow(ArrowType::RIGHT);
    this->resetTitle();
    this->isPrintNormal = this->isPrintPreVal = false;
    this->statusNode = StatusNode::InChain;
    this->show(ArrowType::LEFT);
    this->show(ArrowType::RIGHT);
}
```

7.25.4.11 resetColorArrow()

```
void NodeInfo::resetColorArrow (
            NodeInfo::ArrowType type )
```

Definition at line 75 of file NodeInfo.cpp.

```
                                                          {
    if (this->arrows[0][(int)type])
        this->arrows[0][(int)type]->resetColor();
    if (this->arrows[1][(int)type])
        this->arrows[1][(int)type]->resetColor();
}
```

7.25.4.12 resetColorNode()

```
void NodeInfo::resetColorNode ( )
```

Definition at line 71 of file NodeInfo.cpp.

```
00071                                {
00072     this->node->resetColor();
00073 }
```

7.25.4.13 resetTitle()

```
void NodeInfo::resetTitle ( )
```

Definition at line 227 of file NodeInfo.cpp.

```
00227                            {
00228     this->title.setString("");
00229 }
```

7.25.4.14 setArrows()

```
void NodeInfo::setArrows (
            NodeInfo::ArrowType type,
            sf::Vector2f start,
            sf::Vector2f end )
```

Definition at line 172 of file NodeInfo.cpp.

```
00172                                                                                                       {
00173     if (this->arrows[0][(int)type])
00174         this->arrows[0][(int)type]->setPositions(start, end, false);
00175     if (this->arrows[1][(int)type])
00176         this->arrows[1][(int)type]->setPositions(start, end, true);
00177 }
```

7.25.4.15 setEffectivePosition()

```
void NodeInfo::setEffectivePosition (
            sf::Vector2f start )
```

Definition at line 168 of file NodeInfo.cpp.

```
00168                                                         {
00169     this->positions[(int)TypeNode::Effective] = start;
00170 }
```

7.25.4.16 setNodeInChain()

```
void NodeInfo::setNodeInChain ( )
```

Definition at line 122 of file NodeInfo.cpp.

```
00122                                {
00123     this->statusNode = StatusNode::InChain;
00124 }
```

7.25.4.17 setNodeOutside()

```
void NodeInfo::setNodeOutside ( )
```

Definition at line 118 of file NodeInfo.cpp.

```
00118                                  {
00119     this->statusNode = StatusNode::OutChain;
00120 }
```

7.25.4.18 setNodeVisible()

```
void NodeInfo::setNodeVisible ( )
```

Definition at line 126 of file NodeInfo.cpp.

```
00126                                  {
00127     this->statusNode = StatusNode::Visible;
00128 }
```

7.25.4.19 setPrintNormal()

```
void NodeInfo::setPrintNormal ( )
```

Definition at line 114 of file NodeInfo.cpp.

```
00114                                  {
00115     this->isPrintNormal = true;
00116 }
```

7.25.4.20 setPrintPreVal()

```
void NodeInfo::setPrintPreVal ( )
```

Definition at line 110 of file NodeInfo.cpp.

```
00110                                  {
00111     this->isPrintPreVal = true;
00112 }
```

7.25.4.21 setTitle()

```
void NodeInfo::setTitle (
               const std::string & title )
```

Definition at line 210 of file NodeInfo.cpp.

```
00210                                                 {
00211     std::string preTitle = this->title.getString();
00212     if (!preTitle.empty())
00213         preTitle += "|";
00214     preTitle += _title;
00215     this->title.setString(preTitle);
00216     sf::Vector2f pos = this->node->getPosition();
00217     this->title.setOrigin(
00218             this->title.getGlobalBounds().width / 2,
00219             this->title.getGlobalBounds().height / 2
00220             );
00221     this->title.setPosition(
00222             pos.x,
00223             pos.y + constants::TitleNode::offsetY
00224             );
00225 }
```

7.25.4.22 setValue()

```
void NodeInfo::setValue (
            std::string value )
```

Definition at line 202 of file NodeInfo.cpp.

```
00202                                    {
00203     this->values[0] = std::move(value);
00204 }
```

7.25.4.23 show()

```
void NodeInfo::show (
            NodeInfo::ArrowType type )
```

Definition at line 186 of file NodeInfo.cpp.

```
00186                                         {
00187     if (this->arrows[0][(int)type])
00188         this->arrows[0][(int)type]->show();
00189     if (this->arrows[1][(int)type])
00190         this->arrows[1][(int)type]->show();
00191 }
```

7.25.4.24 toggleActiveColorArrow()

```
void NodeInfo::toggleActiveColorArrow (
            NodeInfo::ArrowType type )
```

Definition at line 64 of file NodeInfo.cpp.

```
00064                                                          {
00065     if (this->arrows[0][(int)type])
00066         this->arrows[0][(int)type]->toggleActiveColor();
00067     if (this->arrows[1][(int)type])
00068         this->arrows[1][(int)type]->toggleActiveColor();
00069 }
```

7.25.4.25 toggleActiveColorNode()

```
void NodeInfo::toggleActiveColorNode ( )
```

Definition at line 60 of file NodeInfo.cpp.

```
00060                                    {
00061     this->node->toggleActiveColor();
00062 }
```

7.25.4.26 updateArrows()

```
void NodeInfo::updateArrows (
             ArrowType type,
             sf::Vector2f end )
```

Definition at line 152 of file NodeInfo.cpp.

```
                                                    {
    if (this->arrows[0][(int)type])
        this->arrows[0][(int)type]->setPositions(this->node->getPosition(), end, false);

    if (this->arrows[1][(int)type])
        this->arrows[1][(int)type]->setPositions(this->node->getPosition(), end, true);
}
```

7.25.4.27 updateNode()

```
void NodeInfo::updateNode ( )
```

Definition at line 131 of file NodeInfo.cpp.

```
                             {
    if (this->statusNode == StatusNode::Visible)
        return;

    if (this->statusNode == StatusNode::InChain) {
        if (this->isPrintNormal) {
            this->node->setPosition(this->positions[(int)TypeNode::Normal]);
        } else {
            this->node->setPosition(this->positions[(int)TypeNode::Effective]);
        }
    } else {
        this->node->setPosition(this->positions[(int)TypeNode::Outside]);
    }

    if (this->isPrintPreVal) {
        this->node->setText(this->values[1]);
    } else {
        this->node->setText(this->values[0]);
    }
}
```

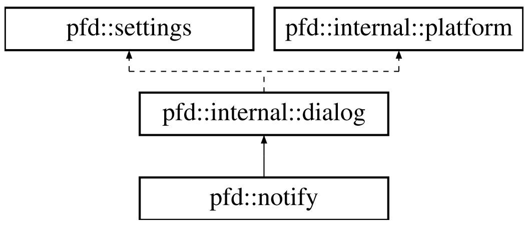
The documentation for this class was generated from the following files:

- include/draw/NodeInfo.hpp
- include/draw/NodeInfo.cpp

7.26 pfd::notify Class Reference

```
#include <FileDialog.h>
```

Inheritance diagram for pfd::notify:



Public Member Functions

- notify (std::string const &title, std::string const &message, icon _icon=icon::info)

Public Member Functions inherited from pfd::internal::dialog

- bool ready (int timeout=default_wait_timeout) const
- bool kill () const

Additional Inherited Members

Protected Types inherited from pfd::settings

- enum class flag {
  is_scanned = 0 , is_verbose , has_zenity , has_matedialog ,
  has_qarma , has_kdialog , is_vista , max_flag }

Protected Member Functions inherited from pfd::internal::dialog

- dialog ()
- std::vector< std::string > desktop_helper () const
- std::string powershell_quote (std::string const &str) const
- std::string osascript_quote (std::string const &str) const
- std::string shell_quote (std::string const &str) const

Protected Member Functions inherited from pfd::settings

- settings (bool resync=false)
- bool check_program (std::string const &program)
- bool is_osascript () const
- bool is_zenity () const
- bool is_kdialog () const
- bool const & flags (flag in_flag) const
- bool & flags (flag in_flag)

Static Protected Member Functions inherited from pfd::internal::dialog

- static std::string buttons_to_name (choice _choice)
- static std::string get_icon_name (icon _icon)

Static Protected Member Functions inherited from pfd::settings

- static bool available ()
- static void verbose (bool value)
- static void rescan ()

Protected Attributes inherited from pfd::internal::dialog

- std::shared_ptr< executor > m_async

7.26.1 Detailed Description

Definition at line 336 of file FileDialog.h.

7.26.2 Constructor & Destructor Documentation

7.26.2.1 notify()

```
pfd::notify::notify (
             std::string const & title,
             std::string const & message,
             icon _icon = icon::info )  [inline]
```

Definition at line 1487 of file FileDialog.h.

```
    {
        if (_icon == icon::question) // Not supported by notifications
            _icon = icon::info;

#if _WIN32
        // Use a static shared pointer for notify_icon so that we can delete
    // it whenever we need to display a new one, and we can also wait
    // until the program has finished running.
    struct notify_icon_data : public NOTIFYICONDATAW
    {
        ~notify_icon_data() { Shell_NotifyIconW(NIM_DELETE, this); }
    };

    static std::shared_ptr<notify_icon_data> nid;

    // Release the previous notification icon, if any, and allocate a new
    // one. Note that std::make_shared() does value initialization, so there
    // is no need to memset the structure.
    nid = nullptr;
    nid = std::make_shared<notify_icon_data>();

    // For XP support
    nid->cbSize = NOTIFYICONDATAW_V2_SIZE;
    nid->hWnd = nullptr;
    nid->uID = 0;

    // Flag Description:
    // - NIF_ICON    The hIcon member is valid.
    // - NIF_MESSAGE The uCallbackMessage member is valid.
    // - NIF_TIP     The szTip member is valid.
    // - NIF_STATE   The dwState and dwStateMask members are valid.
    // - NIF_INFO    Use a balloon ToolTip instead of a standard ToolTip. The szInfo, uTimeout,
szInfoTitle, and dwInfoFlags members are valid.
    // - NIF_GUID    Reserved.
    nid->uFlags = NIF_MESSAGE | NIF_ICON | NIF_INFO;

    // Flag Description
    // - NIIF_ERROR     An error icon.
    // - NIIF_INFO      An information icon.
    // - NIIF_NONE      No icon.
    // - NIIF_WARNING   A warning icon.
    // - NIIF_ICON_MASK Version 6.0. Reserved.
    // - NIIF_NOSOUND   Version 6.0. Do not play the associated sound. Applies only to balloon
ToolTips
    switch (_icon)
    {
        case icon::warning: nid->dwInfoFlags = NIIF_WARNING; break;
        case icon::error: nid->dwInfoFlags = NIIF_ERROR; break;
        /* case icon::info: */ default: nid->dwInfoFlags = NIIF_INFO; break;
    }

    ENUMRESNAMEPROC icon_enum_callback = [](HMODULE, LPCTSTR, LPTSTR lpName, LONG_PTR lParam) -> BOOL
    {
        ((NOTIFYICONDATAW *)lParam)->hIcon = ::LoadIcon(GetModuleHandle(nullptr), lpName);
        return false;
    };

    nid->hIcon = ::LoadIcon(nullptr, IDI_APPLICATION);
    ::EnumResourceNames(nullptr, RT_GROUP_ICON, icon_enum_callback, (LONG_PTR)nid.get());

    nid->uTimeout = 5000;

    StringCchCopyW(nid->szInfoTitle, ARRAYSIZE(nid->szInfoTitle), internal::str2wstr(title).c_str());
    StringCchCopyW(nid->szInfo, ARRAYSIZE(nid->szInfo), internal::str2wstr(message).c_str());

    // Display the new icon
    Shell_NotifyIconW(NIM_ADD, nid.get());
#elif __EMSCRIPTEN__
        // FIXME: do something
    (void)title;
    (void)message;
#else
        auto command = desktop_helper();

        if (is_osascript())
        {
            command.push_back("-e");
            command.push_back("display notification " + osascript_quote(message) +
                              " with title " + osascript_quote(title));
        }
        else if (is_zenity())
        {
            command.push_back("--notification");
            command.push_back("--window-icon");
            command.push_back(get_icon_name(_icon));
            command.push_back("--text");
            command.push_back(title + "\n" + message);
        }
        else if (is_kdialog())
        {
            command.push_back("--icon");
            command.push_back(get_icon_name(_icon));
            command.push_back("--title");
            command.push_back(title);
            command.push_back("--passivepopup");
            command.push_back(message);
            command.push_back("5");
        }

        if (flags(flag::is_verbose))
            std::cerr « "pfd: " « command « std::endl;

        m_async->start_process(command);
#endif
    }
```

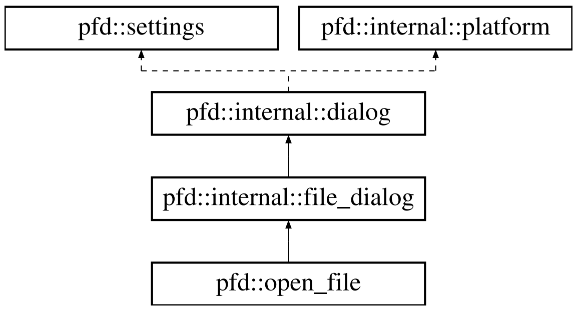
The documentation for this class was generated from the following file:

- include/core/FileDialog.h

7.27 pfd::open_file Class Reference

```
#include <FileDialog.h>
```

Inheritance diagram for pfd::open_file:



Public Member Functions

- open_file (std::string const &title, std::string const &default_path="", std::vector< std::string > const &filters={ "All Files", "*" }, opt options=opt::none)
- open_file (std::string const &title, std::string const &default_path, std::vector< std::string > const &filters, bool allow_multiselect)
- std::vector< std::string > result ()

Public Member Functions inherited from pfd::internal::dialog

- bool ready (int timeout=default_wait_timeout) const
- bool kill () const

Additional Inherited Members

Protected Types inherited from pfd::internal::file_dialog

- enum type { open , save , folder }

Protected Types inherited from pfd::settings

- enum class flag {
  is_scanned = 0 , is_verbose , has_zenity , has_matedialog ,
  has_qarma , has_kdialog , is_vista , max_flag }

Protected Member Functions inherited from pfd::internal::file_dialog

- file_dialog (type in_type, std::string const &title, std::string const &default_path="", std::vector< std::string > const &filters={}, opt options=opt::none)
- std::string string_result ()
- std::vector< std::string > vector_result ()

Protected Member Functions inherited from pfd::internal::dialog

- dialog ()
- std::vector< std::string > desktop_helper () const
- std::string powershell_quote (std::string const &str) const
- std::string osascript_quote (std::string const &str) const
- std::string shell_quote (std::string const &str) const

Protected Member Functions inherited from pfd::settings

- settings (bool resync=false)
- bool check_program (std::string const &program)
- bool is_osascript () const
- bool is_zenity () const
- bool is_kdialog () const
- bool const & flags (flag in_flag) const
- bool & flags (flag in_flag)

Static Protected Member Functions inherited from pfd::internal::dialog

- static std::string buttons_to_name (choice _choice)
- static std::string get_icon_name (icon _icon)

Static Protected Member Functions inherited from pfd::settings

- static bool available ()
- static void verbose (bool value)
- static void rescan ()

Protected Attributes inherited from pfd::internal::dialog

- std::shared_ptr< executor > m_async

7.27.1 Detailed Description

Definition at line 367 of file FileDialog.h.

7.27.2 Constructor & Destructor Documentation

7.27.2.1 open_file() [1/2]

```
pfd::open_file::open_file (
            std::string const & title,
            std::string const & default_path = "",
            std::vector< std::string > const & filters = { "All Files", "*" },
            opt options = opt::none )  [inline]
```

Definition at line 1825 of file FileDialog.h.

```
            : file_dialog(type::open, title, default_path, filters, options)
    {
    }
```

7.27.2.2 open_file() [2/2]

```
pfd::open_file::open_file (
            std::string const & title,
            std::string const & default_path,
            std::vector< std::string > const & filters,
            bool allow_multiselect )  [inline]
```

Definition at line 1833 of file FileDialog.h.

```
            : open_file(title, default_path, filters,
                        (allow_multiselect ? opt::multiselect : opt::none))
    {
    }
```

7.27.3 Member Function Documentation

7.27.3.1 result()

```
std::vector< std::string > pfd::open_file::result ( ) [inline]
```

Definition at line 1842 of file FileDialog.h.

```
01843     {
01844         return vector_result();
01845     }
```

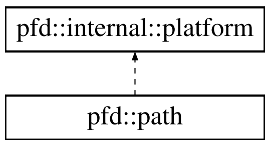
The documentation for this class was generated from the following file:

- include/core/FileDialog.h

7.28 pfd::path Class Reference

```
#include <FileDialog.h>
```

Inheritance diagram for pfd::path:



Static Public Member Functions

- static std::string home ()
- static std::string separator ()

7.28.1 Detailed Description

Definition at line 325 of file FileDialog.h.

7.28.2 Member Function Documentation

7.28.2.1 home()

```
std::string pfd::path::home ( )  [inline], [static]
```

Definition at line 637 of file FileDialog.h.

```
    {
#if _WIN32
        // First try the USERPROFILE environment variable
    auto user_profile = internal::getenv("USERPROFILE");
    if (user_profile.size() > 0)
        return user_profile;
    // Otherwise, try GetUserProfileDirectory()
    HANDLE token = nullptr;
    DWORD len = MAX_PATH;
    char buf[MAX_PATH] = { '\0' };
    if (OpenProcessToken(GetCurrentProcess(), TOKEN_QUERY, &token))
    {
        dll userenv("userenv.dll");
        dll::proc<BOOL WINAPI (HANDLE, LPSTR, LPDWORD)> get_user_profile_directory(userenv,
"GetUserProfileDirectoryA");
        get_user_profile_directory(token, buf, &len);
        CloseHandle(token);
        if (*buf)
            return buf;
    }
#elif __EMSCRIPTEN__
        return "/";
#else
        // First try the HOME environment variable
        auto home = internal::getenv("HOME");
        if (home.size() > 0)
            return home;
        // Otherwise, try getpwuid_r()
        size_t len = 4096;
#if defined(_SC_GETPW_R_SIZE_MAX)
        auto size_max = sysconf(_SC_GETPW_R_SIZE_MAX);
        if (size_max != -1)
            len = size_t(size_max);
#endif
        std::vector<char> buf(len);
        struct passwd pwd, *result;
        if (getpwuid_r(getuid(), &pwd, buf.data(), buf.size(), &result) == 0)
            return result->pw_dir;
#endif
        return "/";
    }
```

7.28.2.2 separator()

```
std::string pfd::path::separator ( )  [inline], [static]
```

Definition at line 679 of file FileDialog.h.

```
    {
#if _WIN32
        return "\\";
#else
        return "/";
#endif
    }
```

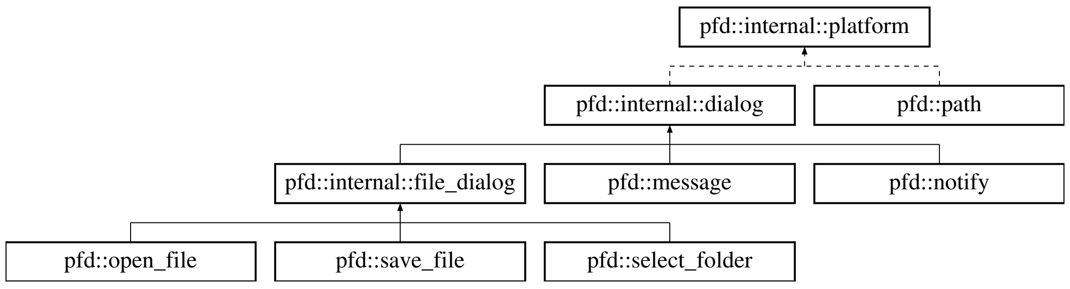
The documentation for this class was generated from the following file:

- include/core/FileDialog.h

7.29 pfd::internal::platform Class Reference

```
#include <FileDialog.h>
```

Inheritance diagram for pfd::internal::platform:



7.29.1 Detailed Description

Definition at line 210 of file FileDialog.h.

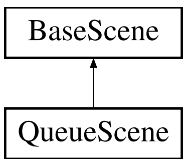
The documentation for this class was generated from the following file:

- include/core/FileDialog.h

7.30 QueueScene Class Reference

```
#include <QueueScene.hpp>
```

Inheritance diagram for QueueScene:



Public Member Functions

- QueueScene (sf::RenderWindow ∗window)
- void reset ()
- void pollEvent (sf::Event event, sf::Vector2f mousePosView) override
- void update () override
- void render () override
- std::vector< EventAnimation > pushModeEvents (int chosenNode)
- std::vector< EventAnimation > popModeEvents (int chosenNode)

Public Member Functions inherited from BaseScene

- BaseScene (sf::RenderWindow ∗window)
- void createModeButton (sf::Vector2f position, std::string textString)
- virtual void pollEvent (sf::Event event, sf::Vector2f mousePosView)=0
- virtual void update ()=0
- virtual void render ()=0

Additional Inherited Members

Public Attributes inherited from BaseScene

- Button ∗ modeButton {}
- bool isMenuOpen {}
- bool isDemoCodeOpen {}

Protected Member Functions inherited from BaseScene

- void setWindow (sf::RenderWindow ∗window)

Protected Attributes inherited from BaseScene

- sf::RenderWindow ∗ window {}
- ControlMenu ∗ controlMenu

7.30.1 Detailed Description

Definition at line 12 of file QueueScene.hpp.

7.30.2 Constructor & Destructor Documentation

7.30.2.1 QueueScene()

```
QueueScene::QueueScene (
            sf::RenderWindow ∗ window )   [explicit]
```

Definition at line 7 of file QueueScene.cpp.

```
00007                                          : BaseScene(window) {
00008     this->init();
00009 }
```

7.30.3 Member Function Documentation

7.30.3.1 pollEvent()

```
void QueueScene::pollEvent (
            sf::Event event,
            sf::Vector2f mousePosView )  [override], [virtual]
```

Implements BaseScene.

Definition at line 111 of file QueueScene.cpp.

```
                                                                         {
    if (this->isMenuOpen)
        this->menu->pollEvents(event, mousePosView);

    this->controlMenu->pollEvents(event, mousePosView);
}
```

7.30.3.2 popModeEvents()

```
std::vector< EventAnimation > QueueScene::popModeEvents (
            int chosenNode )
```

Definition at line 260 of file QueueScene.cpp.

```
                                                                          {
    this->linkedList->resetEvents();
    if (chosenNode < 0 || chosenNode >= this->linkedList->getSize())
        return {};

    this->linkedList->initHighlighter(
            constants::Highlighter::SLL::CODES_PATH[1].second,
            constants::Highlighter::SLL::CODES_PATH[1].first
    );

    std::vector<EventAnimation> events;
    EventAnimation event;

    if (!chosenNode) {
        event.titleNodes.emplace_back(chosenNode, "head|temp");
        event.colorNodes.push_back(chosenNode);
        event.statusChosenNode = NodeInfo::StatusNode::InChain;
        event.lines = {0, 1};

        events.emplace_back(event);

        if (this->linkedList->getSize() > 1) {
            event.reset();
            event.titleNodes = {
                    {chosenNode, "temp"},
                    {1, "head"}
            };
            event.colorNodes.push_back(1);
            event.colorArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
            event.isPrintNormal = true;
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.lines = {2};

            events.emplace_back(event);
        }

        event.reset();
        event.titleNodes.emplace_back(1, "head");
        event.statusChosenNode = NodeInfo::StatusNode::Visible;
        event.lines = {3};

        events.emplace_back(event);
    } else {
        event.reset();
        event.titleNodes.emplace_back(0, "head|current");
        event.colorNodes.push_back(0);
        event.statusChosenNode = NodeInfo::StatusNode::InChain;
        event.lines = {5};

        events.emplace_back(event);

        for (int i = 0; i < chosenNode; ++i) {
            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {i, "current"}
            };
            event.colorNodes.push_back(i);
            event.statusChosenNode = NodeInfo::StatusNode::InChain;
            event.lines = {6};

            events.emplace_back(event);

            if (i == chosenNode - 1) break;

            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {i, "current"}
            };
            event.colorNodes.push_back(i);
            event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::InChain;
            event.lines = {7};

            events.emplace_back(event);
        }

        event.reset();
        event.titleNodes = {
                {0, "head"},
                {chosenNode, "temp"},
                {chosenNode - 1, "current"}
        };
        event.colorNodes.push_back(chosenNode);
        event.colorArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
        event.statusChosenNode = NodeInfo::StatusNode::InChain;
        event.lines = {8};

        events.emplace_back(event);

        if (chosenNode != this->linkedList->getSize() - 1) {
            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {chosenNode, "temp"},
                    {chosenNode - 1, "current"}
            };
            event.colorNodes.push_back(chosenNode);
            event.colorArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.isPrintNormal = true;
            event.lines = {9};

            events.emplace_back(event);

            event.reset();
            event.titleNodes.emplace_back(0, "head");
            event.statusChosenNode = NodeInfo::StatusNode::Visible;
            event.lines = {10};

            events.emplace_back(event);
        } else {
            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {chosenNode, "temp"},
                    {chosenNode - 1, "current"}
            };
            event.colorNodes.push_back(chosenNode);
            event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.lines = {9};

            events.emplace_back(event);

            event.reset();
            event.titleNodes.emplace_back(0, "head");
            event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::Visible;
            event.lines = {10};

            events.emplace_back(event);
        }
    }

    return events;
}
```

7.30.3.3 pushModeEvents()

```
std::vector< EventAnimation > QueueScene::pushModeEvents (
              int chosenNode )
```

Definition at line 127 of file QueueScene.cpp.

```
00127                                                                     {
00128     this->linkedList->resetEvents();
00129     if (chosenNode < 0 || chosenNode > this->linkedList->getSize())
00130         return {};
00131 
00132     this->linkedList->initHighlighter(
00133             constants::Highlighter::SLL::CODES_PATH[0].second,
00134             constants::Highlighter::SLL::CODES_PATH[0].first
00135     );
00136 
00137     std::vector<EventAnimation> events;
00138     EventAnimation event;
00139 
00140     if (chosenNode)
00141         event.titleNodes = {
00142                 {0, "head"},
00143                 {chosenNode, "temp"}
00144         };
00145     else {
00146         event.titleNodes.emplace_back(chosenNode, "temp");
00147         if (this->linkedList->getSize())
00148             event.titleNodes.emplace_back(1, "head");
00149     }
00150     event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00151     if (chosenNode && chosenNode == this->linkedList->getSize())
00152         event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00153     event.colorNodes.push_back(chosenNode);
00154     event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00155     event.lines = {0};
00156 
00157     events.emplace_back(event);
00158 
00159     if (chosenNode == 0) {
00160         if (this->linkedList->getSize()) {
00161             event.reset();
00162             event.titleNodes = {
00163                     {1, "head"},
00164                     {chosenNode, "temp"}
00165             };
00166             event.colorNodes = std::vector<int>{0};
00167             event.colorArrows.emplace_back(0, NodeInfo::ArrowType::RIGHT);
00168             event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00169             event.isPrintNormal = true;
00170             event.lines = {1, 2};
00171 
00172             events.emplace_back(event);
00173         }
00174 
00175         event.reset();
00176         event.titleNodes.emplace_back(chosenNode, "head|temp");
00177         event.lines = {3};
00178         event.statusChosenNode = NodeInfo::StatusNode::InChain;
00179         events.emplace_back(event);
00180     } else {
00181         event.reset();
00182         event.titleNodes = {
00183                 {0, "head|current"},
00184                 {chosenNode, "temp"}
00185         };
00186         event.colorNodes.push_back(0);
00187         event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00188         if (chosenNode == this->linkedList->getSize())
00189             event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00190         event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00191         event.lines = {5};
00192 
00193         events.emplace_back(event);
00194 
00195         for (int i = 0; i < chosenNode; ++i) {
00196             event.reset();
00197             event.titleNodes = {
00198                     {0, "head"},
00199                     {chosenNode, "temp"},
                    {i, "current"}
            };
            event.colorNodes.push_back(i);
            event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
            if (chosenNode == this->linkedList->getSize())
                event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.lines = {6};

            events.emplace_back(event);

            if (i == chosenNode - 1) break;

            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {chosenNode, "temp"},
                    {i, "current"}
            };
            event.colorNodes.push_back(i);
            event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
            event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
            if (chosenNode == this->linkedList->getSize())
                event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.lines = {7};

            events.emplace_back(event);
        }

        if (chosenNode != this->linkedList->getSize()) {
            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {chosenNode, "temp"},
                    {chosenNode - 1, "current"}
            };
            event.colorNodes.push_back(chosenNode);
            event.colorArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.isPrintNormal = true;
            event.lines = {8};

            events.emplace_back(event);
        }

        event.reset();
        event.titleNodes = {
                {0, "head"},
                {chosenNode, "temp"}
        };
        event.statusChosenNode = NodeInfo::StatusNode::InChain;
        event.lines = {9};

        events.emplace_back(event);
    }

    return events;
}
```

7.30.3.4 render()

```
void QueueScene::render ( ) [override], [virtual]
```

Implements BaseScene.

Definition at line 100 of file QueueScene.cpp.

```
                          {
    if (this->isMenuOpen)
        this->menu->render();

    if (this->isDemoCodeOpen)
        this->linkedList->renderHighlighter();

    this->controlMenu->render();
    this->linkedList->render();
}
```

7.30.3.5 reset()

```
void QueueScene::reset ( )
```

Definition at line 123 of file QueueScene.cpp.

```
                            {
    this->menu->resetActiveOptionMenu();
}
```

7.30.3.6 update()

```
void QueueScene::update ( )  [override], [virtual]
```

Implements BaseScene.

Definition at line 11 of file QueueScene.cpp.

```
                             {
    if (this->isMenuOpen) {
        this->menu->update();

        constants::MenuDataStructure::Button status = this->menu->getActiveOptionMenu();
        constants::MenuDataStructure::CreateMode::Button createMode;
        switch (status) {
            case constants::MenuDataStructure::Button::CREATE_BUTTON:
                createMode = this->menu->getActiveCreateMode();
                if (createMode == constants::MenuDataStructure::CreateMode::Button::RANDOM_BUTTON) {
                    if (this->menu->createModeValue[0] == "None")
                        break;
                    if (this->menu->createModeValue[0].empty())
                        this->menu->createModeValue[0] = "0";
                    int size = std::stoi(this->menu->createModeValue[0]);
                    this->linkedList->createLinkedList(size);
                } else if (createMode ==
      constants::MenuDataStructure::CreateMode::Button::DEFINED_LIST_BUTTON) {
                    if (this->menu->createModeValue[1] == "None")
                        break;
                    std::vector<std::string> values;
                    std::string value = this->menu->createModeValue[1];
                    std::stringstream ss(value);
                    std::string token;
                    while (std::getline(ss, token, ',')) {
                        values.push_back(token);
                    }
                    this->linkedList->createLinkedList(values);
                } else if (createMode ==
      constants::MenuDataStructure::CreateMode::Button::FILE_BUTTON) {
                    if (this->menu->createModeValue[2] == "None")
                        break;
                    std::vector<std::string> values;
                    std::string value = this->menu->createModeValue[2];
                    std::stringstream ss(value);
                    std::string token;
                    while (std::getline(ss, token, ','))
                        values.push_back(token);
                    this->linkedList->createLinkedList(values);
                    this->menu->createModeValue[2] = "None";
                }
                this->controlMenu->reset();
                break;
            case constants::MenuDataStructure::Button::PUSH_BUTTON:
                if (this->menu->pushModeValue == "None")
                    break;

                this->linkedList->addNode(
                        this->linkedList->getSize(),
                        this->menu->pushModeValue,
                        this->pushModeEvents(this->linkedList->getSize())
                );

                std::cout « "Pushed " « this->menu->pushModeValue « std::endl;
                this->menu->pushModeValue = "None";
                this->controlMenu->reset();
                break;
            case constants::MenuDataStructure::Button::POP_BUTTON:
                if (this->menu->getActiveOptionMenu() != constants::MenuDataStructure::Button::POP_BUTTON)
                    break;

                this->linkedList->deleteNode(
                        0,
                        this->popModeEvents(0)
                );

                std::cout « "Popped " « std::endl;
                this->menu->resetActiveOptionMenuOnly();
                this->controlMenu->reset();
                break;
            case constants::MenuDataStructure::Button::CLEAR_BUTTON:
                if (this->menu->getActiveOptionMenu() != constants::MenuDataStructure::Button::CLEAR_BUTTON)
                    break;

                this->linkedList->createLinkedList(0);

                std::cout « "Cleared " « std::endl;
                this->menu->resetActiveOptionMenuOnly();
                this->controlMenu->reset();
                break;
        }
    }

    this->controlMenu->update();

    this->linkedList->processControlMenu(this->controlMenu->getStatus());
    this->linkedList->setSpeed(this->controlMenu->getSpeed());

    this->linkedList->update();
}
```

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

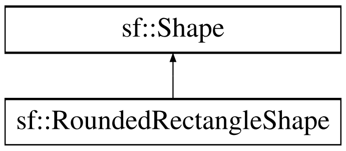
- include/libScene/QueueScene.hpp
- include/libScene/QueueScene.cpp

7.31 sf::RoundedRectangleShape Class Reference

Specialized shape representing a rectangle with rounded corners.

`#include <RoundedRectangleShape.hpp>`

Inheritance diagram for sf::RoundedRectangleShape:



Public Member Functions

- RoundedRectangleShape (const Vector2f &size=Vector2f(0, 0), float radius=0, unsigned int cornerPoint↩
Count=0)

    Default constructor.
- void setSize (const Vector2f &size)

    Set the size of the rounded rectangle.
- const Vector2f & getSize () const

    Get the size of the rounded rectangle.

- void setCornersRadius (float radius)

  Set the radius of the rounded corners.
- float getCornersRadius () const

  Get the radius of the rounded corners.
- void setCornerPointCount (unsigned int count)

  Set the number of points of each corner.
- virtual std::size_t getPointCount () const

  Get the number of points defining the rounded rectangle.
- virtual sf::Vector2f getPoint (std::size_t index) const

  Get a point of the rounded rectangle.

7.31.1 Detailed Description

Specialized shape representing a rectangle with rounded corners.

This class inherits all the functions of sf::Transformable (position, rotation, scale, bounds, ...) as well as the functions of sf::Shape (outline, color, texture, ...).

Usage example:
```
sf::RoundedRectangleShape roundedRectangle;
rectangle.setSize(sf::Vector2f(100, 50));
rectangle.setCornersRadius(5);
rectangle.setOutlineThickness(5);
rectangle.setPosition(10, 20);
...
window.draw(rectangle);
```

See also

sf::Shape, sf::CircleShape, sf::ConvexShape

Definition at line 36 of file RoundedRectangleShape.hpp.

7.31.2 Constructor & Destructor Documentation

7.31.2.1 RoundedRectangleShape()

```
sf::RoundedRectangleShape::RoundedRectangleShape (
             const Vector2f & size = Vector2f(0, 0),
             float radius = 0,
             unsigned int cornerPointCount = 0 )  [explicit]
```

Default constructor.

Parameters

size	Size of the rectangle
radius	Radius for each rounded corner
cornerPointCount	Number of points of each corner

Definition at line 31 of file RoundedRectangleShape.cpp.

```
{
    mySize = size;
    myRadius = radius;
    myCornerPointCount = cornerPointCount;
    update();
}
```

7.31.3 Member Function Documentation

7.31.3.1 getCornersRadius()

`float sf::RoundedRectangleShape::getCornersRadius ( ) const`

Get the radius of the rounded corners.

Returns

Radius of the rounded corners

See also

setCornersRadius

Definition at line 60 of file RoundedRectangleShape.cpp.

```
{
    return myRadius;
}
```

7.31.3.2 getPoint()

```
sf::Vector2f sf::RoundedRectangleShape::getPoint (
                std::size_t index ) const  [virtual]
```

Get a point of the rounded rectangle.

The result is undefined if *index* is out of the valid range.

Parameters

index	Index of the point to get, in range [0 .. GetPointCount() - 1]

Returns

Index-th point of the shape

Definition at line 79 of file RoundedRectangleShape.cpp.

```
    {
        if(index >= myCornerPointCount*4)
            return sf::Vector2f(0,0);

        float deltaAngle = 90.0f/(myCornerPointCount-1);
        sf::Vector2f center;
        unsigned int centerIndex = index/myCornerPointCount;
        static const float pi = 3.141592654f;

        switch(centerIndex)
        {
            case 0: center.x = mySize.x - myRadius; center.y = myRadius; break;
            case 1: center.x = myRadius; center.y = myRadius; break;
            case 2: center.x = myRadius; center.y = mySize.y - myRadius; break;
            case 3: center.x = mySize.x - myRadius; center.y = mySize.y - myRadius; break;
        }

        return sf::Vector2f(myRadius*cos(deltaAngle*(index-centerIndex)*pi/180)+center.x,
                            -myRadius*sin(deltaAngle*(index-centerIndex)*pi/180)+center.y);
    }
```

7.31.3.3 getPointCount()

`std::size_t sf::RoundedRectangleShape::getPointCount ( ) const [virtual]`

Get the number of points defining the rounded rectangle.

Returns

Number of points of the rounded rectangle

Definition at line 73 of file RoundedRectangleShape.cpp.

```
    {
        return myCornerPointCount*4;
    }
```

7.31.3.4 getSize()

`const Vector2f & sf::RoundedRectangleShape::getSize ( ) const`

Get the size of the rounded rectangle.

Returns

Size of the rounded rectangle

See also

setSize

Definition at line 47 of file RoundedRectangleShape.cpp.

```
    {
        return mySize;
    }
```

7.31.3.5 setCornerPointCount()

```
void sf::RoundedRectangleShape::setCornerPointCount (
            unsigned int count )
```

Set the number of points of each corner.

Parameters

count	New number of points of the rounded rectangle

See also

getPointCount

Definition at line 66 of file RoundedRectangleShape.cpp.

```
    {
        myCornerPointCount = count;
        update();
    }
```

7.31.3.6 setCornersRadius()

```
void sf::RoundedRectangleShape::setCornersRadius (
            float radius )
```

Set the radius of the rounded corners.

Parameters

radius	Radius of the rounded corners

See also

getCornersRadius

Definition at line 53 of file RoundedRectangleShape.cpp.

```
    {
        myRadius = radius;
        update();
    }
```

7.31.3.7 setSize()

```
void sf::RoundedRectangleShape::setSize (
            const Vector2f & size )
```

Set the size of the rounded rectangle.

Parameters

size	New size of the rounded rectangle

See also

getSize

Definition at line 40 of file RoundedRectangleShape.cpp.

```
00041     {
00042         mySize = size;
00043         update();
00044     }
```

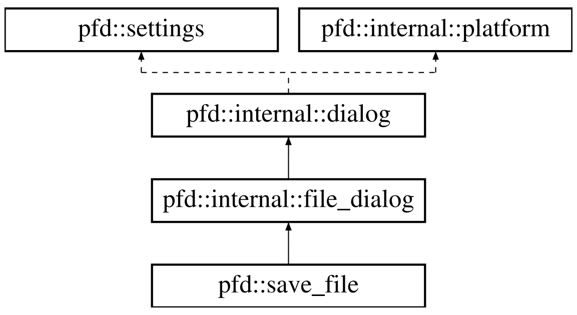
The documentation for this class was generated from the following files:

- include/stuff/RoundedRectangleShape.hpp
- include/stuff/RoundedRectangleShape.cpp

7.32 pfd::save_file Class Reference

```
#include <FileDialog.h>
```

Inheritance diagram for pfd::save_file:



Public Member Functions

- save_file (std::string const &title, std::string const &default_path="", std::vector< std::string > const &filters={ "All Files", "∗" }, opt options=opt::none)
- save_file (std::string const &title, std::string const &default_path, std::vector< std::string > const &filters, bool confirm_overwrite)
- std::string result ()

Public Member Functions inherited from pfd::internal::dialog

- bool ready (int timeout=default_wait_timeout) const
- bool kill () const

Additional Inherited Members

Protected Types inherited from pfd::internal::file_dialog

- enum type { open , save , folder }

Protected Types inherited from pfd::settings

- enum class flag {
  is_scanned = 0 , is_verbose , has_zenity , has_matedialog ,
  has_qarma , has_kdialog , is_vista , max_flag }

Protected Member Functions inherited from pfd::internal::file_dialog

- file_dialog (type in_type, std::string const &title, std::string const &default_path="", std::vector< std::string > const &filters={}, opt options=opt::none)
- std::string string_result ()
- std::vector< std::string > vector_result ()

Protected Member Functions inherited from pfd::internal::dialog

- dialog ()
- std::vector< std::string > desktop_helper () const
- std::string powershell_quote (std::string const &str) const
- std::string osascript_quote (std::string const &str) const
- std::string shell_quote (std::string const &str) const

Protected Member Functions inherited from pfd::settings

- settings (bool resync=false)
- bool check_program (std::string const &program)
- bool is_osascript () const
- bool is_zenity () const
- bool is_kdialog () const
- bool const & flags (flag in_flag) const
- bool & flags (flag in_flag)

Static Protected Member Functions inherited from pfd::internal::dialog

- static std::string buttons_to_name (choice _choice)
- static std::string get_icon_name (icon _icon)

Static Protected Member Functions inherited from pfd::settings

- static bool available ()
- static void verbose (bool value)
- static void rescan ()

Protected Attributes inherited from pfd::internal::dialog

- std::shared_ptr< executor > m_async

7.32.1 Detailed Description

Definition at line 389 of file FileDialog.h.

7.32.2 Constructor & Destructor Documentation

7.32.2.1 save_file() [1/2]

```
pfd::save_file::save_file (
            std::string const & title,
            std::string const & default_path = "",
            std::vector< std::string > const & filters = { "All Files", "*" },
            opt options = opt::none )  [inline]
```

Definition at line 1849 of file FileDialog.h.

```
            : file_dialog(type::save, title, default_path, filters, options)
    {
    }
```

7.32.2.2 save_file() [2/2]

```
pfd::save_file::save_file (
            std::string const & title,
            std::string const & default_path,
            std::vector< std::string > const & filters,
            bool confirm_overwrite )  [inline]
```

Definition at line 1857 of file FileDialog.h.

```
            : save_file(title, default_path, filters,
                        (confirm_overwrite ? opt::none : opt::force_overwrite))
    {
    }
```

7.32.3 Member Function Documentation

7.32.3.1 result()

```
std::string pfd::save_file::result ( )  [inline]
```

Definition at line 1866 of file FileDialog.h.

```
    {
        return string_result();
    }
```

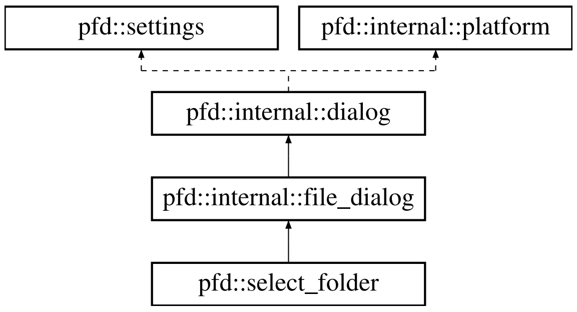
The documentation for this class was generated from the following file:

- include/core/FileDialog.h

7.33 pfd::select_folder Class Reference

```
#include <FileDialog.h>
```

Inheritance diagram for pfd::select_folder:



Public Member Functions

- select_folder (std::string const &title, std::string const &default_path="", opt options=opt::none)
- std::string result ()

Public Member Functions inherited from pfd::internal::dialog

- bool ready (int timeout=default_wait_timeout) const
- bool kill () const

Additional Inherited Members

Protected Types inherited from pfd::internal::file_dialog

- enum type { open , save , folder }

Protected Types inherited from pfd::settings

- enum class flag {
  is_scanned = 0 , is_verbose , has_zenity , has_matedialog ,
  has_qarma , has_kdialog , is_vista , max_flag }

Protected Member Functions inherited from pfd::internal::file_dialog

- file_dialog (type in_type, std::string const &title, std::string const &default_path="", std::vector< std::string > const &filters={}, opt options=opt::none)
- std::string string_result ()
- std::vector< std::string > vector_result ()

Protected Member Functions inherited from pfd::internal::dialog

- dialog ()
- std::vector< std::string > desktop_helper () const
- std::string powershell_quote (std::string const &str) const
- std::string osascript_quote (std::string const &str) const
- std::string shell_quote (std::string const &str) const

Protected Member Functions inherited from pfd::settings

- settings (bool resync=false)
- bool check_program (std::string const &program)
- bool is_osascript () const
- bool is_zenity () const
- bool is_kdialog () const
- bool const & flags (flag in_flag) const
- bool & flags (flag in_flag)

Static Protected Member Functions inherited from pfd::internal::dialog

- static std::string buttons_to_name (choice _choice)
- static std::string get_icon_name (icon _icon)

Static Protected Member Functions inherited from pfd::settings

- static bool available ()
- static void verbose (bool value)
- static void rescan ()

Protected Attributes inherited from pfd::internal::dialog

- std::shared_ptr< executor > m_async

7.33.1 Detailed Description

Definition at line 411 of file FileDialog.h.

7.33.2 Constructor & Destructor Documentation

7.33.2.1 select_folder()

```
pfd::select_folder::select_folder (
            std::string const & title,
            std::string const & default_path = "",
            opt options = opt::none )  [inline]
```

Definition at line 1873 of file FileDialog.h.

```
01876            : file_dialog(type::folder, title, default_path, {}, options)
01877     {
01878     }
```

7.33.3 Member Function Documentation

7.33.3.1 result()

```
std::string pfd::select_folder::result ( )  [inline]
```

Definition at line 1880 of file FileDialog.h.

```
{
    return string_result();
}
```

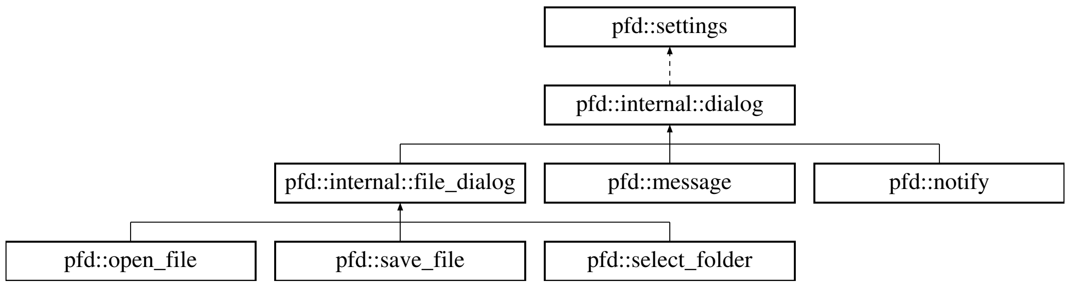
The documentation for this class was generated from the following file:

- include/core/FileDialog.h

7.34 pfd::settings Class Reference

```
#include <FileDialog.h>
```

Inheritance diagram for pfd::settings:



Static Public Member Functions

- static bool available ()
- static void verbose (bool value)
- static void rescan ()

Protected Types

- enum class flag {
  is_scanned = 0 , is_verbose , has_zenity , has_matedialog ,
  has_qarma , has_kdialog , is_vista , max_flag }

Protected Member Functions

- settings (bool resync=false)
- bool check_program (std::string const &program)
- bool is_osascript () const
- bool is_zenity () const
- bool is_kdialog () const
- bool const & flags (flag in_flag) const
- bool & flags (flag in_flag)

7.34.1 Detailed Description

Definition at line 121 of file FileDialog.h.

7.34.2 Member Enumeration Documentation

7.34.2.1 flag

```
enum class pfd::settings::flag [strong], [protected]
```

Enumerator

is_scanned	
is_verbose	
has_zenity	
has_matedialog	
has_qarma	
has_kdialog	
is_vista	
max_flag	

Definition at line 138 of file FileDialog.h.

```
        {
            is_scanned = 0,
            is_verbose,

            has_zenity,
            has_matedialog,
            has_qarma,
            has_kdialog,
            is_vista,

            max_flag,
        };
```

7.34.3 Constructor & Destructor Documentation

7.34.3.1 settings()

```
pfd::settings::settings (
            bool resync = false )  [inline], [explicit], [protected]
```

Definition at line 524 of file FileDialog.h.

```
    {
        flags(flag::is_scanned) &= !resync;

        if (flags(flag::is_scanned))
            return;

        auto pfd_verbose = internal::getenv("PFD_VERBOSE");
        auto match_no = std::regex("(|0|no|false)", std::regex_constants::icase);
        if (!std::regex_match(pfd_verbose, match_no))
            flags(flag::is_verbose) = true;

#if _WIN32
        flags(flag::is_vista) = internal::is_vista();
#elif !__APPLE__
        flags(flag::has_zenity) = check_program("zenity");
        flags(flag::has_matedialog) = check_program("matedialog");
        flags(flag::has_qarma) = check_program("qarma");
        flags(flag::has_kdialog) = check_program("kdialog");

        // If multiple helpers are available, try to default to the best one
        if (flags(flag::has_zenity) && flags(flag::has_kdialog))
        {
            auto desktop_name = internal::getenv("XDG_SESSION_DESKTOP");
            if (desktop_name == std::string("gnome"))
                flags(flag::has_kdialog) = false;
            else if (desktop_name == std::string("KDE"))
                flags(flag::has_zenity) = false;
        }
#endif

        flags(flag::is_scanned) = true;
    }
```

7.34.4 Member Function Documentation

7.34.4.1 available()

```
bool pfd::settings::available ( )  [inline], [static]
```

Definition at line 558 of file FileDialog.h.

```
    {
#if _WIN32
        return true;
#elif __APPLE__
        return true;
#elif __EMSCRIPTEN__
        // FIXME: Return true after implementation is complete.
    return false;
#else
        settings tmp;
        return tmp.flags(flag::has_zenity) ||
               tmp.flags(flag::has_matedialog) ||
               tmp.flags(flag::has_qarma) ||
               tmp.flags(flag::has_kdialog);
#endif
    }
```

7.34.4.2 check_program()

```
bool pfd::settings::check_program (
             std::string const & program )  [inline], [protected]
```

Definition at line 587 of file FileDialog.h.

```
    {
#if _WIN32
        (void)program;
    return false;
#elif __EMSCRIPTEN__
        (void)program;
    return false;
#else
        int exit_code = -1;
        internal::executor async;
        async.start_process({"/bin/sh", "-c", "which " + program});
        async.result(&exit_code);
        return exit_code == 0;
#endif
    }
```

7.34.4.3 flags() [1/2]

```
bool & pfd::settings::flags (
             flag in_flag )  [inline], [protected]
```

Definition at line 631 of file FileDialog.h.

```
    {
        return const_cast<bool &>(static_cast<settings const *>(this)->flags(in_flag));
    }
```

7.34.4.4 flags() [2/2]

```
bool const & pfd::settings::flags (
             flag in_flag ) const  [inline], [protected]
```

Definition at line 625 of file FileDialog.h.

```
    {
        static bool flags[size_t(flag::max_flag)];
        return flags[size_t(in_flag)];
    }
```

7.34.4.5 is_kdialog()

```
bool pfd::settings::is_kdialog ( ) const  [inline], [protected]
```

Definition at line 620 of file FileDialog.h.

```
    {
        return flags(flag::has_kdialog);
    }
```

7.34.4.6 is_osascript()

```
bool pfd::settings::is_osascript ( ) const  [inline], [protected]
```

Definition at line 604 of file FileDialog.h.

```
    {
#if __APPLE__
        return true;
#else
        return false;
#endif
    }
```

7.34.4.7 is_zenity()

```
bool pfd::settings::is_zenity ( ) const  [inline], [protected]
```

Definition at line 613 of file FileDialog.h.

```
    {
        return flags(flag::has_zenity) ||
               flags(flag::has_matedialog) ||
               flags(flag::has_qarma);
    }
```

7.34.4.8 rescan()

```
void pfd::settings::rescan ( )  [inline], [static]
```

Definition at line 581 of file FileDialog.h.

```
    {
        settings(/* resync = */ true);
    }
```

7.34.4.9 verbose()

```
void pfd::settings::verbose (
            bool value )  [inline], [static]
```

Definition at line 576 of file FileDialog.h.

```
    {
        settings().flags(flag::is_verbose) = value;
    }
```

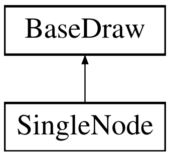
The documentation for this class was generated from the following file:

- include/core/FileDialog.h

7.35 SingleNode Class Reference

```
#include <SingleNode.hpp>
```

Inheritance diagram for SingleNode:



Public Member Functions

- SingleNode (sf::RenderWindow ∗window, std::string value, sf::Vector2f position)
- void render () override
- void toggleActiveColor ()
- void resetColor ()
- void setText (std::string _value)
- void setPosition (sf::Vector2f position)
- sf::Vector2f getPosition ()

Public Member Functions inherited from BaseDraw

- BaseDraw (sf::RenderWindow ∗window)
- virtual void render ()=0

Public Attributes

- sf::Font font

Additional Inherited Members

Protected Attributes inherited from BaseDraw

- sf::RenderWindow ∗ window

7.35.1 Detailed Description

Definition at line 12 of file SingleNode.hpp.

7.35.2 Constructor & Destructor Documentation

7.35.2.1 SingleNode()

```
SingleNode::SingleNode (
              sf::RenderWindow * window,
              std::string value,
              sf::Vector2f position )
```

Definition at line 7 of file SingleNode.cpp.

```
                                                                                        : BaseDraw(window)
     {
    this->value = std::move(value);

    this->circle.setRadius(constants::NodeInfo::radius);
    this->circle.setFillColor(sf::Color::White);
    this->circle.setOutlineThickness(constants::NodeInfo::outlineThickness);
    this->circle.setOutlineColor(sf::Color::Black);
    this->circle.setPointCount(constants::NodeInfo::pointCount);
    sf::FloatRect bounds = this->circle.getLocalBounds();
    this->circle.setOrigin(bounds.left + bounds.width / 2.0f,bounds.top + bounds.height / 2.0f);
    this->circle.setPosition(position);

    this->font.loadFromFile(constants::fontPath);
    this->label.setFont(this->font);
    this->label.setString(this->value);
    this->label.setCharacterSize(constants::NodeInfo::fontSize);
    this->label.setFillColor(sf::Color::Black);
    bounds = this->label.getLocalBounds();
    this->label.setOrigin(bounds.left + bounds.width / 2.0f, bounds.top + bounds.height / 2.0f);
    this->label.setPosition(position);
}
```

7.35.3 Member Function Documentation

7.35.3.1 getPosition()

```
sf::Vector2f SingleNode::getPosition ( )
```

Definition at line 47 of file SingleNode.cpp.

```
                                     {
    return this->circle.getPosition();
}
```

7.35.3.2 render()

```
void SingleNode::render ( )  [override], [virtual]
```

Implements BaseDraw.

Definition at line 29 of file SingleNode.cpp.

```
                        {
    this->window->draw(this->circle);
    this->window->draw(this->label);
}
```

7.35.3.3 resetColor()

```
void SingleNode::resetColor ( )
```

Definition at line 38 of file SingleNode.cpp.

```
00038                          {
00039     this->circle.setOutlineColor(sf::Color::Black);
00040 }
```

7.35.3.4 setPosition()

```
void SingleNode::setPosition (
              sf::Vector2f position )
```

Definition at line 42 of file SingleNode.cpp.

```
00042                                               {
00043     this->circle.setPosition(position);
00044     this->label.setPosition(position);
00045 }
```

7.35.3.5 setText()

```
void SingleNode::setText (
              std::string _value )
```

Definition at line 51 of file SingleNode.cpp.

```
00051                                         {
00052     this->value = std::move(_value);
00053     this->label.setString(this->value);
00054     sf::FloatRect bounds = this->label.getLocalBounds();
00055     this->label.setOrigin(bounds.left + bounds.width / 2.0f, bounds.top + bounds.height / 2.0f);
00056     this->label.setPosition(this->circle.getPosition());
00057 }
```

7.35.3.6 toggleActiveColor()

```
void SingleNode::toggleActiveColor ( )
```

Definition at line 34 of file SingleNode.cpp.

```
00034                                 {
00035     this->circle.setOutlineColor(constants::normalGreen);
00036 }
```

7.35.4 Member Data Documentation

7.35.4.1 font

```
sf::Font SingleNode::font
```

Definition at line 19 of file SingleNode.hpp.

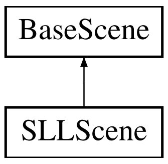
The documentation for this class was generated from the following files:

- include/draw/SingleNode.hpp
- include/draw/SingleNode.cpp

7.36 SLLScene Class Reference

```
#include <SLLScene.hpp>
```

Inheritance diagram for SLLScene:



Public Member Functions

- SLLScene (sf::RenderWindow ∗window)
- void reset ()
- void pollEvent (sf::Event event, sf::Vector2f mousePosView) override
- void update () override
- void render () override
- std::vector< EventAnimation > addModeEvents (int chosenNode)
- std::vector< EventAnimation > deleteModeEvents (int chosenNode)
- std::vector< EventAnimation > updateModeEvents (int chosenNode)
- std::vector< EventAnimation > searchModeEvents (int chosenNode)

Public Member Functions inherited from BaseScene

- BaseScene (sf::RenderWindow ∗window)
- void createModeButton (sf::Vector2f position, std::string textString)
- virtual void pollEvent (sf::Event event, sf::Vector2f mousePosView)=0
- virtual void update ()=0
- virtual void render ()=0

Additional Inherited Members

Public Attributes inherited from BaseScene

- Button ∗ modeButton {}
- bool isMenuOpen {}
- bool isDemoCodeOpen {}

Protected Member Functions inherited from BaseScene

- void setWindow (sf::RenderWindow ∗window)

Protected Attributes inherited from BaseScene

- sf::RenderWindow ∗ window {}
- ControlMenu ∗ controlMenu

7.36.1 Detailed Description

Definition at line 12 of file SLLScene.hpp.

7.36.2 Constructor & Destructor Documentation

7.36.2.1 SLLScene()

```
SLLScene::SLLScene (
              sf::RenderWindow ∗ window )  [explicit]
```

Definition at line 130 of file SLLScene.cpp.

```
                                           : BaseScene(window) {
    this->init();
}
```

7.36.3 Member Function Documentation

7.36.3.1 addModeEvents()

```
std::vector< EventAnimation > SLLScene::addModeEvents (
              int chosenNode )
```

Definition at line 143 of file SLLScene.cpp.

```
                                                               {
    this->linkedList->resetEvents();
    if (chosenNode < 0 || chosenNode > this->linkedList->getSize())
        return {};

    this->linkedList->initHighlighter(
            constants::Highlighter::SLL::CODES_PATH[0].second,
            constants::Highlighter::SLL::CODES_PATH[0].first
    );

    std::vector<EventAnimation> events;
    EventAnimation event;

    if (chosenNode)
        event.titleNodes = {
                {0, "head"},
                {chosenNode, "temp"}
        };
    else {
        event.titleNodes.emplace_back(chosenNode, "temp");
        if (this->linkedList->getSize())
            event.titleNodes.emplace_back(1, "head");
    }
    event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
    if (chosenNode && chosenNode == this->linkedList->getSize())
        event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
    event.colorNodes.push_back(chosenNode);
    event.statusChosenNode = NodeInfo::StatusNode::OutChain;
    event.lines = {0};

    events.emplace_back(event);

    if (chosenNode == 0) {
        if (this->linkedList->getSize()) {
            event.reset();
            event.titleNodes = {
                    {1, "head"},
                    {chosenNode, "temp"}
            };
            event.colorNodes = std::vector<int>{0};
            event.colorArrows.emplace_back(0, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.isPrintNormal = true;
            event.lines = {1, 2};

            events.emplace_back(event);
        }

        event.reset();
        event.titleNodes.emplace_back(chosenNode, "head|temp");
        event.lines = {3};
        event.statusChosenNode = NodeInfo::StatusNode::InChain;
        events.emplace_back(event);
    } else {
        event.reset();
        event.titleNodes = {
                {0, "head|current"},
                {chosenNode, "temp"}
        };
        event.colorNodes.push_back(0);
        event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
        if (chosenNode == this->linkedList->getSize())
            event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
        event.statusChosenNode = NodeInfo::StatusNode::OutChain;
        event.lines = {5};

        events.emplace_back(event);

        for (int i = 0; i < chosenNode; ++i) {
            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {chosenNode, "temp"},
                    {i, "current"}
            };
            event.colorNodes.push_back(i);
            event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
            if (chosenNode == this->linkedList->getSize())
                event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.lines = {6};

            events.emplace_back(event);

            if (i == chosenNode - 1) break;

            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {chosenNode, "temp"},
                    {i, "current"}
            };
            event.colorNodes.push_back(i);
            event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
            event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
            if (chosenNode == this->linkedList->getSize())
                event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.lines = {7};

            events.emplace_back(event);
        }

        if (chosenNode != this->linkedList->getSize()) {
            event.reset();
```

```
00248             event.titleNodes = {
00249                     {0, "head"},
00250                     {chosenNode, "temp"},
00251                     {chosenNode - 1, "current"}
00252             };
00253             event.colorNodes.push_back(chosenNode);
00254             event.colorArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00255             event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00256             event.isPrintNormal = true;
00257             event.lines = {8};
00258 
00259             events.emplace_back(event);
00260         }
00261 
00262         event.reset();
00263         event.titleNodes = {
00264                 {0, "head"},
00265                 {chosenNode, "temp"}
00266         };
00267         event.statusChosenNode = NodeInfo::StatusNode::InChain;
00268         event.lines = {9};
00269 
00270         events.emplace_back(event);
00271     }
00272 
00273     return events;
00274 }
```

7.36.3.2 deleteModeEvents()

```
std::vector< EventAnimation > SLLScene::deleteModeEvents (
               int chosenNode )
```

Definition at line 276 of file SLLScene.cpp.

```
00276                                                                         {
00277     this->linkedList->resetEvents();
00278     if (chosenNode < 0 || chosenNode >= this->linkedList->getSize())
00279         return {};
00280 
00281     this->linkedList->initHighlighter(
00282             constants::Highlighter::SLL::CODES_PATH[1].second,
00283             constants::Highlighter::SLL::CODES_PATH[1].first
00284     );
00285 
00286     std::vector<EventAnimation> events;
00287     EventAnimation event;
00288 
00289     if (!chosenNode) {
00290         event.titleNodes.emplace_back(chosenNode, "head|temp");
00291         event.colorNodes.push_back(chosenNode);
00292         event.statusChosenNode = NodeInfo::StatusNode::InChain;
00293         event.lines = {0, 1};
00294 
00295         events.emplace_back(event);
00296 
00297         if (this->linkedList->getSize() > 1) {
00298             event.reset();
00299             event.titleNodes = {
00300                     {chosenNode, "temp"},
00301                     {1, "head"}
00302             };
00303             event.colorNodes.push_back(1);
00304             event.colorArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00305             event.isPrintNormal = true;
00306             event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00307             event.lines = {2};
00308 
00309             events.emplace_back(event);
00310         }
00311 
00312         event.reset();
00313         event.titleNodes.emplace_back(1, "head");
00314         event.statusChosenNode = NodeInfo::StatusNode::Visible;
00315         event.lines = {3};
00316 
00317         events.emplace_back(event);
00318     } else {
00319         event.reset();
```

```
        event.titleNodes.emplace_back(0, "head|current");
        event.colorNodes.push_back(0);
        event.statusChosenNode = NodeInfo::StatusNode::InChain;
        event.lines = {5};

        events.emplace_back(event);

        for (int i = 0; i < chosenNode; ++i) {
            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {i, "current"}
            };
            event.colorNodes.push_back(i);
            event.statusChosenNode = NodeInfo::StatusNode::InChain;
            event.lines = {6};

            events.emplace_back(event);

            if (i == chosenNode - 1) break;

            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {i, "current"}
            };
            event.colorNodes.push_back(i);
            event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::InChain;
            event.lines = {7};

            events.emplace_back(event);
        }

        event.reset();
        event.titleNodes = {
                {0, "head"},
                {chosenNode, "temp"},
                {chosenNode - 1, "current"}
        };
        event.colorNodes.push_back(chosenNode);
        event.colorArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
        event.statusChosenNode = NodeInfo::StatusNode::InChain;
        event.lines = {8};

        events.emplace_back(event);

        if (chosenNode != this->linkedList->getSize() - 1) {
            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {chosenNode, "temp"},
                    {chosenNode - 1, "current"}
            };
            event.colorNodes.push_back(chosenNode);
            event.colorArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.isPrintNormal = true;
            event.lines = {9};

            events.emplace_back(event);

            event.reset();
            event.titleNodes.emplace_back(0, "head");
            event.statusChosenNode = NodeInfo::StatusNode::Visible;
            event.lines = {10};

            events.emplace_back(event);
        } else {
            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {chosenNode, "temp"},
                    {chosenNode - 1, "current"}
            };
            event.colorNodes.push_back(chosenNode);
            event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.lines = {9};

            events.emplace_back(event);

            event.reset();
            event.titleNodes.emplace_back(0, "head");
            event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::Visible;
            event.lines = {10};
```

```
00407
00408             events.emplace_back(event);
00409         }
00410     }
00411
00412     return events;
00413 }
```

7.36.3.3 pollEvent()

```
void SLLScene::pollEvent (
               sf::Event event,
               sf::Vector2f mousePosView )  [override], [virtual]
```

Implements BaseScene.

Definition at line 7 of file SLLScene.cpp.

```
00007                                                                         {
00008     if (this->isMenuOpen)
00009         this->menu->pollEvents(event, mousePosView);
00010
00011     this->controlMenu->pollEvents(event, mousePosView);
00012 }
```

7.36.3.4 render()

```
void SLLScene::render ( )  [override], [virtual]
```

Implements BaseScene.

Definition at line 119 of file SLLScene.cpp.

```
00119                            {
00120     if (this->isMenuOpen)
00121         this->menu->render();
00122
00123     if (this->isDemoCodeOpen)
00124         this->linkedList->renderHighlighter();
00125
00126     this->controlMenu->render();
00127     this->linkedList->render();
00128 }
```

7.36.3.5 reset()

```
void SLLScene::reset ( )
```

Definition at line 139 of file SLLScene.cpp.

```
00139                           {
00140     this->menu->resetActiveOptionMenu();
00141 }
```

7.36.3.6 searchModeEvents()

```
std::vector< EventAnimation > SLLScene::searchModeEvents (
              int chosenNode )
```

Definition at line 479 of file SLLScene.cpp.

```
00479                                                                    {
00480     this->linkedList->resetEvents();
00481     this->linkedList->initHighlighter(
00482             constants::Highlighter::SLL::CODES_PATH[3].second,
00483             constants::Highlighter::SLL::CODES_PATH[3].first
00484     );
00485 
00486     std::vector<EventAnimation> events;
00487     EventAnimation event;
00488 
00489     event.titleNodes.emplace_back(0, "head|current");
00490     event.colorNodes.push_back(0);
00491     event.lines = {0};
00492 
00493     events.emplace_back(event);
00494 
00495     for (int i = 0; i <= chosenNode; ++i) {
00496         if (i == chosenNode && chosenNode == this->linkedList->getSize())
00497             break;
00498 
00499         event.reset();
00500         event.titleNodes = {
00501                 {0, "head"},
00502                 {i, "current"}
00503         };
00504         event.colorNodes.push_back(i);
00505         event.lines = {1};
00506 
00507         events.emplace_back(event);
00508 
00509         if (i == chosenNode) break;
00510 
00511         event.reset();
00512         event.titleNodes = {
00513                 {0, "head"},
00514                 {i, "current"}
00515         };
00516         event.colorNodes.push_back(i);
00517         event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
00518         event.lines = {4};
00519 
00520         events.emplace_back(event);
00521     }
00522 
00523     if (chosenNode == this->linkedList->getSize()) {
00524         event.reset();
00525         event.titleNodes.emplace_back(0, "head");
00526         event.lines = {5};
00527 
00528         events.emplace_back(event);
00529     } else {
00530         event.reset();
00531         event.titleNodes = {
00532                 {0, "head"},
00533                 {chosenNode, "current"}
00534         };
00535         event.colorNodes.push_back(chosenNode);
00536         event.lines = {2, 3};
00537 
00538         events.emplace_back(event);
00539     }
00540 
00541     return events;
00542 }
```

7.36.3.7 update()

```
void SLLScene::update ( )  [override], [virtual]
```

Implements BaseScene.

Definition at line 14 of file SLLScene.cpp.

```
00014                       {
00015     if (this->isMenuOpen) {
00016         this->menu->update();
00017 
00018         constants::MenuLinkedList::Button status = this->menu->getActiveOptionMenu();
00019         constants::MenuLinkedList::CreateMode::Button createMode;
00020         switch (status){
00021             case constants::MenuLinkedList::Button::CREATE_BUTTON:
00022                 createMode = this->menu->getActiveCreateMode();
00023                 if (createMode == constants::MenuLinkedList::CreateMode::Button::RANDOM_BUTTON) {
00024                     if (this->menu->createModeValue[0] == "None")
00025                         break;
00026                     if (this->menu->createModeValue[0].empty())
00027                         this->menu->createModeValue[0] = "0";
00028                     int size = std::stoi(this->menu->createModeValue[0]);
00029                     this->linkedList->createLinkedList(size);
00030                 } else if (createMode ==
      constants::MenuLinkedList::CreateMode::Button::DEFINED_LIST_BUTTON) {
00031                     if (this->menu->createModeValue[1] == "None")
00032                         break;
00033                     std::vector<std::string> values;
00034                     std::string value = this->menu->createModeValue[1];
00035                     std::stringstream ss(value);
00036                     std::string token;
00037                     while (std::getline(ss, token, ',')) {
00038                         values.push_back(token);
00039                     }
00040                     this->linkedList->createLinkedList(values);
00041                 } else if (createMode == constants::MenuLinkedList::CreateMode::Button::FILE_BUTTON) {
00042                     if (this->menu->createModeValue[2] == "None")
00043                         break;
00044                     std::vector<std::string> values;
00045                     std::string value = this->menu->createModeValue[2];
00046                     std::stringstream ss(value);
00047                     std::string token;
00048                     while (std::getline(ss, token, ','))
00049                         values.push_back(token);
00050                     this->linkedList->createLinkedList(values);
00051                     this->menu->createModeValue[2] = "None";
00052                 }
00053                 this->controlMenu->reset();
00054                 break;
00055             case constants::MenuLinkedList::Button::ADD_BUTTON:
00056                 if (this->menu->addModeValue[0] == "None" || this->menu->addModeValue[1] == "None" ||
      this->menu->addModeValue[0].empty())
00057                     break;
00058 
00059                 this->linkedList->addNode(
00060                         std::stoi(this->menu->addModeValue[0]),
00061                         this->menu->addModeValue[1],
00062                         this->addModeEvents(std::stoi(this->menu->addModeValue[0]))
00063                         );
00064 
00065                 std::cout « "Add: " « this->menu->addModeValue[0] « " " « this->menu->addModeValue[1]
      « std::endl;
00066                 this->menu->addModeValue[0] = this->menu->addModeValue[1] = "None";
00067                 this->controlMenu->reset();
00068                 break;
00069             case constants::MenuLinkedList::Button::DELETE_BUTTON:
00070                 if (this->menu->deleteModeValue == "None" || this->menu->deleteModeValue.empty())
00071                     break;
00072 
00073                 this->linkedList->deleteNode(
00074                         std::stoi(this->menu->deleteModeValue),
00075                         this->deleteModeEvents(std::stoi(this->menu->deleteModeValue))
00076                         );
00077 
00078                 std::cout « "Delete: " « this->menu->deleteModeValue « std::endl;
00079                 this->menu->deleteModeValue = "None";
00080                 this->controlMenu->reset();
00081                 break;
00082             case constants::MenuLinkedList::Button::UPDATE_BUTTON:
00083                 if (this->menu->updateModeValue[0] == "None" || this->menu->updateModeValue[1] ==
      "None" || this->menu->updateModeValue[0].empty())
00084                     break;
00085 
00086                 this->linkedList->updateNode(
00087                         std::stoi(this->menu->updateModeValue[0]),
00088                         this->menu->updateModeValue[1],
00089                         this->updateModeEvents(std::stoi(this->menu->updateModeValue[0]))
00090                         );
00091 
00092                 std::cout « "Update: " « this->menu->updateModeValue[0] « " " «
      this->menu->updateModeValue[1] « std::endl;
00093                 this->menu->updateModeValue[0] = this->menu->updateModeValue[1] = "None";
00094                 this->controlMenu->reset();
00095                 break;
00096             case constants::MenuLinkedList::Button::SEARCH_BUTTON:
00097                 if (this->menu->searchModeValue == "None" || this->menu->searchModeValue.empty())
00098                     break;
00099 
00100                 this->linkedList->searchNode(
00101 
      this->searchModeEvents(this->linkedList->findValue(this->menu->searchModeValue))
00102                         );
00103 
00104                 std::cout « "Search: " « this->menu->searchModeValue « std::endl;
00105                 this->menu->searchModeValue = "None";
00106                 this->controlMenu->reset();
00107                 break;
00108         }
00109     }
00110 
00111     this->controlMenu->update();
00112 
00113     this->linkedList->processControlMenu(this->controlMenu->getStatus());
00114     this->linkedList->setSpeed(this->controlMenu->getSpeed());
00115 
00116     this->linkedList->update();
00117 }
```

7.36.3.8 updateModeEvents()

```
std::vector< EventAnimation > SLLScene::updateModeEvents (
               int chosenNode )
```

Definition at line 415 of file SLLScene.cpp.

```
00415                                                                   {
00416     this->linkedList->resetEvents();
00417     if (chosenNode < 0 || chosenNode >= this->linkedList->getSize())
00418         return {};
00419 
00420     this->linkedList->initHighlighter(
00421             constants::Highlighter::SLL::CODES_PATH[2].second,
00422             constants::Highlighter::SLL::CODES_PATH[2].first
00423     );
00424 
00425     std::vector<EventAnimation> events;
00426     EventAnimation event;
00427 
00428     event.titleNodes.emplace_back(0, "head|current");
00429     event.colorNodes.push_back(0);
00430     event.isPrintPreVal = true;
00431     event.lines = {0};
00432 
00433     events.emplace_back(event);
00434 
00435     if (chosenNode) {
00436         for (int i = 0; i <= chosenNode; ++i) {
00437             event.reset();
00438             event.titleNodes = {
00439                     {0, "head"},
00440                     {i, "current"}
00441             };
00442             event.colorNodes.push_back(i);
00443             event.isPrintPreVal = true;
00444             event.lines = {1};
00445 
00446             events.emplace_back(event);
00447 
00448             if (i == chosenNode) break;
00449 
00450             event.reset();
00451             event.titleNodes = {
00452                     {0, "head"},
00453                     {i, "current"}
00454             };
00455             event.colorNodes.push_back(i);
00456             event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
00457             event.isPrintPreVal = true;
00458             event.lines = {2};
00459 
00460             events.emplace_back(event);
00461         }
```

```
    }

    event.reset();
    if (chosenNode == 0)
        event.titleNodes.emplace_back(0, "head|current");
    else
        event.titleNodes = {
                {0, "head"},
                {chosenNode, "current"}
        };
    event.lines = {3};

    events.emplace_back(event);

    return events;
}
```

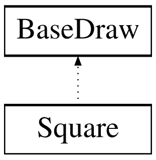
The documentation for this class was generated from the following files:

- include/libScene/SLLScene.hpp
- include/libScene/SLLScene.cpp

7.37 Square Class Reference

`#include <Square.hpp>`

Inheritance diagram for Square:



Public Types

- enum class Status { inactive , active , chosen , hidden }

Public Member Functions

- Square (sf::RenderWindow ∗window, std::string value, sf::Vector2f position)
- void render () override
- void setStatus (Status _status)
- void resetColor ()
- Status getStatus ()
- void setText (std::string _value)
- void setPosition (sf::Vector2f position)
- sf::Vector2f getPosition ()

Public Attributes

- sf::Font font

7.37.1 Detailed Description

Definition at line 11 of file Square.hpp.

7.37.2 Member Enumeration Documentation

7.37.2.1 Status

```
enum class Square::Status  [strong]
```

Enumerator

inactive	
active	
chosen	
hidden	

Definition at line 13 of file Square.hpp.

```
                         {
        inactive,
        active,
        chosen,
        hidden
    };
```

7.37.3 Constructor & Destructor Documentation

7.37.3.1 Square()

```
Square::Square (
            sf::RenderWindow * window,
            std::string value,
            sf::Vector2f position )
```

Definition at line 7 of file Square.cpp.

```
        : BaseDraw(window) {
    this->value = std::move(value);

    this->square.setSize(sf::Vector2f(constants::Square::length, constants::Square::length));
    this->square.setFillColor(sf::Color::White);
    this->square.setOutlineThickness(constants::Square::outlineThickness);
    this->square.setOutlineColor(sf::Color::Black);
    sf::FloatRect bounds = this->square.getLocalBounds();
    this->square.setOrigin(bounds.left + bounds.width / 2.0f,bounds.top + bounds.height / 2.0f);
    this->square.setPosition(position);

    this->font.loadFromFile(constants::fontPath);
    this->label.setFont(this->font);
    this->label.setString(this->value);
    this->label.setCharacterSize(constants::Square::fontSize);
    this->label.setFillColor(sf::Color::Black);
    bounds = this->label.getLocalBounds();
    this->label.setOrigin(bounds.left + bounds.width / 2.0f, bounds.top + bounds.height / 2.0f);
    this->label.setPosition(position);
}
```

7.37.4 Member Function Documentation

7.37.4.1 getPosition()

```
sf::Vector2f Square::getPosition ( )
```

Definition at line 62 of file Square.cpp.

```
                                {
    return this->square.getPosition();
}
```

7.37.4.2 getStatus()

```
Square::Status Square::getStatus ( )
```

Definition at line 70 of file Square.cpp.

```
                                {
    return this->status;
}
```

7.37.4.3 render()

```
void Square::render ( )  [override], [virtual]
```

Implements BaseDraw.

Definition at line 29 of file Square.cpp.

```
                         {
    switch (this->status) {
        case Status::active:
            this->square.setOutlineColor(constants::normalGreen);
            break;
        case Status::inactive:
            this->square.setOutlineColor(sf::Color::Black);
            break;
        case Status::chosen:
            this->square.setOutlineColor(constants::clickGreen);
            break;
    }
    this->window->draw(this->square);
    this->window->draw(this->label);
}
```

7.37.4.4 resetColor()

```
void Square::resetColor ( )
```

Definition at line 45 of file Square.cpp.

```
                             {
    this->status = Status::inactive;
}
```

7.37.4.5 setPosition()

```
void Square::setPosition (
            sf::Vector2f position )
```

Definition at line 57 of file Square.cpp.

```
00057                                         {
00058     this->square.setPosition(position);
00059     this->label.setPosition(position);
00060 }
```

7.37.4.6 setStatus()

```
void Square::setStatus (
            Square::Status _status )
```

Definition at line 66 of file Square.cpp.

```
00066                                        {
00067     this->status = _status;
00068 }
```

7.37.4.7 setText()

```
void Square::setText (
            std::string _value )
```

Definition at line 49 of file Square.cpp.

```
00049                                   {
00050     this->value = std::move(_value);
00051     this->label.setString(this->value);
00052     sf::FloatRect bounds = this->label.getLocalBounds();
00053     this->label.setOrigin(bounds.left + bounds.width / 2.0f, bounds.top + bounds.height / 2.0f);
00054     this->label.setPosition(this->square.getPosition());
00055 }
```

7.37.5 Member Data Documentation

7.37.5.1 font

```
sf::Font Square::font
```

Definition at line 20 of file Square.hpp.

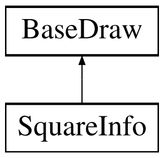
The documentation for this class was generated from the following files:

- include/draw/Square.hpp
- include/draw/Square.cpp

7.38 SquareInfo Class Reference

```
#include <SquareInfo.hpp>
```

Inheritance diagram for SquareInfo:



Public Member Functions

- SquareInfo (sf::RenderWindow ∗window, std::string value, sf::Vector2f position)
- ∼SquareInfo ()=default
- void update ()
- void render () override
- void setValue (std::string value)
- void setTitle (const std::string &_title)
- void setStatus (Square::Status _status)
- void setPrintPreVal (bool _isPrintPreVal)
- std::string getValue ()
- Square::Status getStatus ()
- void resetTitle ()
- void reset ()

Public Member Functions inherited from BaseDraw

- BaseDraw (sf::RenderWindow ∗window)
- virtual void render ()=0

Additional Inherited Members

Protected Attributes inherited from BaseDraw

- sf::RenderWindow ∗ window

7.38.1 Detailed Description

Definition at line 10 of file SquareInfo.hpp.

7.38.2 Constructor & Destructor Documentation

7.38.2.1 SquareInfo()

```
SquareInfo::SquareInfo (
            sf::RenderWindow * window,
            std::string value,
            sf::Vector2f position )
```

Definition at line 7 of file SquareInfo.cpp.

```
                                                                    : BaseDraw(window)
{
    this->position = position;
    this->square = new Square(window, value, position);
    this->values[0] = std::move(value);
    this->values[1] = "";
    this->isPrintPreVal = false;

    this->title.setFont(this->square->font);
    this->title.setCharacterSize(20);
    this->title.setFillColor(sf::Color::Black);
}
```

7.38.2.2 ∼SquareInfo()

```
SquareInfo::∼SquareInfo ( )  [default]
```

7.38.3 Member Function Documentation

7.38.3.1 getStatus()

```
Square::Status SquareInfo::getStatus ( )
```

Definition at line 67 of file SquareInfo.cpp.

```
                                   {
    return this->square->getStatus();
}
```

7.38.3.2 getValue()

```
std::string SquareInfo::getValue ( )
```

Definition at line 59 of file SquareInfo.cpp.

```
                               {
    return this->values[0];
}
```

7.38.3.3 render()

```
void SquareInfo::render ( )  [override], [virtual]
```

Implements BaseDraw.

Definition at line 19 of file SquareInfo.cpp.

```
                                {
    if (this->square->getStatus() != Square::Status::hidden) {
        this->square->render();
        this->window->draw(this->title);
    }
}
```

7.38.3.4 reset()

```
void SquareInfo::reset ( )
```

Definition at line 49 of file SquareInfo.cpp.

```
                               {
    this->resetTitle();
    this->square->resetColor();
    this->isPrintPreVal = false;
}
```

7.38.3.5 resetTitle()

```
void SquareInfo::resetTitle ( )
```

Definition at line 45 of file SquareInfo.cpp.

```
                                    {
    this->title.setString("");
}
```

7.38.3.6 setPrintPreVal()

```
void SquareInfo::setPrintPreVal (
            bool _isPrintPreVal )
```

Definition at line 63 of file SquareInfo.cpp.

```
                                                    {
    this->isPrintPreVal = _isPrintPreVal;
}
```

7.38.3.7 setStatus()

```
void SquareInfo::setStatus (
            Square::Status _status )
```

Definition at line 55 of file SquareInfo.cpp.

```
                                                {
    this->square->setStatus(_status);
}
```

7.38.3.8 setTitle()

```
void SquareInfo::setTitle (
            const std::string & _title )
```

Definition at line 38 of file SquareInfo.cpp.

```
                                                  {
    this->title.setString(_title);
    sf::FloatRect bounds = this->title.getLocalBounds();
    this->title.setOrigin(bounds.left + bounds.width / 2.0f, bounds.top + bounds.height / 2.0f);
    this->title.setPosition(this->position.x, this->position.y + constants::TitleNode::offsetY);
}
```

7.38.3.9 setValue()

```
void SquareInfo::setValue (
            std::string value )
```

Definition at line 26 of file SquareInfo.cpp.

```
                                          {
    this->values[1] = this->values[0];
    this->values[0] = std::move(value);
}
```

7.38.3.10 update()

```
void SquareInfo::update ( )
```

Definition at line 31 of file SquareInfo.cpp.

```
                          {
    if (this->isPrintPreVal)
        this->square->setText(this->values[1]);
    else
        this->square->setText(this->values[0]);
}
```

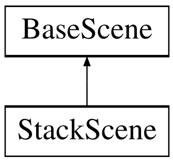
The documentation for this class was generated from the following files:

- include/draw/SquareInfo.hpp
- include/draw/SquareInfo.cpp

7.39 StackScene Class Reference

`#include <StackScene.hpp>`

Inheritance diagram for StackScene:



Public Member Functions

- StackScene (sf::RenderWindow ∗window)
- void reset ()
- void pollEvent (sf::Event event, sf::Vector2f mousePosView) override
- void update () override
- void render () override
- std::vector< EventAnimation > pushModeEvents (int chosenNode)
- std::vector< EventAnimation > popModeEvents (int chosenNode)

Public Member Functions inherited from BaseScene

- BaseScene (sf::RenderWindow ∗window)
- void createModeButton (sf::Vector2f position, std::string textString)
- virtual void pollEvent (sf::Event event, sf::Vector2f mousePosView)=0
- virtual void update ()=0
- virtual void render ()=0

Additional Inherited Members

Public Attributes inherited from BaseScene

- Button ∗ modeButton {}
- bool isMenuOpen {}
- bool isDemoCodeOpen {}

Protected Member Functions inherited from BaseScene

- void setWindow (sf::RenderWindow ∗window)

Protected Attributes inherited from BaseScene

- sf::RenderWindow ∗ window {}
- ControlMenu ∗ controlMenu

7.39.1 Detailed Description

Definition at line 12 of file StackScene.hpp.

7.39.2 Constructor & Destructor Documentation

7.39.2.1 StackScene()

```
StackScene::StackScene (
            sf::RenderWindow * window ) [explicit]
```

Definition at line 7 of file StackScene.cpp.

```
                                                    : BaseScene(window) {
    this->init();
}
```

7.39.3 Member Function Documentation

7.39.3.1 pollEvent()

```
void StackScene::pollEvent (
            sf::Event event,
            sf::Vector2f mousePosView ) [override], [virtual]
```

Implements BaseScene.

Definition at line 111 of file StackScene.cpp.

```
                                                                      {
    if (this->isMenuOpen)
        this->menu->pollEvents(event, mousePosView);

    this->controlMenu->pollEvents(event, mousePosView);
}
```

7.39.3.2 popModeEvents()

```
std::vector< EventAnimation > StackScene::popModeEvents (
            int chosenNode )
```

Definition at line 260 of file StackScene.cpp.

```
                                                                            {
          this->linkedList->resetEvents();
          if (chosenNode < 0 || chosenNode >= this->linkedList->getSize())
              return {};

          this->linkedList->initHighlighter(
                  constants::Highlighter::SLL::CODES_PATH[1].second,
                  constants::Highlighter::SLL::CODES_PATH[1].first
          );

          std::vector<EventAnimation> events;
          EventAnimation event;

          if (!chosenNode) {
              event.titleNodes.emplace_back(chosenNode, "head|temp");
              event.colorNodes.push_back(chosenNode);
              event.statusChosenNode = NodeInfo::StatusNode::InChain;
              event.lines = {0, 1};

              events.emplace_back(event);

              if (this->linkedList->getSize() > 1) {
                  event.reset();
                  event.titleNodes = {
                          {chosenNode, "temp"},
                          {1, "head"}
                  };
                  event.colorNodes.push_back(1);
                  event.colorArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
                  event.isPrintNormal = true;
                  event.statusChosenNode = NodeInfo::StatusNode::OutChain;
                  event.lines = {2};

                  events.emplace_back(event);
              }

              event.reset();
              event.titleNodes.emplace_back(1, "head");
              event.statusChosenNode = NodeInfo::StatusNode::Visible;
              event.lines = {3};

              events.emplace_back(event);
          } else {
              event.reset();
              event.titleNodes.emplace_back(0, "head|current");
              event.colorNodes.push_back(0);
              event.statusChosenNode = NodeInfo::StatusNode::InChain;
              event.lines = {5};

              events.emplace_back(event);

              for (int i = 0; i < chosenNode; ++i) {
                  event.reset();
                  event.titleNodes = {
                          {0, "head"},
                          {i, "current"}
                  };
                  event.colorNodes.push_back(i);
                  event.statusChosenNode = NodeInfo::StatusNode::InChain;
                  event.lines = {6};

                  events.emplace_back(event);

                  if (i == chosenNode - 1) break;

                  event.reset();
                  event.titleNodes = {
                          {0, "head"},
                          {i, "current"}
                  };
                  event.colorNodes.push_back(i);
                  event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
                  event.statusChosenNode = NodeInfo::StatusNode::InChain;
                  event.lines = {7};

                  events.emplace_back(event);
              }

```

```
        event.reset();
        event.titleNodes = {
                {0, "head"},
                {chosenNode, "temp"},
                {chosenNode - 1, "current"}
        };
        event.colorNodes.push_back(chosenNode);
        event.colorArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
        event.statusChosenNode = NodeInfo::StatusNode::InChain;
        event.lines = {8};

        events.emplace_back(event);

        if (chosenNode != this->linkedList->getSize() - 1) {
            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {chosenNode, "temp"},
                    {chosenNode - 1, "current"}
            };
            event.colorNodes.push_back(chosenNode);
            event.colorArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.isPrintNormal = true;
            event.lines = {9};

            events.emplace_back(event);

            event.reset();
            event.titleNodes.emplace_back(0, "head");
            event.statusChosenNode = NodeInfo::StatusNode::Visible;
            event.lines = {10};

            events.emplace_back(event);
        } else {
            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {chosenNode, "temp"},
                    {chosenNode - 1, "current"}
            };
            event.colorNodes.push_back(chosenNode);
            event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.lines = {9};

            events.emplace_back(event);

            event.reset();
            event.titleNodes.emplace_back(0, "head");
            event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::Visible;
            event.lines = {10};

            events.emplace_back(event);
        }
    }

    return events;
}
```

7.39.3.3 pushModeEvents()

```
std::vector< EventAnimation > StackScene::pushModeEvents (
            int chosenNode )
```

Definition at line 127 of file StackScene.cpp.

```
                                                                    {
    this->linkedList->resetEvents();
    if (chosenNode < 0 || chosenNode > this->linkedList->getSize())
        return {};

    this->linkedList->initHighlighter(
            constants::Highlighter::SLL::CODES_PATH[0].second,
            constants::Highlighter::SLL::CODES_PATH[0].first
    );

    std::vector<EventAnimation> events;
    EventAnimation event;

    if (chosenNode)
        event.titleNodes = {
                {0, "head"},
                {chosenNode, "temp"}
        };
    else {
        event.titleNodes.emplace_back(chosenNode, "temp");
        if (this->linkedList->getSize())
            event.titleNodes.emplace_back(1, "head");
    }
    event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
    if (chosenNode && chosenNode == this->linkedList->getSize())
        event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
    event.colorNodes.push_back(chosenNode);
    event.statusChosenNode = NodeInfo::StatusNode::OutChain;
    event.lines = {0};

    events.emplace_back(event);

    if (chosenNode == 0) {
        if (this->linkedList->getSize()) {
            event.reset();
            event.titleNodes = {
                    {1, "head"},
                    {chosenNode, "temp"}
            };
            event.colorNodes = std::vector<int>{0};
            event.colorArrows.emplace_back(0, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.isPrintNormal = true;
            event.lines = {1, 2};

            events.emplace_back(event);
        }

        event.reset();
        event.titleNodes.emplace_back(chosenNode, "head|temp");
        event.lines = {3};
        event.statusChosenNode = NodeInfo::StatusNode::InChain;
        events.emplace_back(event);
    } else {
        event.reset();
        event.titleNodes = {
                {0, "head|current"},
                {chosenNode, "temp"}
        };
        event.colorNodes.push_back(0);
        event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
        if (chosenNode == this->linkedList->getSize())
            event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
        event.statusChosenNode = NodeInfo::StatusNode::OutChain;
        event.lines = {5};

        events.emplace_back(event);

        for (int i = 0; i < chosenNode; ++i) {
            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {chosenNode, "temp"},
                    {i, "current"}
            };
            event.colorNodes.push_back(i);
            event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
            if (chosenNode == this->linkedList->getSize())
                event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.lines = {6};

            events.emplace_back(event);

            if (i == chosenNode - 1) break;

            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {chosenNode, "temp"},
                    {i, "current"}
            };
            event.colorNodes.push_back(i);
            event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
            event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
            if (chosenNode == this->linkedList->getSize())
                event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.lines = {7};

            events.emplace_back(event);
        }

        if (chosenNode != this->linkedList->getSize()) {
            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {chosenNode, "temp"},
                    {chosenNode - 1, "current"}
            };
            event.colorNodes.push_back(chosenNode);
            event.colorArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.isPrintNormal = true;
            event.lines = {8};

            events.emplace_back(event);
        }

        event.reset();
        event.titleNodes = {
                {0, "head"},
                {chosenNode, "temp"}
        };
        event.statusChosenNode = NodeInfo::StatusNode::InChain;
        event.lines = {9};

        events.emplace_back(event);
    }

    return events;
}
```

7.39.3.4 render()

```
void StackScene::render ( )  [override], [virtual]
```

Implements BaseScene.

Definition at line 100 of file StackScene.cpp.

```
                                     {
    if (this->isMenuOpen)
        this->menu->render();

    if (this->isDemoCodeOpen)
        this->linkedList->renderHighlighter();

    this->controlMenu->render();
    this->linkedList->render();
}
```

7.39.3.5 reset()

```
void StackScene::reset ( )
```

Definition at line 123 of file StackScene.cpp.

```
                                    {
    this->menu->resetActiveOptionMenu();
}
```

7.39.3.6 update()

```
void StackScene::update ( )  [override], [virtual]
```

Implements BaseScene.

Definition at line 11 of file StackScene.cpp.

```
00011                         {
00012     if (this->isMenuOpen) {
00013         this->menu->update();
00014 
00015         constants::MenuDataStructure::Button status = this->menu->getActiveOptionMenu();
00016         constants::MenuDataStructure::CreateMode::Button createMode;
00017         switch (status) {
00018             case constants::MenuDataStructure::Button::CREATE_BUTTON:
00019                 createMode = this->menu->getActiveCreateMode();
00020                 if (createMode == constants::MenuDataStructure::CreateMode::Button::RANDOM_BUTTON) {
00021                     if (this->menu->createModeValue[0] == "None")
00022                         break;
00023                     if (this->menu->createModeValue[0].empty())
00024                         this->menu->createModeValue[0] = "0";
00025                     int size = std::stoi(this->menu->createModeValue[0]);
00026                     this->linkedList->createLinkedList(size);
00027                 } else if (createMode ==
      constants::MenuDataStructure::CreateMode::Button::DEFINED_LIST_BUTTON) {
00028                     if (this->menu->createModeValue[1] == "None")
00029                         break;
00030                     std::vector<std::string> values;
00031                     std::string value = this->menu->createModeValue[1];
00032                     std::stringstream ss(value);
00033                     std::string token;
00034                     while (std::getline(ss, token, ',')) {
00035                         values.push_back(token);
00036                     }
00037                     this->linkedList->createLinkedList(values);
00038                 } else if (createMode ==
      constants::MenuDataStructure::CreateMode::Button::FILE_BUTTON) {
00039                     if (this->menu->createModeValue[2] == "None")
00040                         break;
00041                     std::vector<std::string> values;
00042                     std::string value = this->menu->createModeValue[2];
00043                     std::stringstream ss(value);
00044                     std::string token;
00045                     while (std::getline(ss, token, ','))
00046                         values.push_back(token);
00047                     this->linkedList->createLinkedList(values);
00048                     this->menu->createModeValue[2] = "None";
00049                 }
00050                 this->controlMenu->reset();
00051                 break;
00052             case constants::MenuDataStructure::Button::PUSH_BUTTON:
00053                 if (this->menu->pushModeValue == "None")
00054                     break;
00055 
00056                 this->linkedList->addNode(
00057                         0,
00058                         this->menu->pushModeValue,
00059                         this->pushModeEvents(0)
00060                         );
00061 
00062                 std::cout « "Pushed " « this->menu->pushModeValue « std::endl;
00063                 this->menu->pushModeValue = "None";
00064                 this->controlMenu->reset();
00065                 break;
00066             case constants::MenuDataStructure::Button::POP_BUTTON:
00067                 if (this->menu->getActiveOptionMenu() !=
      constants::MenuDataStructure::Button::POP_BUTTON)
00068                     break;
00069 
00070                 this->linkedList->deleteNode(
00071                         0,
00072                         this->popModeEvents(0)
00073                         );
00074 
00075                 std::cout « "Popped " « std::endl;
00076                 this->menu->resetActiveOptionMenuOnly();
00077                 this->controlMenu->reset();
00078                 break;
00079             case constants::MenuDataStructure::Button::CLEAR_BUTTON:
00080                 if (this->menu->getActiveOptionMenu() !=
      constants::MenuDataStructure::Button::CLEAR_BUTTON)
00081                     break;
00082 
```

```
                this->linkedList->createLinkedList(0);

                std::cout « "Cleared " « std::endl;
                this->menu->resetActiveOptionMenuOnly();
                this->controlMenu->reset();
                break;
            }
        }

        this->controlMenu->update();

        this->linkedList->processControlMenu(this->controlMenu->getStatus());
        this->linkedList->setSpeed(this->controlMenu->getSpeed());

        this->linkedList->update();
}
```

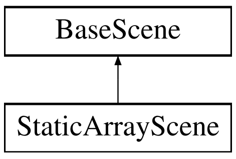
The documentation for this class was generated from the following files:

- include/libScene/StackScene.hpp
- include/libScene/StackScene.cpp

7.40 StaticArrayScene Class Reference

```
#include <StaticArrayScene.hpp>
```

Inheritance diagram for StaticArrayScene:



Public Member Functions

- StaticArrayScene (sf::RenderWindow ∗window)
- void reset ()
- void pollEvent (sf::Event event, sf::Vector2f mousePosView) override
- void update () override
- void render () override
- std::vector< EventAnimation > addModeEvents (int chosenNode)
- std::vector< EventAnimation > deleteModeEvents (int chosenNode)
- std::vector< EventAnimation > updateModeEvents (int chosenNode)
- std::vector< EventAnimation > searchModeEvents (int chosenNode)

Public Member Functions inherited from BaseScene

- BaseScene (sf::RenderWindow ∗window)
- void createModeButton (sf::Vector2f position, std::string textString)
- virtual void pollEvent (sf::Event event, sf::Vector2f mousePosView)=0
- virtual void update ()=0
- virtual void render ()=0

Additional Inherited Members

Public Attributes inherited from BaseScene

- Button ∗ modeButton {}
- bool isMenuOpen {}
- bool isDemoCodeOpen {}

Protected Member Functions inherited from BaseScene

- void setWindow (sf::RenderWindow ∗window)

Protected Attributes inherited from BaseScene

- sf::RenderWindow ∗ window {}
- ControlMenu ∗ controlMenu

7.40.1 Detailed Description

Definition at line 12 of file StaticArrayScene.hpp.

7.40.2 Constructor & Destructor Documentation

7.40.2.1 StaticArrayScene()

```
StaticArrayScene::StaticArrayScene (
            sf::RenderWindow * window )  [explicit]
```

Definition at line 7 of file StaticArrayScene.cpp.

```
                                                           : BaseScene(window) {
    this->init();
}
```

7.40.3 Member Function Documentation

7.40.3.1 addModeEvents()

```
std::vector< EventAnimation > StaticArrayScene::addModeEvents (
                int chosenNode )
```

Definition at line 143 of file StaticArrayScene.cpp.

```
                                                                                    {
    this->array->resetEvents();
    if (chosenNode < 0 || chosenNode > this->array->getSize())
        return {};

    // init highlighter
    // ...

    int size = this->array->getSize() + 1,
            squaresSize = this->array->getSquaresSize();
    if (size > squaresSize) --size;
    if (!squaresSize) return {};

    std::vector<EventAnimation> events;
    EventAnimation event;

    if (size < squaresSize) {
        event = EventAnimation();
        event.eventSquares.assign(squaresSize, EventSquare());
        for (auto &square: event.eventSquares) {
            square.status = Square::Status::active;
            square.isPrintPreVal = true;
        }
        for (int i = size - 1; i < squaresSize; ++i)
            event.eventSquares[i].status = Square::Status::inactive;
        if (size > 1)
            event.eventSquares[size - 2].title = "n";

        events.emplace_back(event);

        event = EventAnimation();
        event.eventSquares.assign(squaresSize, EventSquare());
        for (auto &square : event.eventSquares) {
            square.status = Square::Status::active;
            square.isPrintPreVal = true;
        }
        for (int i = size; i < squaresSize; ++i)
            event.eventSquares[i].status = Square::Status::inactive;
        event.eventSquares[size - 1].title = "n";

        events.emplace_back(event);
    }

    for (int i = size - 1; i >= chosenNode; --i) {
        event = EventAnimation();
        event.eventSquares.assign(squaresSize, EventSquare());
        for (auto &square: event.eventSquares) {
            square.status = Square::Status::active;
            square.isPrintPreVal = true;
        }
        for (int j = size; j < squaresSize; ++j)
            event.eventSquares[j].status = Square::Status::inactive;
        event.eventSquares[size - 1].title = "n";
        for (int j = size - 1; j > i; --j)
            event.eventSquares[j].isPrintPreVal = false;
        event.eventSquares[i].status = Square::Status::chosen;

        events.emplace_back(event);

        event.eventSquares[i].isPrintPreVal = false;
        if (i > chosenNode)
            event.eventSquares[i - 1].status = Square::Status::chosen;

        events.emplace_back(event);
    }

    return events;
}
```

7.40.3.2 deleteModeEvents()

```
std::vector< EventAnimation > StaticArrayScene::deleteModeEvents (
                int chosenNode )
```

Definition at line 212 of file StaticArrayScene.cpp.

```
00212                                                                     {
00213     this->array->resetEvents();
00214     if (chosenNode < 0 || chosenNode >= this->array->getSize())
00215         return {};
00216 
00217     // init highlighter
00218     // ...
00219 
00220     int size = this->array->getSize(),
00221             squaresSize = this->array->getSquaresSize();
00222     std::vector<EventAnimation> events;
00223 //    events.reserve(100);
00224     EventAnimation event;
00225 
00226     for (int i = chosenNode; i < size - 1; ++i) {
00227         event = EventAnimation();
00228         event.eventSquares.assign(squaresSize, EventSquare());
00229         for (auto &square : event.eventSquares) {
00230             square.status = Square::Status::active;
00231             square.isPrintPreVal = true;
00232         }
00233         for (int j = size; j < squaresSize; ++j)
00234             event.eventSquares[j].status = Square::Status::inactive;
00235         for (int j = 0; j < i; ++j)
00236             event.eventSquares[j].isPrintPreVal = false;
00237         event.eventSquares[i].status = Square::Status::chosen;
00238         for (auto &square : event.eventSquaresTemp)
00239             square.status = Square::Status::hidden;
00240         event.eventSquares[size - 1].title = "n";
00241 
00242         events.emplace_back(event);
00243 
00244         event.eventSquares[i].isPrintPreVal = false;
00245         event.eventSquares[i + 1].status = Square::Status::chosen;
00246 
00247         events.emplace_back(event);
00248     }
00249 
00250     event = EventAnimation();
00251     event.eventSquares.assign(squaresSize, EventSquare());
00252     for (int i = 0; i < size - 1; ++i) {
00253         event.eventSquares[i].status = Square::Status::active;
00254         if (i == size - 2)
00255             event.eventSquares[i].title = "n";
00256     }
00257     for (int i = size - 1; i < squaresSize; ++i)
00258         event.eventSquares[i].status = Square::Status::inactive;
00259 
00260     events.emplace_back(event);
00261 
00262     return events;
00263 }
```

7.40.3.3 pollEvent()

```
void StaticArrayScene::pollEvent (
                sf::Event event,
                sf::Vector2f mousePosView )  [override], [virtual]
```

Implements BaseScene.

Definition at line 127 of file StaticArrayScene.cpp.

```
00127                                                                  {
00128     if (this->isMenuOpen)
00129         this->menu->pollEvents(event, mousePosView);
00130 
00131     this->controlMenu->pollEvents(event, mousePosView);
00132 }
```

7.40.3.4 render()

```
void StaticArrayScene::render ( )  [override], [virtual]
```

Implements BaseScene.

Definition at line 116 of file StaticArrayScene.cpp.

```
                              {
    if (this->isMenuOpen)
        this->menu->render();

    if (this->isDemoCodeOpen)
        this->array->renderHighlighter();

    this->controlMenu->render();
    this->array->render();
}
```

7.40.3.5 reset()

```
void StaticArrayScene::reset ( )
```

Definition at line 139 of file StaticArrayScene.cpp.

```
                             {
    this->menu->resetActiveOptionMenu();
}
```

7.40.3.6 searchModeEvents()

```
std::vector< EventAnimation > StaticArrayScene::searchModeEvents (
               int chosenNode )
```

Definition at line 295 of file StaticArrayScene.cpp.

```
                                                                          {
    this->array->resetEvents();

    // init highlighter
    // ...

    int size = this->array->getSize(),
            squaresSize = this->array->getSquaresSize();
    std::vector<EventAnimation> events;
    EventAnimation event;

    for (int i = 0; i <= chosenNode; ++i) {
        if (i == size) break;

        event = EventAnimation();
        event.eventSquares.assign(squaresSize, EventSquare());
        for (int j = 0; j < size; ++j) {
            event.eventSquares[j].status = Square::Status::active;
            if (j == size - 1)
                event.eventSquares[size - 1].title = "n";
        }
        event.eventSquares[i].status = Square::Status::chosen;

        events.emplace_back(event);
    }

    if (chosenNode == size) {
        event = EventAnimation();
        event.eventSquares.assign(squaresSize, EventSquare());
        for (int j = 0; j < size; ++j) {
            event.eventSquares[j].status = Square::Status::active;
            if (j == size - 1)
                event.eventSquares[size - 1].title = "n";
        }

        events.emplace_back(event);
    }

    return events;
}
```

7.40.3.7 update()

```
void StaticArrayScene::update ( )  [override], [virtual]
```

Implements BaseScene.

Definition at line 11 of file StaticArrayScene.cpp.

```
               {
    if (this->isMenuOpen) {
        this->menu->update();

        constants::MenuArray::Button status = this->menu->getActiveOptionMenu();
        constants::MenuArray::CreateMode::Button createMode;
        switch (status){
            case constants::MenuArray::Button::CREATE_BUTTON:
                createMode = this->menu->getActiveCreateMode();
                if (createMode == constants::MenuArray::CreateMode::Button::RANDOM_BUTTON) {
                    if (this->menu->createModeValue[0] == "None")
                        break;
                    if (this->menu->createModeValue[0].empty())
                        this->menu->createModeValue[0] = "0";
                    int size = std::stoi(this->menu->createModeValue[0]);
                    this->array->createArray(size);
                } else if (createMode == constants::MenuArray::CreateMode::Button::DEFINED_LIST_BUTTON) {
                    if (this->menu->createModeValue[1] == "None")
                        break;
                    std::vector<std::string> values;
                    std::string value = this->menu->createModeValue[1];
                    std::stringstream ss(value);
                    std::string token;
                    while (std::getline(ss, token, ',')) {
                        values.push_back(token);
                    }
                    this->array->createArray(values);
                } else if (createMode == constants::MenuArray::CreateMode::Button::FILE_BUTTON) {
                    if (this->menu->createModeValue[2] == "None")
                        break;
                    std::vector<std::string> values;
                    std::string value = this->menu->createModeValue[2];
                    std::stringstream ss(value);
                    std::string token;
                    while (std::getline(ss, token, ','))
                        values.push_back(token);
                    this->array->createArray(values);
                    this->menu->createModeValue[2] = "None";
                }
                this->controlMenu->reset();
                break;
            case constants::MenuArray::Button::ADD_BUTTON:
                if (this->menu->addModeValue[0] == "None" || this->menu->addModeValue[1] == "None" || this->menu->addModeValue[0].empty())
                    break;

                this->array->addSquare(
                        std::stoi(this->menu->addModeValue[0]),
                        this->menu->addModeValue[1],
                        this->addModeEvents(std::stoi(this->menu->addModeValue[0]))
                );

                std::cout « "Add: " « this->menu->addModeValue[0] « " " « this->menu->addModeValue[1] « std::endl;
                this->menu->addModeValue[0] = this->menu->addModeValue[1] = "None";
                this->controlMenu->reset();
                break;
            case constants::MenuArray::Button::DELETE_BUTTON:
                if (this->menu->deleteModeValue == "None" || this->menu->deleteModeValue.empty())
                    break;

                this->array->deleteSquare(
                        std::stoi(this->menu->deleteModeValue),
                        this->deleteModeEvents(std::stoi(this->menu->deleteModeValue))
                );

                std::cout « "Delete: " « this->menu->deleteModeValue « std::endl;
                this->menu->deleteModeValue = "None";
                this->controlMenu->reset();
                break;
            case constants::MenuArray::Button::UPDATE_BUTTON:
                if (this->menu->updateModeValue[0] == "None" || this->menu->updateModeValue[1] == "None" || this->menu->updateModeValue[0].empty())
                    break;

```

```
00083                 this->array->updateSquare(
00084                         std::stoi(this->menu->updateModeValue[0]),
00085                         this->menu->updateModeValue[1],
00086                         this->updateModeEvents(std::stoi(this->menu->updateModeValue[0]))
00087                 );
00088 
00089                 std::cout « "Update: " « this->menu->updateModeValue[0] « " " «
      this->menu->updateModeValue[1] « std::endl;
00090                 this->menu->updateModeValue[0] = this->menu->updateModeValue[1] = "None";
00091                 this->controlMenu->reset();
00092                 break;
00093             case constants::MenuArray::Button::SEARCH_BUTTON:
00094                 if (this->menu->searchModeValue == "None" || this->menu->searchModeValue.empty())
00095                     break;
00096 
00097                 this->array->searchSquare(
00098                         this->searchModeEvents(this->array->findValue(this->menu->searchModeValue))
00099                 );
00100 
00101                 std::cout « "Search: " « this->menu->searchModeValue « std::endl;
00102                 this->menu->searchModeValue = "None";
00103                 this->controlMenu->reset();
00104                 break;
00105         }
00106     }
00107 
00108     this->controlMenu->update();
00109 
00110     this->array->processControlMenu(this->controlMenu->getStatus());
00111     this->array->setSpeed(this->controlMenu->getSpeed());
00112 
00113     this->array->update();
00114 }
```

7.40.3.8 updateModeEvents()

```
std::vector< EventAnimation > StaticArraySceneः:updateModeEvents (
              int chosenNode )
```

Definition at line 265 of file StaticArrayScene.cpp.

```
00265                                                                          {
00266     this->array->resetEvents();
00267     if (chosenNode < 0 || chosenNode >= this->array->getSize())
00268         return {};
00269 
00270     // init highlighter
00271     // ...
00272 
00273     std::vector<EventAnimation> events;
00274     EventAnimation event;
00275 
00276     event = EventAnimation();
00277     event.eventSquares.assign(this->array->getSquaresSize(), EventSquare());
00278     for (int i = 0; i < this->array->getSize(); ++i) {
00279         event.eventSquares[i].status = Square::Status::active;
00280         if (i == this->array->getSize() - 1)
00281             event.eventSquares[this->array->getSize() - 1].title = "n";
00282     }
00283     event.eventSquares[chosenNode].status = Square::Status::chosen;
00284     event.eventSquares[chosenNode].isPrintPreVal = true;
00285 
00286     events.emplace_back(event);
00287 
00288     event.eventSquares[chosenNode].isPrintPreVal = false;
00289 
00290     events.emplace_back(event);
00291 
00292     return events;
00293 }
```

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

- include/libScene/StaticArrayScene.hpp
- include/libScene/StaticArrayScene.cpp

7.41 TextBox Class Reference

```
#include <Textbox.hpp>
```

Public Member Functions

- TextBox (sf::RenderWindow *window, sf::Vector2f position, int size, sf::Color textColor, sf::Color boxColor, int maxLength)
- void pollEvent (sf::Event event)
- void update ()
- void render ()
- std::string getTextString () const
- sf::RectangleShape getBox () const
- void resetInput ()

7.41.1 Detailed Description

Definition at line 13 of file Textbox.hpp.

7.41.2 Constructor & Destructor Documentation

7.41.2.1 TextBox()

```
TextBox::TextBox (
              sf::RenderWindow * window,
              sf::Vector2f position,
              int size,
              sf::Color textColor,
              sf::Color boxColor,
              int maxLength )
```

Definition at line 7 of file Textbox.cpp.

```
                                                    {
    this->window = window;

    this->cursor = "|";

    this->box.setPosition(position);
    this->box.setSize(sf::Vector2f(static_cast<float>((maxLength + 1) * 12), static_cast<float>(size * 1.5)));
    this->box.setFillColor(boxColor);
    this->box.setOutlineColor(sf::Color::Black);
    this->box.setOutlineThickness(1);

    this->font.loadFromFile(constants::fontPath);
    this->text.setFont(this->font);
    this->text.setCharacterSize(size);
    this->text.setFillColor(textColor);
    this->text.setPosition(position);

    this->maxLength = maxLength;
    this->textColor = textColor;
    this->boxColor = boxColor;

    this->cursorVisible = true;
    this->flickerClock.restart();
}
```

7.41.3 Member Function Documentation

7.41.3.1 getBox()

```
sf::RectangleShape TextBox::getBox ( ) const
```

Definition at line 88 of file Textbox.cpp.

```
                                        {
    return this->box;
}
```

7.41.3.2 getTextString()

```
std::string TextBox::getTextString ( ) const
```

Definition at line 84 of file Textbox.cpp.

```
                                        {
    return this->inputString;
}
```

7.41.3.3 pollEvent()

```
void TextBox::pollEvent (
            sf::Event event )
```

Definition at line 33 of file Textbox.cpp.

```
                                      {
    if (event.type == sf::Event::TextEntered)
    {
        if (event.text.unicode == '\b')
        {
            if (!this->inputString.empty())
            {
                this->inputString.pop_back();
            }
        }
        else if (((48 <= event.text.unicode && event.text.unicode <= 57) || event.text.unicode ==
static_cast<int>(',')) && this->inputString.size() < this->maxLength)
        {
            this->inputString += static_cast<char>(event.text.unicode);
        }

        this->text.setString(this->inputString);
    }

//    if (event.type == sf::Event::Resized)
//    {
//        box.setPosition(
//                static_cast<float>(this->window->getSize().x) / 2 - box.getSize().x / 2,
//                static_cast<float>(this->window->getSize().y) / 2 - box.getSize().y / 2
//                );
//        text.setPosition(box.getPosition().x + 10, box.getPosition().y);
//        cursor.setPosition(text.getGlobalBounds().width + text.getPosition().x,
cursor.getPosition().y);
//    }
}
```

7.41.3.4 render()

```
void TextBox::render ( )
```

Definition at line 79 of file Textbox.cpp.

```
                        {
    this->window->draw(this->box);
    this->window->draw(this->text);
}
```

7.41.3.5 resetInput()

```
void TextBox::resetInput ( )
```

Definition at line 92 of file Textbox.cpp.

```
                            {
    this->inputString = "";
    this->text.setString(this->inputString);
}
```

7.41.3.6 update()

```
void TextBox::update ( )
```

Definition at line 62 of file Textbox.cpp.

```
                        {
    if (this->flickerClock.getElapsedTime().asSeconds() >= 0.5)
    {
        this->cursorVisible = !this->cursorVisible;
        this->flickerClock.restart();
    }

    if (this->cursorVisible)
    {
        this->text.setString(this->inputString + this->cursor);
    }
    else
    {
        this->text.setString(this->inputString);
    }
}
```

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

- include/stuff/Textbox.hpp
- include/stuff/Textbox.cpp

7.42 Vector< T > Class Template Reference

```
#include <Vector.h>
```

Public Member Functions

- Vector ()
- Vector (int capacity)
- Vector (const Vector< T > &other)
- ∼Vector ()
- void push_back (T data)
- void pop_back ()
- void insert (int index, T data)
- void erase (int index)
- void erase (T ∗position)
- void clear ()
- void resize (int capacity)
- void assign (int capacity, T data)
- T & operator[] (int index)
- Vector< T > & operator= (const Vector< T > &other)
- int getCapacity () const
- int size () const
- bool empty () const
- T & at (int index) const
- T & front () const
- T & back () const
- T ∗ data () const
- T ∗ begin ()
- T ∗ end ()

7.42.1 Detailed Description

template<class T>
class Vector< T >

Definition at line 8 of file Vector.h.

7.42.2 Constructor & Destructor Documentation

7.42.2.1 Vector() [1/3]

```
template<class T >
Vector< T >::Vector
```

Definition at line 223 of file Vector.h.

```
00223                    {
00224     this->capacity = 10;
00225     this->_size = 0;
00226     this->arr = new T[this->capacity];
00227 }
```

7.42.2.2 Vector() [2/3]

```
template<class T >
Vector< T >::Vector (
              int capacity )  [explicit]
```

Definition at line 216 of file Vector.h.

```
                              {
    this->capacity = capacity;
    this->_size = capacity;
    this->arr = new T[this->capacity];
}
```

7.42.2.3 Vector() [3/3]

```
template<class T >
Vector< T >::Vector (
              const Vector< T > & other )
```

Definition at line 201 of file Vector.h.

```
                                        {
    this->capacity = other.capacity;
    this->_size = other._size;
    this->arr = new T[this->capacity];
    for (int i = 0; i < this->_size; i++) {
        this->arr[i] = other.arr[i];
    }
}
```

7.42.2.4 ∼Vector()

```
template<class T >
Vector< T >::∼Vector
```

Definition at line 211 of file Vector.h.

```
                    {
    delete[] this->arr;
}
```

7.42.3 Member Function Documentation

7.42.3.1 assign()

```
template<class T >
void Vector< T >::assign (
              int capacity,
              T data )
```

Definition at line 46 of file Vector.h.

```
                                            {
    this->clear();
    this->resize(_capacity);
    for (int i = 0; i < capacity; ++i) {
        this->arr[i] = data;
    }

}
```

7.42.3.2 at()

```
template<class T >
T & Vector< T >::at (
             int index ) const
```

Definition at line 91 of file Vector.h.

```
                                 {
    return this->arr[index];
}
```

7.42.3.3 back()

```
template<class T >
T & Vector< T >::back
```

Definition at line 81 of file Vector.h.

```
                           {
    return this->arr[this->_size - 1];
}
```

7.42.3.4 begin()

```
template<class T >
T * Vector< T >::begin
```

Definition at line 71 of file Vector.h.

```
                      {
    return this->arr;
}
```

7.42.3.5 clear()

```
template<class T >
void Vector< T >::clear
```

Definition at line 144 of file Vector.h.

```
                        {
    this->_size = 0;
}
```

7.42.3.6 data()

```
template<class T >
T * Vector< T >::data
```

Definition at line 76 of file Vector.h.

```
                           {
    return this->arr;
}
```

7.42.3.7 empty()

```
template<class T >
bool Vector< T >::empty
```

Definition at line 96 of file Vector.h.

```
                          {
    return this->_size == 0;
}
```

7.42.3.8 end()

```
template<class T >
T * Vector< T >::end
```

Definition at line 66 of file Vector.h.

```
                    {
    return this->arr + this->_size;
}
```

7.42.3.9 erase() [1/2]

```
template<class T >
void Vector< T >::erase (
              int index )
```

Definition at line 149 of file Vector.h.

```
                                        {
    if (index >= 0 && index < this->_size) {
        for (int i = index; i < this->_size - 1; i++) {
            this->arr[i] = this->arr[i + 1];
        }
        this->_size--;
    }
}
```

7.42.3.10 erase() [2/2]

```
template<class T >
void Vector< T >::erase (
              T * position )
```

Definition at line 56 of file Vector.h.

```
                                          {
    for (int i = 0; i < this->_size; ++i) {
        if (this->arr + i == position) {
            this->erase(i);
            break;
        }
    }
}
```

7.42.3.11 front()

```
template<class T >
T & Vector< T >::front
```

Definition at line 86 of file Vector.h.

```
00086                           {
00087     return this->arr[0];
00088 }
```

7.42.3.12 getCapacity()

```
template<class T >
int Vector< T >::getCapacity
```

Definition at line 106 of file Vector.h.

```
00106                                 {
00107     return this->capacity;
00108 }
```

7.42.3.13 insert()

```
template<class T >
void Vector< T >::insert (
              int index,
              T data )
```

Definition at line 159 of file Vector.h.

```
00159                                       {
00160     if (index >= 0 && index <= this->_size) {
00161         if (this->_size >= this->capacity) {
00162             this->capacity *= 2;
00163             T* temp = new T[this->capacity];
00164             for (int i = 0; i < this->_size; i++) {
00165                 temp[i] = this->arr[i];
00166             }
00167             delete[] this->arr;
00168             this->arr = temp;
00169         }
00170         for (int i = this->_size; i > index; i--) {
00171             this->arr[i] = this->arr[i - 1];
00172         }
00173         this->arr[index] = data;
00174         this->_size++;
00175     }
00176 }
```

7.42.3.14 operator=()

```
template<class T >
Vector< T > & Vector< T >::operator= (
              const Vector< T > & other )
```

Definition at line 111 of file Vector.h.

```
00111                                                         {
00112     if (this != &other) {
00113         this->capacity = other.capacity;
00114         this->_size = other._size;
00115         delete[] this->arr;
00116         this->arr = new T[this->capacity];
00117         for (int i = 0; i < this->_size; i++) {
00118             this->arr[i] = other.arr[i];
00119         }
00120     }
00121     return *this;
00122 }
```

7.42.3.15 operator[]()

```
template<class T >
T & Vector< T >::operator[ ] (
            int index )
```

Definition at line 125 of file Vector.h.

```
                              {
    return this->arr[index];
}
```

7.42.3.16 pop_back()

```
template<class T >
void Vector< T >::pop_back
```

Definition at line 179 of file Vector.h.

```
                      {
    if (this->_size > 0) {
        this->_size--;
    }
}
```

7.42.3.17 push_back()

```
template<class T >
void Vector< T >::push_back (
            T data )
```

Definition at line 186 of file Vector.h.

```
                            {
    if (this->_size >= this->capacity) {
        this->capacity *= 2;
        T* temp = new T[this->capacity];
        for (int i = 0; i < this->_size; i++) {
            temp[i] = this->arr[i];
        }
        delete[] this->arr;
        this->arr = temp;
    }
    this->arr[this->_size] = data;
    this->_size++;
}
```

7.42.3.18 resize()

```
template<class T >
void Vector< T >::resize (
            int capacity )
```

Definition at line 130 of file Vector.h.

```
                                {
    this->_size = _capacity;
    if (_capacity > 0) {
        this->capacity = _capacity;
        T* temp = new T[this->capacity];
        for (int i = 0; i < this->_size; i++) {
            temp[i] = this->arr[i];
        }
        delete[] this->arr;
        this->arr = temp;
    }
}
```

7.42.3.19 size()

```
template<class T >
int Vector< T >::size
```

Definition at line 101 of file Vector.h.

```
                              {
    return this->_size;
}
```

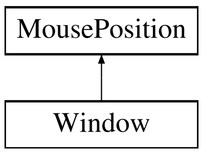
The documentation for this class was generated from the following file:

- include/core/Vector.h

7.43 Window Class Reference

```
#include <Window.hpp>
```

Inheritance diagram for Window:



Public Member Functions

- Window ()
- ∼Window ()=default
- const bool running () const
- void pollEvent ()
- void update ()
- void render ()

Public Member Functions inherited from MousePosition

- void updateMousePosition ()

Additional Inherited Members

Protected Attributes inherited from MousePosition

- sf::RenderWindow ∗ relativeWindow
- sf::Vector2i mousePos
- sf::Vector2f mousePosView

7.43.1 Detailed Description

Definition at line 15 of file Window.hpp.

7.43.2 Constructor & Destructor Documentation

7.43.2.1 Window()

```
Window::Window ( )
```

Definition at line 50 of file Window.cpp.

```
                {
    this->initWindow();
    this->initScenes();
    this->init();
}
```

7.43.2.2 ∼Window()

```
Window::∼Window ( ) [default]
```

7.43.3 Member Function Documentation

7.43.3.1 pollEvent()

```
void Window::pollEvent ( )
```

Definition at line 60 of file Window.cpp.

```
                        {
    // event polling
    while (this->window->pollEvent(this->event)) {
        switch (this->event.type) {
            case sf::Event::Closed:
                this->window->close();
                break;
            case sf::Event::KeyPressed:
                if (this->event.key.code == sf::Keyboard::Q) {
                    std::cout « "You have pressed Q!\n";
                }
                if (this->event.key.code == sf::Keyboard::W) {
                    std::cout « "You have pressed W!\n";
                }
                break;
            default:
                break;
        }

        if (this->submenuButton->pollEvent(this->mousePosView)) {
            std::cout « "You have clicked on submenu button!\n";
            this->scenes[this->currentScene]->isMenuOpen = (this->submenuButton->getTextString() ==
      "<");
        }

```

```
00084         if (this->demoCodeButton->pollEvent(this->mousePosView)) {
00085             std::cout « "You have clicked on demo code button!\n";
00086             this->scenes[this->currentScene]->isDemoCodeOpen = (this->demoCodeButton->getTextString()
      == ">");
00087         }
00088 
00089         for (int i = 1; i < constants::sceneVariables::SCENE_COUNT; i++) {
00090             if (this->scenes[i]->modeButton->pollEvent(this->mousePosView)) {
00091                 std::cout « "You have clicked on " « constants::sceneVariables::SCENE_NAMES[i] « "
      scene!\n";
00092                 this->currentScene = static_cast<constants::sceneVariables::Scene>(i);
00093                 this->scenes[this->currentScene]->isMenuOpen = (this->submenuButton->getTextString()
      == "<");
00094                 this->scenes[this->currentScene]->isDemoCodeOpen =
      (this->demoCodeButton->getTextString() == ">");
00095             }
00096         }
00097 
00098         this->scenes[this->currentScene]->pollEvent(this->event, this->mousePosView);
00099     }
00100 }
```

7.43.3.2 render()

```
void Window::render ( )
```

Definition at line 119 of file Window.cpp.

```
00119                       {
00120     /*
00121  * clear old frames
00122  * create objects
00123  * display it
00124  */
00125 
00126     this->window->clear(sf::Color::White);
00127 
00128     // drawing game
00129     this->submenuButton->render();
00130     this->demoCodeButton->render();
00131     for (int i = 1; i < constants::sceneVariables::SCENE_COUNT; i++) {
00132         this->scenes[i]->modeButton->render();
00133     }
00134 
00135     this->scenes[this->currentScene]->render();
00136 
00137     this->window->display();
00138 }
```

7.43.3.3 running()

```
const bool Window::running ( ) const
```

Definition at line 56 of file Window.cpp.

```
00056                                   {
00057     return this->window->isOpen();
00058 }
```

7.43.3.4 update()

```
void Window::update ( )
```

Definition at line 102 of file Window.cpp.

```
00102                       {
00103     this->scenes[this->currentScene]->modeButton->setColor(constants::normalGray);
00104 
00105     this->updateMousePosition();
00106     this->pollEvent();
00107 
00108     this->submenuButton->update();
00109     this->demoCodeButton->update();
00110     this->scenes[this->currentScene]->modeButton->setColor(constants::hoverGreen);
00111 
00112     for (int i = 1; i < constants::sceneVariables::SCENE_COUNT; i++) {
00113         this->scenes[i]->modeButton->update();
00114     }
00115 
00116     this->scenes[this->currentScene]->update();
00117 }
```

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

- include/Window.hpp
- include/Window.cpp

Chapter 8

File Documentation

8.1 include/Constants.hpp File Reference

```
#include <SFML/Graphics.hpp>
```

Namespaces

- namespace constants
- namespace constants::sceneVariables
- namespace constants::MenuArray
- namespace constants::MenuArray::CreateMode
- namespace constants::MenuArray::AddMode
- namespace constants::MenuArray::DeleteMode
- namespace constants::MenuArray::UpdateMode
- namespace constants::MenuArray::SearchMode
- namespace constants::MenuArray::AllocateMode
- namespace constants::MenuDataStructure
- namespace constants::MenuDataStructure::CreateMode
- namespace constants::MenuDataStructure::PushMode
- namespace constants::MenuLinkedList
- namespace constants::MenuLinkedList::CreateMode
- namespace constants::MenuLinkedList::AddMode
- namespace constants::MenuLinkedList::DeleteMode
- namespace constants::MenuLinkedList::UpdateMode
- namespace constants::MenuLinkedList::SearchMode
- namespace constants::NodeInfo
- namespace constants::Square
- namespace constants::Arrow
- namespace constants::ControlMenu
- namespace constants::Highlighter
- namespace constants::Highlighter::SLL
- namespace constants::Highlighter::DLL
- namespace constants::LinkedList
- namespace constants::TitleNode

Enumerations

- enum constants::sceneVariables::Scene {
  constants::sceneVariables::MAIN_MENU_SCENE , constants::sceneVariables::SINGLY_LINKED_LIST_SCENE
  , constants::sceneVariables::DOUBLY_LINKED_LIST_SCENE , constants::sceneVariables::CIRCULAR_LINKED_LIST_SCEN
  ,
  constants::sceneVariables::STACK_SCENE , constants::sceneVariables::QUEUE_SCENE , constants::sceneVariables::STATIC
  , constants::sceneVariables::DYNAMIC_ARRAY_SCENE }
- enum class constants::MenuArray::Type { constants::MenuArray::DYNAMIC , constants::MenuArray::STATIC
  }
- enum constants::MenuArray::Button {
  constants::MenuArray::CREATE_BUTTON , constants::MenuArray::ADD_BUTTON , constants::MenuArray::DELETE_BUTTON
  , constants::MenuArray::UPDATE_BUTTON ,
  constants::MenuArray::SEARCH_BUTTON , constants::MenuArray::ALLOCATE_BUTTON , constants::MenuArray::NONE
  }
- enum constants::MenuArray::CreateMode::Button { constants::MenuArray::CreateMode::RANDOM_BUTTON
  , constants::MenuArray::CreateMode::DEFINED_LIST_BUTTON , constants::MenuArray::CreateMode::FILE_BUTTON
  , constants::MenuArray::CreateMode::NONE }
- enum constants::MenuArray::AddMode::Textbox { constants::MenuArray::AddMode::POSITION_TEXTBOX ,
  constants::MenuArray::AddMode::VALUE_TEXTBOX , constants::MenuArray::AddMode::NONE }
- enum constants::MenuArray::DeleteMode::Textbox { constants::MenuArray::DeleteMode::POSITION_TEXTBOX
  , constants::MenuArray::DeleteMode::NONE }
- enum constants::MenuArray::UpdateMode::Textbox { constants::MenuArray::UpdateMode::POSITION_TEXTBOX
  , constants::MenuArray::UpdateMode::VALUE_TEXTBOX , constants::MenuArray::UpdateMode::NONE }
- enum constants::MenuArray::SearchMode::Textbox { constants::MenuArray::SearchMode::VALUE_TEXTBOX
  , constants::MenuArray::SearchMode::NONE }
- enum constants::MenuArray::AllocateMode::Textbox { constants::MenuArray::AllocateMode::VALUE_TEXTBOX
  , constants::MenuArray::AllocateMode::NONE }
- enum constants::MenuDataStructure::Button {
  constants::MenuDataStructure::CREATE_BUTTON , constants::MenuDataStructure::PUSH_BUTTON ,
  constants::MenuDataStructure::POP_BUTTON , constants::MenuDataStructure::CLEAR_BUTTON ,
  constants::MenuDataStructure::NONE }
- enum constants::MenuDataStructure::CreateMode::Button { constants::MenuDataStructure::CreateMode::RANDOM_BUTTON
  , constants::MenuDataStructure::CreateMode::DEFINED_LIST_BUTTON , constants::MenuDataStructure::CreateMode::FILE_
  , constants::MenuDataStructure::CreateMode::NONE }
- enum constants::MenuDataStructure::PushMode::Textbox { constants::MenuDataStructure::PushMode::VALUE_TEXTBOX
  , constants::MenuDataStructure::PushMode::NONE }
- enum constants::MenuLinkedList::Button {
  constants::MenuLinkedList::CREATE_BUTTON , constants::MenuLinkedList::ADD_BUTTON , constants::MenuLinkedList::DEL
  , constants::MenuLinkedList::UPDATE_BUTTON ,
  constants::MenuLinkedList::SEARCH_BUTTON , constants::MenuLinkedList::NONE }
- enum constants::MenuLinkedList::CreateMode::Button { constants::MenuLinkedList::CreateMode::RANDOM_BUTTON
  , constants::MenuLinkedList::CreateMode::DEFINED_LIST_BUTTON , constants::MenuLinkedList::CreateMode::FILE_BUTTO
  , constants::MenuLinkedList::CreateMode::NONE }
- enum constants::MenuLinkedList::AddMode::Textbox { constants::MenuLinkedList::AddMode::POSITION_TEXTBOX
  , constants::MenuLinkedList::AddMode::VALUE_TEXTBOX , constants::MenuLinkedList::AddMode::NONE }
- enum constants::MenuLinkedList::DeleteMode::Textbox { constants::MenuLinkedList::DeleteMode::POSITION_TEXTBOX
  , constants::MenuLinkedList::DeleteMode::NONE }
- enum constants::MenuLinkedList::UpdateMode::Textbox { constants::MenuLinkedList::UpdateMode::POSITION_TEXTBOX
  , constants::MenuLinkedList::UpdateMode::VALUE_TEXTBOX , constants::MenuLinkedList::UpdateMode::NONE
  }
- enum constants::MenuLinkedList::SearchMode::Textbox { constants::MenuLinkedList::SearchMode::VALUE_TEXTBOX
  , constants::MenuLinkedList::SearchMode::NONE }
- enum class constants::ControlMenu::Button {
  constants::ControlMenu::PREVIOUS , constants::ControlMenu::PLAY , constants::ControlMenu::NEXT ,
  constants::ControlMenu::SPEED_DOWN ,
  constants::ControlMenu::SPEED_UP , constants::ControlMenu::None }

Functions

- static sf::Vector2i constants::Arrow::sizeRectangle (192, 37)
- static sf::Vector2f constants::Arrow::defaultScaleRectangle (0.6f, 0.16f)
- static sf::Vector2f constants::Highlighter::codeScale (0.6f, 0.6f)
- static sf::Color constants::hoverGreen (162, 178, 159)
- static sf::Color constants::clickGreen (121, 135, 119)
- static sf::Color constants::transparentGreen (189, 210, 182, 150)
- static sf::Color constants::hoverGray (150, 150, 150)
- static sf::Color constants::clickGray (100, 100, 100)

Variables

- constexpr int constants::sceneVariables::SCENE_COUNT = 8
- constexpr char constants::sceneVariables::SCENE_NAMES [SCENE_COUNT][50]
- constexpr char constants::sceneVariables::NAME_MODE_BUTTON [SCENE_COUNT][50]
- constexpr int constants::MenuArray::BUTTON_COUNT = 6
- constexpr char constants::MenuArray::BUTTON_NAMES [BUTTON_COUNT][50]
- constexpr int constants::MenuArray::BUTTON_NAME_SIZE = 15
- constexpr int constants::MenuArray::CreateMode::BUTTON_COUNT = 3
- constexpr char constants::MenuArray::CreateMode::BUTTON_NAMES [BUTTON_COUNT][50]
- constexpr int constants::MenuArray::CreateMode::NAME_SIZE = 15
- constexpr int constants::MenuArray::CreateMode::TEXTBOX_COUNT = 2
- constexpr char constants::MenuArray::CreateMode::TEXTBOX_NAMES [2][50]
- constexpr int constants::MenuArray::CreateMode::TEXTBOX_LENGTH [2]
- constexpr int constants::MenuArray::AddMode::TEXTBOX_COUNT = 2
- constexpr char constants::MenuArray::AddMode::TEXTBOX_NAMES [2][50]
- constexpr int constants::MenuArray::AddMode::TEXTBOX_LENGTH [2]
- constexpr int constants::MenuArray::DeleteMode::TEXTBOX_COUNT = 1
- constexpr char constants::MenuArray::DeleteMode::TEXTBOX_NAME [50] = "Position = "
- constexpr int constants::MenuArray::DeleteMode::TEXTBOX_LENGTH = 2
- constexpr int constants::MenuArray::UpdateMode::TEXTBOX_COUNT = 2
- constexpr char constants::MenuArray::UpdateMode::TEXTBOX_NAMES [2][50]
- constexpr int constants::MenuArray::UpdateMode::TEXTBOX_LENGTH [2]
- constexpr int constants::MenuArray::SearchMode::TEXTBOX_COUNT = 1
- constexpr char constants::MenuArray::SearchMode::TEXTBOX_NAME [50] = "Value = "
- constexpr int constants::MenuArray::SearchMode::TEXTBOX_LENGTH = 2
- constexpr int constants::MenuArray::AllocateMode::TEXTBOX_COUNT = 1
- constexpr char constants::MenuArray::AllocateMode::TEXTBOX_NAME [50] = "Size = "
- constexpr int constants::MenuArray::AllocateMode::TEXTBOX_LENGTH = 2
- constexpr int constants::MenuDataStructure::BUTTON_COUNT = 4
- constexpr char constants::MenuDataStructure::BUTTON_NAMES [BUTTON_COUNT][50]
- constexpr int constants::MenuDataStructure::BUTTON_NAME_SIZE = 15
- constexpr int constants::MenuDataStructure::CreateMode::BUTTON_COUNT = 3
- constexpr char constants::MenuDataStructure::CreateMode::BUTTON_NAMES [BUTTON_COUNT][50]
- constexpr int constants::MenuDataStructure::CreateMode::NAME_SIZE = 15
- constexpr int constants::MenuDataStructure::CreateMode::TEXTBOX_COUNT = 2
- constexpr char constants::MenuDataStructure::CreateMode::TEXTBOX_NAMES [2][50]
- constexpr int constants::MenuDataStructure::CreateMode::TEXTBOX_LENGTH [2]
- constexpr int constants::MenuDataStructure::PushMode::TEXTBOX_COUNT = 1
- constexpr char constants::MenuDataStructure::PushMode::TEXTBOX_NAME [50] = "Value = "
- constexpr int constants::MenuDataStructure::PushMode::TEXTBOX_LENGTH = 2
- constexpr int constants::MenuLinkedList::BUTTON_COUNT = 5
- constexpr char constants::MenuLinkedList::BUTTON_NAMES [BUTTON_COUNT][50]

- constexpr int constants::MenuLinkedList::BUTTON_NAME_SIZE = 15
- constexpr int constants::MenuLinkedList::CreateMode::BUTTON_COUNT = 3
- constexpr char constants::MenuLinkedList::CreateMode::BUTTON_NAMES [BUTTON_COUNT][50]
- constexpr int constants::MenuLinkedList::CreateMode::NAME_SIZE = 15
- constexpr int constants::MenuLinkedList::CreateMode::TEXTBOX_COUNT = 2
- constexpr char constants::MenuLinkedList::CreateMode::TEXTBOX_NAMES [2][50]
- constexpr int constants::MenuLinkedList::CreateMode::TEXTBOX_LENGTH [2]
- constexpr int constants::MenuLinkedList::AddMode::TEXTBOX_COUNT = 2
- constexpr char constants::MenuLinkedList::AddMode::TEXTBOX_NAMES [2][50]
- constexpr int constants::MenuLinkedList::AddMode::TEXTBOX_LENGTH [2]
- constexpr int constants::MenuLinkedList::DeleteMode::TEXTBOX_COUNT = 1
- constexpr char constants::MenuLinkedList::DeleteMode::TEXTBOX_NAME [50] = "Position = "
- constexpr int constants::MenuLinkedList::DeleteMode::TEXTBOX_LENGTH = 2
- constexpr int constants::MenuLinkedList::UpdateMode::TEXTBOX_COUNT = 2
- constexpr char constants::MenuLinkedList::UpdateMode::TEXTBOX_NAMES [2][50]
- constexpr int constants::MenuLinkedList::UpdateMode::TEXTBOX_LENGTH [2]
- constexpr int constants::MenuLinkedList::SearchMode::TEXTBOX_COUNT = 1
- constexpr char constants::MenuLinkedList::SearchMode::TEXTBOX_NAME [50] = "Value = "
- constexpr int constants::MenuLinkedList::SearchMode::TEXTBOX_LENGTH = 2
- constexpr int constants::ControlMenu::BUTTON_COUNT = 5
- constexpr int constants::ControlMenu::BUTTON_NAME_SIZE = 15
- constexpr int constants::ControlMenu::TEXT_SIZE = 15
- constexpr char constants::ControlMenu::BUTTON_NAMES [BUTTON_COUNT][50]
- const std::pair< const char ∗, const int > constants::Highlighter::SLL::CODES_PATH [4]
- const std::pair< const char ∗, const int > constants::Highlighter::DLL::CODES_PATH [ ]
- constexpr char constants::titleWindow [ ] = "Visualgo CS162 - Phan Minh Quang"
- constexpr char constants::fontPath [ ] = "../assets/fonts/Hack_reg.ttf"

8.2 Constants.hpp

Go to the documentation of this file.

```
//
// Created by dirii on 23/03/2023.
//

#ifndef VISUALGO_CS162_CONSTANTS_HPP
#define VISUALGO_CS162_CONSTANTS_HPP

#include <SFML/Graphics.hpp>

namespace constants{
    namespace sceneVariables {
        constexpr int SCENE_COUNT = 8;
        enum Scene {
            MAIN_MENU_SCENE,
            SINGLY_LINKED_LIST_SCENE,
            DOUBLY_LINKED_LIST_SCENE,
            CIRCULAR_LINKED_LIST_SCENE,
            STACK_SCENE,
            QUEUE_SCENE,
            STATIC_ARRAY_SCENE,
            DYNAMIC_ARRAY_SCENE,
        };
        constexpr char SCENE_NAMES[SCENE_COUNT][50] = {
                "Main Menu",
                "Singly Linked List",
                "Doubly Linked List",
                "Circular Linked List",
                "Stack",
                "Queue",
                "Static Array",
                "Dynamic Array",
        };
        constexpr char NAME_MODE_BUTTON[SCENE_COUNT][50] = {
            "Main Menu",
            "SLL",
            "DLL",
            "CLL",
            "Stack",
            "Queue",
            "Static Array",
            "Dynamic Array"
        };
    }

    namespace MenuArray{
        enum class Type{
            DYNAMIC,
            STATIC
        };

        constexpr int BUTTON_COUNT = 6;
        enum Button{
            CREATE_BUTTON,
            ADD_BUTTON,
            DELETE_BUTTON,
            UPDATE_BUTTON,
            SEARCH_BUTTON,
            ALLOCATE_BUTTON,
            NONE
        };
        constexpr char BUTTON_NAMES[BUTTON_COUNT][50] = {
                "Create",
                "Add",
                "Delete",
                "Update",
                "Search",
                "Allocate"
        };
        constexpr int BUTTON_NAME_SIZE = 15;

        namespace CreateMode {
            constexpr int BUTTON_COUNT = 3;
            enum Button {
                RANDOM_BUTTON,
                DEFINED_LIST_BUTTON,
                FILE_BUTTON,
                NONE
            };
            constexpr char BUTTON_NAMES[BUTTON_COUNT][50] = {
                    "Random",
                    "Defined List",
                    "File"
            };
            constexpr int NAME_SIZE = 15;

            constexpr int TEXTBOX_COUNT = 2;
            constexpr char TEXTBOX_NAMES[2][50] = {
                    "Amount = ",
                    "List = "
            };

            constexpr int TEXTBOX_LENGTH[2] = {
                    2,
                    30 // for input a defined list
            };
        }
        namespace AddMode {
            constexpr int TEXTBOX_COUNT = 2;
            constexpr char TEXTBOX_NAMES[2][50] = {
                    "Position = ",
                    "Value = "
            };
            constexpr int TEXTBOX_LENGTH[2] = {
                    2,
                    2
            };
            enum Textbox{
                POSITION_TEXTBOX,
                VALUE_TEXTBOX,
                NONE
            };
        };
        namespace DeleteMode {
            constexpr int TEXTBOX_COUNT = 1;
            constexpr char TEXTBOX_NAME[50] = "Position = ";
            constexpr int TEXTBOX_LENGTH = 2;
            enum Textbox{
                POSITION_TEXTBOX,
                NONE
            };
        }
        namespace UpdateMode {
            constexpr int TEXTBOX_COUNT = 2;
            constexpr char TEXTBOX_NAMES[2][50] = {
                    "Position = ",
                    "Value = "
            };
            constexpr int TEXTBOX_LENGTH[2] = {
                    2,
                    2
            };
            enum Textbox{
                POSITION_TEXTBOX,
                VALUE_TEXTBOX,
                NONE
            };
        }
        namespace SearchMode {
            constexpr int TEXTBOX_COUNT = 1;
            constexpr char TEXTBOX_NAME[50] = "Value = ";
            constexpr int TEXTBOX_LENGTH = 2;
            enum Textbox{
                VALUE_TEXTBOX,
                NONE
            };
        }
        namespace AllocateMode {
            constexpr int TEXTBOX_COUNT = 1;
            constexpr char TEXTBOX_NAME[50] = "Size = ";
            constexpr int TEXTBOX_LENGTH = 2;
            enum Textbox{
                VALUE_TEXTBOX,
                NONE
            };
        }
    };

    namespace MenuDataStructure{
        constexpr int BUTTON_COUNT = 4;
        enum Button{
            CREATE_BUTTON,
            PUSH_BUTTON,
            POP_BUTTON,
            CLEAR_BUTTON,
            NONE
        };
        constexpr char BUTTON_NAMES[BUTTON_COUNT][50] = {
                "Create",
                "Push",
                "Pop",
                "Clear"
        };
        constexpr int BUTTON_NAME_SIZE = 15;

        namespace CreateMode {
            constexpr int BUTTON_COUNT = 3;
            enum Button {
                RANDOM_BUTTON,
                DEFINED_LIST_BUTTON,
                FILE_BUTTON,
                NONE
            };
            constexpr char BUTTON_NAMES[BUTTON_COUNT][50] = {
                    "Random",
                    "Defined List",
                    "File"
            };
            constexpr int NAME_SIZE = 15;

            constexpr int TEXTBOX_COUNT = 2;
            constexpr char TEXTBOX_NAMES[2][50] = {
                    "Amount = ",
                    "List = "
            };

            constexpr int TEXTBOX_LENGTH[2] = {
                    2,
                    30 // for input a defined list
            };
        }
        namespace PushMode{
            constexpr int TEXTBOX_COUNT = 1;
            constexpr char TEXTBOX_NAME[50] = "Value = ";
            constexpr int TEXTBOX_LENGTH = 2;
            enum Textbox{
                VALUE_TEXTBOX,
                NONE
            };
        }
    }

    namespace MenuLinkedList {
        constexpr int BUTTON_COUNT = 5;
        enum Button {
            CREATE_BUTTON,
            ADD_BUTTON,
            DELETE_BUTTON,
            UPDATE_BUTTON,
            SEARCH_BUTTON,
            NONE
        };
        constexpr char BUTTON_NAMES[BUTTON_COUNT][50] = {
                "Create",
                "Add",
                "Delete",
                "Update",
                "Search"
        };
        constexpr int BUTTON_NAME_SIZE = 15;

        namespace CreateMode {
            constexpr int BUTTON_COUNT = 3;
            enum Button {
                RANDOM_BUTTON,
                DEFINED_LIST_BUTTON,
                FILE_BUTTON,
                NONE
            };
            constexpr char BUTTON_NAMES[BUTTON_COUNT][50] = {
                    "Random",
                    "Defined List",
                    "File"
            };
            constexpr int NAME_SIZE = 15;

            constexpr int TEXTBOX_COUNT = 2;
            constexpr char TEXTBOX_NAMES[2][50] = {
                    "Amount = ",
                    "List = "
            };

            constexpr int TEXTBOX_LENGTH[2] = {
                    2,
                    30 // for input a defined list
            };
        }
        namespace AddMode{
            constexpr int TEXTBOX_COUNT = 2;
            constexpr char TEXTBOX_NAMES[2][50] = {
                    "Position = ",
                    "Value = "
            };
            constexpr int TEXTBOX_LENGTH[2] = {
                    2,
                    2
            };
            enum Textbox{
                POSITION_TEXTBOX,
                VALUE_TEXTBOX,
                NONE
            };
        }
        namespace DeleteMode{
            constexpr int TEXTBOX_COUNT = 1;
            constexpr char TEXTBOX_NAME[50] = "Position = ";
            constexpr int TEXTBOX_LENGTH = 2;
            enum Textbox{
                POSITION_TEXTBOX,
                NONE
            };
        }
        namespace UpdateMode{
            constexpr int TEXTBOX_COUNT = 2;
            constexpr char TEXTBOX_NAMES[2][50] = {
                    "Position = ",
                    "Value = "
            };
            constexpr int TEXTBOX_LENGTH[2] = {
                    2,
                    2
            };
            enum Textbox{
                POSITION_TEXTBOX,
                VALUE_TEXTBOX,
                NONE
            };
        }
        namespace SearchMode{
            constexpr int TEXTBOX_COUNT = 1;
            constexpr char TEXTBOX_NAME[50] = "Value = ";
            constexpr int TEXTBOX_LENGTH = 2;
            enum Textbox{
                VALUE_TEXTBOX,
                NONE
            };
        }
    }

    namespace NodeInfo{
        static float radius = 30,
                     outlineThickness = 2;
        static int pointCount = 200,
                     fontSize = 20;
        static sf::Vector2f originNode(100, 300);
        static float offsetX = 170,
                     offsetY = 150;
    }

    namespace Square{
        static float length = 60,
                     outlineThickness = 6;
        static int fontSize = 20;
        static sf::Vector2f originNode(100, 300);
        static float offsetX = 72,
                     offsetY = 150;
    }

    namespace Arrow{
        static sf::Vector2i sizeArrow(752, 214),
                            sizeRectangle(192, 37);
        static sf::Vector2f defaultScaleArrow(0.2f, 0.15f),
                            defaultScaleRectangle(0.6f, 0.16f);
    }

    namespace ControlMenu{
        enum class Button{
            PREVIOUS,
            PLAY,
            NEXT,
            SPEED_DOWN,
            SPEED_UP,
            None
        };

        constexpr int BUTTON_COUNT = 5,
                        BUTTON_NAME_SIZE = 15,
                        TEXT_SIZE = 15;
        constexpr char BUTTON_NAMES[BUTTON_COUNT][50] = {
                "<",
                "[=]",
                ">",
                "«",
                "»"
        };

        static sf::Vector2f buttonSize(50, 50);
        static float coordinateY = 930,
                     middleX = 1760 / 2.0f  - buttonSize.x / 2.0f,
                     leftX = 1760 / 7.0f;
        static sf::Vector2f buttonPos[5] = {
                sf::Vector2f(middleX - 2 * buttonSize.x, coordinateY),
                sf::Vector2f(middleX, coordinateY),
                sf::Vector2f(middleX + 2 * buttonSize.x, coordinateY),
                sf::Vector2f(leftX, coordinateY),
                sf::Vector2f(leftX + 3 * buttonSize.x, coordinateY)
        };
    }

    namespace Highlighter{
        static sf::Vector2f codePos(1726, 930),
                            codeScale(0.6f, 0.6f);

        namespace SLL{
            const std::pair<const char*, const int> CODES_PATH[4] = {
                std::make_pair("../assets/code/SLL/add.png", 10),
                std::make_pair("../assets/code/SLL/delete.png", 11),
                std::make_pair("../assets/code/SLL/update.png", 4),
                std::make_pair("../assets/code/SLL/search.png", 6)
            };
        }

        namespace DLL{
            const std::pair<const char*, const int> CODES_PATH[] = {
                std::make_pair("../assets/code/DLL/add_beginning.png", 8),
                std::make_pair("../assets/code/DLL/add_ending.png", 5),
                std::make_pair("../assets/code/DLL/add_middle.png", 9),
                std::make_pair("../assets/code/DLL/delete_beginning.png", 8),
                std::make_pair("../assets/code/DLL/delete_ending.png", 5),
                std::make_pair("../assets/code/DLL/delete_middle.png", 7),
                std::make_pair("../assets/code/DLL/update.png", 4),
                std::make_pair("../assets/code/DLL/search.png", 6)
            };
        }
    }

    namespace LinkedList{
        static float DELAY_TIME = 1.0f;
    }

    namespace TitleNode{
        static int fontSize = 20;
        static float offsetY = 50;
    }

    // information of window
    static int Width = 1760,
            Height = 992;
    constexpr char titleWindow[] = "Visualgo CS162 - Phan Minh Quang";
    static int fps = 144;

    constexpr char fontPath[] = "../assets/fonts/Hack_reg.ttf";

    // colors
    static sf::Color normalGreen(189, 210, 182),
                    hoverGreen(162, 178, 159),
                    clickGreen(121, 135, 119),
                    transparentGreen(189, 210, 182, 150);

    static sf::Color normalGray(200, 200, 200),
                    hoverGray(150, 150, 150),
                    clickGray(100, 100, 100);

    static sf::Color normalOrange(255, 145, 77);

    static sf::Color titleGreen(64, 81, 59);

    // positions of buttons
    static sf::Vector2f submenuButtonPos = sf::Vector2f(0, 690),
                    democodeButtonPos = sf::Vector2f(1736, 690),
                    modeButtonPos = sf::Vector2f(10, 10);

    // size of buttons
    static sf::Vector2f sideButtonSize = sf::Vector2f(24, 200),
                    modeButtonSize = sf::Vector2f(150, 40),
                    optionButtonSize = sf::Vector2f(130, sideButtonSize.y / static_cast<float>(5)
      - 1),
                    goButtonSize = sf::Vector2f(50, 30);
    static float distance2ModeButtons = 10;

    // size text of buttons
    static int sizeTextModeButton = 15;

    // rounding button
    static int CORNER_POINT_COUNT_BUTTON = 15;
    static float CORNER_RADIUS_BUTTON = 5;
}

#endif //VISUALGO_CS162_CONSTANTS_HPP
```

8.3 include/core/Array.cpp File Reference

```
#include "Array.hpp"
#include <utility>
```

8.4 Array.cpp

Go to the documentation of this file.

```
//
// Created by dirii on 28/04/2023.
//

#include "Array.hpp"

#include <utility>

Array::Array(sf::RenderWindow *window, TypeArray typeArray) : BaseDraw(window) {
    this->init(typeArray);
    this->createArray(0);
}

Array::Array(sf::RenderWindow *window, Array::TypeArray typeArray, int size) : BaseDraw(window)  {
    this->init(typeArray);
    this->createArray(size);
}

Array::Array(sf::RenderWindow *window, Array::TypeArray typeArray, std::vector<std::string> values) :
BaseDraw(window)  {
    this->init(typeArray);
    this->createArray(std::move(values));
}

void Array::init(Array::TypeArray typeArray) {
    this->typeArray = typeArray;
    this->highlighter = nullptr;
    this->delayTime = constants::LinkedList::DELAY_TIME;
    this->size = 0;
}

void Array::render() {
    for (auto &square : this->squares) {
        square->render();
    }
    for (auto &square: this->squaresTemp) {
        square->render();
    }
}

void Array::renderHighlighter() {
    if (this->highlighter)
        this->highlighter->render();
}

void Array::update() {
    if ((int)this->events.size() && (this->isDelay or this->clock.getElapsedTime().asSeconds() >
this->delayTime / this->speed))
        this->updateAnimation();
    this->isDelay = false;
}

void Array::setSpeed(float _speed) {
    this->speed = _speed;
}

int Array::findValue(const std::string &value) {
    for (int i = 0; i < this->size; i++) {
        if (this->squares[i]->getValue() == value)
            return i;
    }
    return this->size;
}

int Array::getSize() const {
    return this->size;
}

void Array::processControlMenu(ControlMenu::StatusCode status) {
    if (this->clock.getElapsedTime().asSeconds() < this->delayTime / this->speed)
        return;
    switch (status){
        case ControlMenu::StatusCode::PREVIOUS:
            if (this->currentEvent > 0)
                --this->currentEvent;
            break;
        case ControlMenu::StatusCode::PAUSE:
//             std::cout « "PAUSE" « std::endl;
            break;
        case ControlMenu::StatusCode::PLAY:
            if (this->currentEvent + 1 < this->events.size()) {
                this->isDelay = true;
                this->clock.restart();
            }
        case ControlMenu::StatusCode::NEXT:
            if (this->currentEvent + 1 < this->events.size())
                ++this->currentEvent;
            break;
        default:
            break;
    }
}

void Array::initHighlighter(int linesCount, const char *codePath) {
    delete this->highlighter;
    this->highlighter = new Highlighter(
            this->window,
            linesCount,
            codePath
    );
}

void Array::toggleLines(std::vector<int> lines) {
    this->highlighter->toggle(std::move(lines));
}

void Array::updateAnimation() {
    if (this->squares.empty())
        return;

    for (auto &square : this->squares) {
        square->reset();
    }
    for (auto &square : this->squaresTemp) {
        square->reset();
    }

    EventAnimation &event = this->events[this->currentEvent];
    for (int i = 0; i < event.eventSquares.size(); ++i) {
        this->squares[i]->setStatus(event.eventSquares[i].status);
        this->squares[i]->setPrintPreVal(event.eventSquares[i].isPrintPreVal);
        this->squares[i]->setTitle(event.eventSquares[i].title);
    }
    for (int i = 0; i < event.eventSquaresTemp.size(); ++i) {
        this->squaresTemp[i]->setStatus(event.eventSquaresTemp[i].status);
        this->squaresTemp[i]->setPrintPreVal(event.eventSquaresTemp[i].isPrintPreVal);
        this->squaresTemp[i]->setTitle(event.eventSquaresTemp[i].title);
    }

    if (this->highlighter)
        this->highlighter->toggle(event.lines);

    for (auto &square : this->squares) {
        square->update();
    }
    for (auto &square : this->squaresTemp) {
        square->update();
    }
}

void Array::resetEvents() {
    delete this->highlighter;
    this->highlighter = nullptr;
    this->currentEvent = 0;
    this->events.clear();
    this->squaresTemp.clear();

    while (!this->squares.empty() && this->squares.back()->getStatus() == Square::Status::hidden)
        this->squares.pop_back();

    for (int i = 0; i < this->size; ++i)
        this->squares[i]->setStatus(Square::Status::active);
    for (int i = this->size; i < this->squares.size(); ++i)
        this->squares[i]->setStatus(Square::Status::inactive);
    if (this->size)
        this->squares[this->size - 1]->setTitle("n");
}

void Array::createArray(int _size) {
    this->resetEvents();
    this->size = _size;
    for (auto &square : this->squares)
        delete square;
    this->squares.resize(this->size);
    for (int i = 0; i < this->size; ++i) {
        this->squares[i] = new SquareInfo(
                this->window,
                std::to_string(Random::randomInt(0, 99)),
                sf::Vector2f(
                        constants::Square::originNode.x + static_cast<float>(i) *
constants::Square::offsetX,
                        constants::Square::originNode.y
                        )
        );
```

```
            this->squares[i]->setStatus(Square::Status::active);
        }
        if (this->size)
            this->squares[this->size - 1]->setTitle("n");
}

void Array::createArray(const std::vector<std::string>& values) {
        this->resetEvents();
        this->size = (int)values.size();
        for (auto &square : this->squares)
            delete square;
        this->squares.resize(this->size);
        for (int i = 0; i < this->size; ++i) {
            this->squares[i] = new SquareInfo(
                    this->window,
                    values[i],
                    sf::Vector2f(
                            constants::Square::originNode.x + static_cast<float>(i) *
constants::Square::offsetX,
                            constants::Square::originNode.y
                    )
            );
            this->squares[i]->setStatus(Square::Status::active);
        }
        if (this->size)
            this->squares[this->size - 1]->setTitle("n");
}

int Array::getSquaresSize() const {
        return (int)this->squares.size();
}

void Array::allocateSquare(int _size, const std::vector<EventAnimation> &listEvents) {
        this->squaresTemp.resize(_size);
//    this->squares.resize();

        while (this->squares.size() < _size)
            this->squares.push_back(new SquareInfo(
                    this->window,
                    "",
                    sf::Vector2f(
                            constants::Square::originNode.x + static_cast<float>(this->squares.size()) *
constants::Square::offsetX,
                            constants::Square::originNode.y
                    )
            ));

        for (int i = 0; i < _size; ++i) {
            this->squaresTemp[i] = new SquareInfo(
                    this->window,
                    "",
                    sf::Vector2f(
                            constants::Square::originNode.x + static_cast<float>(i) *
constants::Square::offsetX,
                            constants::Square::originNode.y + constants::Square::offsetY
                    )
            );
            this->squaresTemp[i]->setValue(this->squares[i]->getValue());
        }

        this->size = std::min(this->size, _size);
        this->currentEvent = 0;
        this->events = listEvents;
}

void Array::addSquare(int position, std::string value, const std::vector<EventAnimation> &listEvents)
{
        if (position < 0 || position > this->size)
            return;

        ++this->size;
        if (this->typeArray == TypeArray::DYNAMIC && this->size > this->getSquaresSize()) {
            this->squares.push_back(new SquareInfo(
                    this->window,
                    "",
                    sf::Vector2f(
                            constants::Square::originNode.x + static_cast<float>(this->getSquaresSize()) *
constants::Square::offsetX,
                            constants::Square::originNode.y
                    )
            ));
            this->squaresTemp.resize(this->size);
            for (int i = 0; i < this->size; ++i) {
                this->squaresTemp[i] = new SquareInfo(
                        this->window,
                        "",
                        sf::Vector2f(
                                constants::Square::originNode.x + static_cast<float>(i) *
    constants::Square::offsetX,
                                constants::Square::originNode.y + constants::Square::offsetY
                        )
                );
                this->squaresTemp[i]->setValue(this->squares[i]->getValue());
            }
        }

        if (size > this->getSquaresSize())
            --this->size;

        for (int i = this->size - 1; i > position; --i)
            this->squares[i]->setValue(this->squares[i - 1]->getValue());
        this->squares[position]->setValue(std::move(value));
        for (int i = 0; i < position; ++i)
            this->squares[i]->setValue(this->squares[i]->getValue());

        this->currentEvent = 0;
        this->events = listEvents;
    }

    void Array::deleteSquare(int position, const std::vector<EventAnimation> &listEvents) {
        if (position < 0 || position >= this->size)
            return;

        --this->size;

        for (int i = position; i < this->size; ++i)
            this->squares[i]->setValue(this->squares[i + 1]->getValue());
        for (int i = 0; i < position; ++i)
            this->squares[i]->setValue(this->squares[i]->getValue());
        this->squares[this->size]->setValue(this->squares[this->size]->getValue());

        this->currentEvent = 0;
        this->events = listEvents;
    }

    void Array::updateSquare(int position, std::string value, const std::vector<EventAnimation>
    &listEvents) {
        if (position < 0 || position >= this->size)
            return;

        this->squares[position]->setValue(std::move(value));

        this->currentEvent = 0;
        this->events = listEvents;
    }

    void Array::searchSquare(const std::vector<EventAnimation> &listEvents) {
        this->currentEvent = 0;
        this->events = listEvents;
    }
```

8.5 include/core/Array.hpp File Reference

```
#include "Random.h"
#include "core/Vector.h"
#include "draw/SquareInfo.hpp"
#include "libScene/Highlighter.hpp"
#include "libScene/ControlMenu.hpp"
#include "EventAnimation.hpp"
```

Classes

- class Array

8.6 Array.hpp

Go to the documentation of this file.

```
//
// Created by dirii on 28/04/2023.
//

#ifndef VISUALGO_CS162_ARRAY_HPP
#define VISUALGO_CS162_ARRAY_HPP

#include "Random.h"
#include "core/Vector.h"
#include "draw/SquareInfo.hpp"
#include "libScene/Highlighter.hpp"
#include "libScene/ControlMenu.hpp"
#include "EventAnimation.hpp"

class Array : public BaseDraw{
public:
    enum class TypeArray{
        DYNAMIC,
        STATIC
    };

    Array(sf::RenderWindow *window, TypeArray typeArray);
    Array(sf::RenderWindow *window, TypeArray typeArray, int size);
    Array(sf::RenderWindow *window, TypeArray typeArray, std::vector<std::string> values);
    void init(TypeArray typeArray);
    ~Array() = default;
    void render() override;
    void renderHighlighter();
    void update();

    void setSpeed(float speed);
    int findValue(const std::string& value);

    void updateAnimation();
    void resetEvents();

    [[nodiscard]] int getSize() const;
    [[nodiscard]] int getSquaresSize() const;

    void processControlMenu(ControlMenu::StatusCode status);

    // operations of highlighter
    void initHighlighter(int linesCount, const char *codePath);
    void toggleLines(std::vector<int> lines);

    // operations of array
    void createArray(int size);
    void createArray(const std::vector<std::string>& values);
    void allocateSquare(int size, const std::vector<EventAnimation>& listEvents);
    void addSquare(int position, std::string value, const std::vector<EventAnimation>& listEvents);
    void deleteSquare(int position, const std::vector<EventAnimation>& listEvents);
    void updateSquare(int position, std::string value, const std::vector<EventAnimation>& listEvents);
    void searchSquare(const std::vector<EventAnimation>& listEvents);

private:
    sf::Clock clock;
    int chosenNode = 0, deletedNode = -1;
    TypeArray typeArray;

    Vector<SquareInfo*> squares, squaresTemp;
    int size;

    Highlighter* highlighter;

    std::vector<EventAnimation> events;
    int currentEvent = 0;

    float speed, delayTime;
    bool isDelay = false;
};

#endif //VISUALGO_CS162_ARRAY_HPP
```

8.7 include/core/EventAnimation.cpp File Reference

```
#include "EventAnimation.hpp"
```

8.8 EventAnimation.cpp

Go to the documentation of this file.

```
//
// Created by dirii on 16/04/2023.
//

#include "EventAnimation.hpp"

EventAnimation::EventAnimation() {
    this->statusChosenNode = NodeInfo::StatusNode::InChain;
    this->isPrintPreVal = this->isPrintNormal = this->isShowBackArrow = false;
    this->indexBackArrow = {-1, -1};

    this->titleNodes = {};
    this->colorArrows = {};
    this->hiddenArrows = {};
    this->colorNodes = {};
    this->lines = {};

    this->eventSquares = {};
    this->eventSquaresTemp = {};
}

void EventAnimation::reset() {
    this->titleNodes.clear();
    this->colorArrows.clear();
    this->hiddenArrows.clear();
    this->colorNodes.clear();
    this->lines.clear();

    this->statusChosenNode = NodeInfo::StatusNode::InChain;
    this->isPrintPreVal = this->isPrintNormal = this->isShowBackArrow = false;
    this->indexBackArrow = {-1, -1};

    this->eventSquares.clear();
    this->eventSquaresTemp.clear();
}

EventAnimation::~EventAnimation() {
    this->titleNodes = {};
    this->colorArrows = {};
    this->hiddenArrows = {};
    this->colorNodes = {};
    this->lines = {};

    this->eventSquares = {};
    this->eventSquaresTemp = {};
}

EventAnimation &EventAnimation::operator=(const EventAnimation &other) = default;
```

8.9 include/core/EventAnimation.hpp File Reference

```
#include "draw/NodeInfo.hpp"
#include "draw/SquareInfo.hpp"
```

Classes

- struct EventSquare
- class EventAnimation

8.10 EventAnimation.hpp

Go to the documentation of this file.

```
//
```

```
// Created by dirii on 16/04/2023.
//

#ifndef VISUALGO_CS162_EVENTANIMATION_HPP
#define VISUALGO_CS162_EVENTANIMATION_HPP

#include "draw/NodeInfo.hpp"
#include "draw/SquareInfo.hpp"

struct EventSquare{
    Square::Status status = Square::Status::inactive;
    bool isPrintPreVal = false;
    std::string title{};

    EventSquare() = default;
    ~EventSquare() = default;
};

class EventAnimation{
public:
    // for linked list
    std::vector<std::pair<int, std::string» titleNodes;
    std::vector<std::pair<int, NodeInfo::ArrowType» colorArrows;
    std::vector<std::pair<int, NodeInfo::ArrowType» hiddenArrows;
    std::vector<int> colorNodes;
    NodeInfo::StatusNode statusChosenNode;
    bool isPrintPreVal, isPrintNormal, isShowBackArrow;
    std::pair<int, int> indexBackArrow;

    // for array
    std::vector<EventSquare> eventSquares{}, eventSquaresTemp{};

    std::vector<int> lines;

    EventAnimation();
    ~EventAnimation();

    EventAnimation& operator=(const EventAnimation& other);

    void reset();
};

#endif //VISUALGO_CS162_EVENTANIMATION_HPP
```

8.11 include/core/FileDialog.h File Reference

```
#include <cstdio>
#include <cstdlib>
#include <fcntl.h>
#include <unistd.h>
#include <csignal>
#include <sys/stat.h>
#include <sys/wait.h>
#include <pwd.h>
#include <string>
#include <memory>
#include <iostream>
#include <map>
#include <set>
#include <regex>
#include <thread>
#include <chrono>
```

Classes

- class pfd::settings
- class pfd::internal::executor

- class pfd::internal::platform
- class pfd::internal::dialog
- class pfd::internal::file_dialog
- class pfd::path
- class pfd::notify
- class pfd::message
- class pfd::open_file
- class pfd::save_file
- class pfd::select_folder

Namespaces

- namespace pfd
- namespace pfd::internal

Macros

- #define _POSIX_C_SOURCE 2
- #define PFD_HAS_IFILEDIALOG 1
- #define PFD_OSX_ICON(n)

Enumerations

- enum class pfd::button {
  pfd::cancel = -1 , pfd::ok , pfd::yes , pfd::no ,
  pfd::abort , pfd::retry , pfd::ignore }
- enum class pfd::choice {
  pfd::ok = 0 , pfd::ok_cancel , pfd::yes_no , pfd::yes_no_cancel ,
  pfd::retry_cancel , pfd::abort_retry_ignore }
- enum class pfd::icon { pfd::info = 0 , pfd::warning , pfd::error , pfd::question }
- enum class pfd::opt : uint8_t { pfd::none = 0 , pfd::multiselect = 0x1 , pfd::force_overwrite = 0x2 , pfd::force_path = 0x4 }

Functions

- opt pfd::operator| (opt a, opt b)
- bool pfd::operator& (opt a, opt b)
- std::ostream & pfd::operator<< (std::ostream &s, std::vector< std::string > const &v)

8.11.1 Macro Definition Documentation

8.11.1.1 _POSIX_C_SOURCE

```
#define _POSIX_C_SOURCE 2
```

Definition at line 33 of file FileDialog.h.

8.11.1.2 PFD_HAS_IFILEDIALOG

```
#define PFD_HAS_IFILEDIALOG 1
```

Definition at line 61 of file FileDialog.h.

8.11.1.3 PFD_OSX_ICON

```
#define PFD_OSX_ICON(
                n )
```

Value:

```
"alias ((path to library folder from system domain) as text " \
"& \"CoreServices:CoreTypes.bundle:Contents:Resources:" n ".icns\")"
```

8.12 FileDialog.h

Go to the documentation of this file.

```
//
//  Portable File Dialogs
//
//  Copyright l' 20182022 Sam Hocevar <sam@hocevar.net>
//
//  This library is free software. It comes without any warranty, to
//  the extent permitted by applicable law. You can redistribute it
//  and/or modify it under the terms of the Do What the Fuck You Want
//  to Public License, Version 2, as published by the WTFPL Task Force.
//  See http://www.wtfpl.net/ for more details.
//

#pragma once

#if _WIN32
#ifndef WIN32_LEAN_AND_MEAN
#   define WIN32_LEAN_AND_MEAN 1
#endif
#include <windows.h>
#include <commdlg.h>
#include <shlobj.h>
#include <shobjidl.h> // IFileDialog
#include <shellapi.h>
#include <strsafe.h>
#include <future>     // std::async
#include <userenv.h>  // GetUserProfileDirectory()

#elif __EMSCRIPTEN__
#include <emscripten.h>

#else
#ifndef _POSIX_C_SOURCE
#   define _POSIX_C_SOURCE 2 // for popen()
#endif
#ifdef __APPLE__
#   ifndef _DARWIN_C_SOURCE
#       define _DARWIN_C_SOURCE
#   endif
#endif
#include <cstdio>     // popen()
#include <cstdlib>    // std::getenv()
#include <fcntl.h>    // fcntl()
#include <unistd.h>   // read(), pipe(), dup2(), getuid()
#include <csignal>    // ::kill, std::signal
#include <sys/stat.h> // stat()
#include <sys/wait.h> // waitpid()
#include <pwd.h>      // getpwnam()
#endif

#include <string>   // std::string
#include <memory>   // std::shared_ptr
```

```
#include <iostream> // std::ostream
#include <map>      // std::map
#include <set>      // std::set
#include <regex>    // std::regex
#include <thread>   // std::mutex, std::this_thread
#include <chrono>   // std::chrono

// Versions of mingw64 g++ up to 9.3.0 do not have a complete IFileDialog
#ifndef PFD_HAS_IFILEDIALOG
#   define PFD_HAS_IFILEDIALOG 1
#   if (defined __MINGW64__ || defined __MINGW32__) && defined __GXX_ABI_VERSION
#       if __GXX_ABI_VERSION <= 1013
#           undef PFD_HAS_IFILEDIALOG
#           define PFD_HAS_IFILEDIALOG 0
#       endif
#   endif
#endif

namespace pfd
{

    enum class button
    {
        cancel = -1,
        ok,
        yes,
        no,
        abort,
        retry,
        ignore,
    };

    enum class choice
    {
        ok = 0,
        ok_cancel,
        yes_no,
        yes_no_cancel,
        retry_cancel,
        abort_retry_ignore,
    };

    enum class icon
    {
        info = 0,
        warning,
        error,
        question,
    };

// Additional option flags for various dialog constructors
    enum class opt : uint8_t
    {
        none = 0,
        // For file open, allow multiselect.
        multiselect     = 0x1,
        // For file save, force overwrite and disable the confirmation dialog.
        force_overwrite = 0x2,
        // For folder select, force path to be the provided argument instead
        // of the last opened directory, which is the Microsoft-recommended,
        // user-friendly behaviour.
        force_path      = 0x4,
    };

    inline opt operator |(opt a, opt b) { return opt(uint8_t(a) | uint8_t(b)); }
    inline bool operator &(opt a, opt b) { return bool(uint8_t(a) & uint8_t(b)); }

// The settings class, only exposing to the user a way to set verbose mode
// and to force a rescan of installed desktop helpers (zenity, kdialog).
    class settings
    {
    public:
        static bool available();

        static void verbose(bool value);
        static void rescan();

    protected:
        explicit settings(bool resync = false);

        bool check_program(std::string const &program);

        inline bool is_osascript() const;
        inline bool is_zenity() const;
        inline bool is_kdialog() const;

        enum class flag
        {
            is_scanned = 0,
            is_verbose,

            has_zenity,
            has_matedialog,
            has_qarma,
            has_kdialog,
            is_vista,

            max_flag,
        };

        // Static array of flags for internal state
        bool const &flags(flag in_flag) const;

        // Non-const getter for the static array of flags
        bool &flags(flag in_flag);
    };

// Internal classes, not to be used by client applications
    namespace internal
    {

// Process wait timeout, in milliseconds
        static int const default_wait_timeout = 20;

        class executor
        {
            friend class dialog;

        public:
            // High level function to get the result of a command
            std::string result(int *exit_code = nullptr);

            // High level function to abort
            bool kill();

#if _WIN32
            void start_func(std::function<std::string(int *)> const &fun);
    static BOOL CALLBACK enum_windows_callback(HWND hwnd, LPARAM lParam);
#elif __EMSCRIPTEN__
            void start(int exit_code);
#else
            void start_process(std::vector<std::string> const &command);
#endif

            ~executor();

        protected:
            bool ready(int timeout = default_wait_timeout);
            void stop();

        private:
            bool m_running = false;
            std::string m_stdout;
            int m_exit_code = -1;
#if _WIN32
            std::future<std::string> m_future;
    std::set<HWND> m_windows;
    std::condition_variable m_cond;
    std::mutex m_mutex;
    DWORD m_tid;
#elif __EMSCRIPTEN__ || __NX__
            // FIXME: do something
#else
            pid_t m_pid = 0;
            int m_fd = -1;
#endif
        };

        class platform
        {
        protected:
#if _WIN32
            // Helper class around LoadLibraryA() and GetProcAddress() with some safety
    class dll
    {
    public:
        dll(std::string const &name);
        ~dll();

        template<typename T> class proc
        {
        public:
            proc(dll const &lib, std::string const &sym)
              : m_proc(reinterpret_cast<T *>((void *)::GetProcAddress(lib.handle, sym.c_str())))
            {}

            operator bool() const { return m_proc != nullptr; }
            operator T *() const { return m_proc; }

        private:
            T *m_proc;
        };

    private:
        HMODULE handle;
    };

    // Helper class around CoInitialize() and CoUnInitialize()
    class ole32_dll : public dll
    {
    public:
        ole32_dll();
        ~ole32_dll();
        bool is_initialized();

    private:
        HRESULT m_state;
    };

    // Helper class around CreateActCtx() and ActivateActCtx()
    class new_style_context
    {
    public:
        new_style_context();
        ~new_style_context();

    private:
        HANDLE create();
        ULONG_PTR m_cookie = 0;
    };
#endif
        };

        class dialog : protected settings, protected platform
        {
        public:
            bool ready(int timeout = default_wait_timeout) const;
            bool kill() const;

        protected:
            explicit dialog();

            std::vector<std::string> desktop_helper() const;
            static std::string buttons_to_name(choice _choice);
            static std::string get_icon_name(icon _icon);

            std::string powershell_quote(std::string const &str) const;
            std::string osascript_quote(std::string const &str) const;
            std::string shell_quote(std::string const &str) const;

            // Keep handle to executing command
            std::shared_ptr<executor> m_async;
        };

        class file_dialog : public dialog
        {
        protected:
            enum type
            {
                open,
                save,
                folder,
            };

            file_dialog(type in_type,
                        std::string const &title,
                        std::string const &default_path = "",
                        std::vector<std::string> const &filters = {},
                        opt options = opt::none);

        protected:
            std::string string_result();
            std::vector<std::string> vector_result();

#if _WIN32
            static int CALLBACK bffcallback(HWND hwnd, UINT uMsg, LPARAM, LPARAM pData);
#if PFD_HAS_IFILEDIALOG
    std::string select_folder_vista(IFileDialog *ifd, bool force_path);
#endif

    std::wstring m_wtitle;
    std::wstring m_wdefault_path;

    std::vector<std::string> m_vector_result;
#endif
        };

    } // namespace internal

//
// The path class provides some platform-specific path constants
//

    class path : protected internal::platform
    {
    public:
        static std::string home();
        static std::string separator();
    };

//
// The notify widget
//

    class notify : public internal::dialog
    {
    public:
        notify(std::string const &title,
               std::string const &message,
               icon _icon = icon::info);
    };

//
// The message widget
//

    class message : public internal::dialog
    {
    public:
        message(std::string const &title,
                std::string const &text,
                choice _choice = choice::ok_cancel,
                icon _icon = icon::info);

        button result();

    private:
        // Some extra logic to map the exit code to button number
        std::map<int, button> m_mappings;
    };

//
// The open_file, save_file, and open_folder widgets
//

    class open_file : public internal::file_dialog
    {
    public:
        open_file(std::string const &title,
                  std::string const &default_path = "",
                  std::vector<std::string> const &filters = { "All Files", "*" },
                  opt options = opt::none);

#if defined(__has_cpp_attribute)
#if __has_cpp_attribute(deprecated)
        // Backwards compatibility
    [[deprecated("Use pfd::opt::multiselect instead of allow_multiselect")]]
#endif
#endif
        open_file(std::string const &title,
                  std::string const &default_path,
                  std::vector<std::string> const &filters,
                  bool allow_multiselect);

        std::vector<std::string> result();
    };

    class save_file : public internal::file_dialog
    {
    public:
        save_file(std::string const &title,
                  std::string const &default_path = "",
                  std::vector<std::string> const &filters = { "All Files", "*" },
                  opt options = opt::none);

#if defined(__has_cpp_attribute)
#if __has_cpp_attribute(deprecated)
        // Backwards compatibility
    [[deprecated("Use pfd::opt::force_overwrite instead of confirm_overwrite")]]
#endif
#endif
        save_file(std::string const &title,
                  std::string const &default_path,
                  std::vector<std::string> const &filters,
                  bool confirm_overwrite);

        std::string result();
    };

    class select_folder : public internal::file_dialog
    {
    public:
        select_folder(std::string const &title,
                      std::string const &default_path = "",
                      opt options = opt::none);

        std::string result();
    };

//
// Below this are all the method implementations. You may choose to define the
// macro PFD_SKIP_IMPLEMENTATION everywhere before including this header except
// in one place. This may reduce compilation times.
//

#if !defined PFD_SKIP_IMPLEMENTATION

// internal free functions implementations

    namespace internal
    {

#if _WIN32
        static inline std::wstring str2wstr(std::string const &str)
{
    int len = MultiByteToWideChar(CP_UTF8, 0, str.c_str(), (int)str.size(), nullptr, 0);
    std::wstring ret(len, '\0');
    MultiByteToWideChar(CP_UTF8, 0, str.c_str(), (int)str.size(), (LPWSTR)ret.data(),
(int)ret.size());
    return ret;
}

static inline std::string wstr2str(std::wstring const &str)
{
    int len = WideCharToMultiByte(CP_UTF8, 0, str.c_str(), (int)str.size(), nullptr, 0, nullptr,
nullptr);
    std::string ret(len, '\0');
    WideCharToMultiByte(CP_UTF8, 0, str.c_str(), (int)str.size(), (LPSTR)ret.data(), (int)ret.size(),
nullptr, nullptr);
    return ret;
}

static inline bool is_vista()
{
    OSVERSIONINFOEXW osvi;
    memset(&osvi, 0, sizeof(osvi));
    DWORDLONG const mask = VerSetConditionMask(
            VerSetConditionMask(
                    VerSetConditionMask(
                            0, VER_MAJORVERSION, VER_GREATER_EQUAL),
                    VER_MINORVERSION, VER_GREATER_EQUAL),
            VER_SERVICEPACKMAJOR, VER_GREATER_EQUAL);
    osvi.dwOSVersionInfoSize = sizeof(osvi);
    osvi.dwMajorVersion = HIBYTE(_WIN32_WINNT_VISTA);
    osvi.dwMinorVersion = LOBYTE(_WIN32_WINNT_VISTA);
    osvi.wServicePackMajor = 0;

    return VerifyVersionInfoW(&osvi, VER_MAJORVERSION | VER_MINORVERSION | VER_SERVICEPACKMAJOR, mask)
!= FALSE;
}
#endif

// This is necessary until C++20 which will have std::string::ends_with() etc.

        static inline bool ends_with(std::string const &str, std::string const &suffix)
        {
            return suffix.size() <= str.size() &&
                   str.compare(str.size() - suffix.size(), suffix.size(), suffix) == 0;
        }

        static inline bool starts_with(std::string const &str, std::string const &prefix)
        {
            return prefix.size() <= str.size() &&
                   str.compare(0, prefix.size(), prefix) == 0;
        }
```

```

// This is necessary until C++17 which will have std::filesystem::is_directory

        static inline bool is_directory(std::string const &path)
        {
#if _WIN32
            auto attr = GetFileAttributesA(path.c_str());
    return attr != INVALID_FILE_ATTRIBUTES && (attr & FILE_ATTRIBUTE_DIRECTORY);
#elif __EMSCRIPTEN__
            // TODO
    return false;
#else
            struct stat s;
            return stat(path.c_str(), &s) == 0 && S_ISDIR(s.st_mode);
#endif
        }

// This is necessary because getenv is not thread-safe

        static inline std::string getenv(std::string const &str)
        {
#if _MSC_VER
            char *buf = nullptr;
    size_t size = 0;
    if (_dupenv_s(&buf, &size, str.c_str()) == 0 && buf)
    {
        std::string ret(buf);
        free(buf);
        return ret;
    }
    return "";
#else
            auto buf = std::getenv(str.c_str());
            return buf ? buf : "";
#endif
        }

    } // namespace internal

// settings implementation

    inline settings::settings(bool resync)
    {
        flags(flag::is_scanned) &= !resync;

        if (flags(flag::is_scanned))
            return;

        auto pfd_verbose = internal::getenv("PFD_VERBOSE");
        auto match_no = std::regex("(|0|no|false)", std::regex_constants::icase);
        if (!std::regex_match(pfd_verbose, match_no))
            flags(flag::is_verbose) = true;

#if _WIN32
        flags(flag::is_vista) = internal::is_vista();
#elif !__APPLE__
        flags(flag::has_zenity) = check_program("zenity");
        flags(flag::has_matedialog) = check_program("matedialog");
        flags(flag::has_qarma) = check_program("qarma");
        flags(flag::has_kdialog) = check_program("kdialog");

        // If multiple helpers are available, try to default to the best one
        if (flags(flag::has_zenity) && flags(flag::has_kdialog))
        {
            auto desktop_name = internal::getenv("XDG_SESSION_DESKTOP");
            if (desktop_name == std::string("gnome"))
                flags(flag::has_kdialog) = false;
            else if (desktop_name == std::string("KDE"))
                flags(flag::has_zenity) = false;
        }
#endif

        flags(flag::is_scanned) = true;
    }

    inline bool settings::available()
    {
#if _WIN32
        return true;
#elif __APPLE__
        return true;
#elif __EMSCRIPTEN__
        // FIXME: Return true after implementation is complete.
    return false;
#else
        settings tmp;
        return tmp.flags(flag::has_zenity) ||
                tmp.flags(flag::has_matedialog) ||
                tmp.flags(flag::has_qarma) ||
                tmp.flags(flag::has_kdialog);
#endif
    }

    inline void settings::verbose(bool value)
    {
        settings().flags(flag::is_verbose) = value;
    }

    inline void settings::rescan()
    {
        settings(/* resync = */ true);
    }

// Check whether a program is present using which.
    inline bool settings::check_program(std::string const &program)
    {
#if _WIN32
        (void)program;
    return false;
#elif __EMSCRIPTEN__
        (void)program;
    return false;
#else
        int exit_code = -1;
        internal::executor async;
        async.start_process({"/bin/sh", "-c", "which " + program});
        async.result(&exit_code);
        return exit_code == 0;
#endif
    }

    inline bool settings::is_osascript() const
    {
#if __APPLE__
        return true;
#else
        return false;
#endif
    }

    inline bool settings::is_zenity() const
    {
        return flags(flag::has_zenity) ||
               flags(flag::has_matedialog) ||
               flags(flag::has_qarma);
    }

    inline bool settings::is_kdialog() const
    {
        return flags(flag::has_kdialog);
    }

    inline bool const &settings::flags(flag in_flag) const
    {
        static bool flags[size_t(flag::max_flag)];
        return flags[size_t(in_flag)];
    }

    inline bool &settings::flags(flag in_flag)
    {
        return const_cast<bool &>(static_cast<settings const *>(this)->flags(in_flag));
    }

// path implementation
    inline std::string path::home()
    {
#if _WIN32
        // First try the USERPROFILE environment variable
    auto user_profile = internal::getenv("USERPROFILE");
    if (user_profile.size() > 0)
        return user_profile;
    // Otherwise, try GetUserProfileDirectory()
    HANDLE token = nullptr;
    DWORD len = MAX_PATH;
    char buf[MAX_PATH] = { '\0' };
    if (OpenProcessToken(GetCurrentProcess(), TOKEN_QUERY, &token))
    {
        dll userenv("userenv.dll");
        dll::proc<BOOL WINAPI (HANDLE, LPSTR, LPDWORD)> get_user_profile_directory(userenv,
"GetUserProfileDirectoryA");
        get_user_profile_directory(token, buf, &len);
        CloseHandle(token);
        if (*buf)
            return buf;
    }
#elif __EMSCRIPTEN__
        return "/";
#else
        // First try the HOME environment variable
        auto home = internal::getenv("HOME");
        if (home.size() > 0)
            return home;
        // Otherwise, try getpwuid_r()
        size_t len = 4096;
#if defined(_SC_GETPW_R_SIZE_MAX)
        auto size_max = sysconf(_SC_GETPW_R_SIZE_MAX);
        if (size_max != -1)
            len = size_t(size_max);
#endif
        std::vector<char> buf(len);
        struct passwd pwd, *result;
        if (getpwuid_r(getuid(), &pwd, buf.data(), buf.size(), &result) == 0)
            return result->pw_dir;
#endif
        return "/";
    }

    inline std::string path::separator()
    {
#if _WIN32
        return "\\";
#else
        return "/";
#endif
    }

// executor implementation

    inline std::string internal::executor::result(int *exit_code /* = nullptr */)
    {
        stop();
        if (exit_code)
            *exit_code = m_exit_code;
        return m_stdout;
    }

    inline bool internal::executor::kill()
    {
#if _WIN32
        if (m_future.valid())
    {
        // Close all windows that werent open when we started the future
        auto previous_windows = m_windows;
        EnumWindows(&enum_windows_callback, (LPARAM)this);
        for (auto hwnd : m_windows)
            if (previous_windows.find(hwnd) == previous_windows.end())
            {
                SendMessage(hwnd, WM_CLOSE, 0, 0);
                // Also send IDNO in case of a Yes/No or Abort/Retry/Ignore messagebox
                SendMessage(hwnd, WM_COMMAND, IDNO, 0);
            }
    }
#elif __EMSCRIPTEN__ || __NX__
        // FIXME: do something
    return false; // cannot kill
#else
        ::kill(m_pid, SIGKILL);
#endif
        stop();
        return true;
    }

#if _WIN32
    inline BOOL CALLBACK internal::executor::enum_windows_callback(HWND hwnd, LPARAM lParam)
{
    auto that = (executor *)lParam;

    DWORD pid;
    auto tid = GetWindowThreadProcessId(hwnd, &pid);
    if (tid == that->m_tid)
        that->m_windows.insert(hwnd);
    return TRUE;
}
#endif

#if _WIN32
    inline void internal::executor::start_func(std::function<std::string(int *)> const &fun)
{
    stop();

    auto trampoline = [fun, this]()
    {
        // Save our thread id so that the caller can cancel us
        m_tid = GetCurrentThreadId();
        EnumWindows(&enum_windows_callback, (LPARAM)this);
        m_cond.notify_all();
        return fun(&m_exit_code);
    };

    std::unique_lock<std::mutex> lock(m_mutex);
    m_future = std::async(std::launch::async, trampoline);
    m_cond.wait(lock);
    m_running = true;
}

#elif __EMSCRIPTEN__
    inline void internal::executor::start(int exit_code)
{
    m_exit_code = exit_code;
}

#else
    inline void internal::executor::start_process(std::vector<std::string> const &command)
    {
        stop();
        m_stdout.clear();
        m_exit_code = -1;

        int in[2], out[2];
        if (pipe(in) != 0 || pipe(out) != 0)
            return;

        m_pid = fork();
        if (m_pid < 0)
            return;

        close(in[m_pid ? 0 : 1]);
        close(out[m_pid ? 1 : 0]);

        if (m_pid == 0)
        {
            dup2(in[0], STDIN_FILENO);
            dup2(out[1], STDOUT_FILENO);

            // Ignore stderr so that it doesnt pollute the console (e.g. GTK+ errors from zenity)
            int fd = open("/dev/null", O_WRONLY);
            dup2(fd, STDERR_FILENO);
            close(fd);

            std::vector<char *> args;
            std::transform(command.cbegin(), command.cend(), std::back_inserter(args),
                           [](std::string const &s) { return const_cast<char *>(s.c_str()); });
            args.push_back(nullptr); // null-terminate argv[]

            execvp(args[0], args.data());
            exit(1);
        }

        close(in[1]);
        m_fd = out[0];
        auto flags = fcntl(m_fd, F_GETFL);
        fcntl(m_fd, F_SETFL, flags | O_NONBLOCK);

        m_running = true;
    }
#endif

    inline internal::executor::~executor()
    {
        stop();
    }

    inline bool internal::executor::ready(int timeout /* = default_wait_timeout */)
    {
        if (!m_running)
            return true;

#if _WIN32
        if (m_future.valid())
    {
        auto status = m_future.wait_for(std::chrono::milliseconds(timeout));
        if (status != std::future_status::ready)
        {
            // On Windows, we need to run the message pump. If the async
            // thread uses a Windows API dialog, it may be attached to the
            // main thread and waiting for messages that only we can dispatch.
            MSG msg;
            while (PeekMessage(&msg, nullptr, 0, 0, PM_REMOVE))
            {
                TranslateMessage(&msg);
                DispatchMessage(&msg);
            }
            return false;
        }

        m_stdout = m_future.get();
    }
#elif __EMSCRIPTEN__ || __NX__
        // FIXME: do something
    (void)timeout;
#else
        char buf[BUFSIZ];
        ssize_t received = read(m_fd, buf, BUFSIZ); // Flawfinder: ignore
        if (received > 0)
        {
            m_stdout += std::string(buf, received);
            return false;
        }

        // Reap child process if it is dead. It is possible that the system has already reaped it
        // (this happens when the calling application handles or ignores SIG_CHLD) and results in
        // waitpid() failing with ECHILD. Otherwise we assume the child is running and we sleep for
        // a little while.
        int status;
        pid_t child = waitpid(m_pid, &status, WNOHANG);
        if (child != m_pid && (child >= 0 || errno != ECHILD))
        {
            // FIXME: this happens almost always at first iteration
            std::this_thread::sleep_for(std::chrono::milliseconds(timeout));
            return false;
        }

        close(m_fd);
        m_exit_code = WEXITSTATUS(status);
#endif

        m_running = false;
        return true;
    }

    inline void internal::executor::stop()
    {
        // Loop until the user closes the dialog
        while (!ready())
            ;
    }

// dll implementation

#if _WIN32
    inline internal::platform::dll::dll(std::string const &name)
  : handle(::LoadLibraryA(name.c_str()))
{}

inline internal::platform::dll::~dll()
{
    if (handle)
        ::FreeLibrary(handle);
}
#endif // _WIN32

// ole32_dll implementation

#if _WIN32
    inline internal::platform::ole32_dll::ole32_dll()
    : dll("ole32.dll")
{
    // Use COINIT_MULTITHREADED because COINIT_APARTMENTTHREADED causes crashes.
    // See https://github.com/samhocevar/portable-file-dialogs/issues/51
    auto coinit = proc<HRESULT WINAPI (LPVOID, DWORD)>(*this, "CoInitializeEx");
    m_state = coinit(nullptr, COINIT_MULTITHREADED);
}

inline internal::platform::ole32_dll::~ole32_dll()
{
    if (is_initialized())
        proc<void WINAPI ()>(*this, "CoUninitialize")();
}

inline bool internal::platform::ole32_dll::is_initialized()
{
    return m_state == S_OK || m_state == S_FALSE;
}
#endif

```

```
// new_style_context implementation

#if _WIN32
    inline internal::platform::new_style_context::new_style_context()
{
    // Only create one activation context for the whole app lifetime.
    static HANDLE hctx = create();

    if (hctx != INVALID_HANDLE_VALUE)
        ActivateActCtx(hctx, &m_cookie);
}

inline internal::platform::new_style_context::~new_style_context()
{
    DeactivateActCtx(0, m_cookie);
}

inline HANDLE internal::platform::new_style_context::create()
{
    // This hack seems to be necessary for this code to work on windows XP.
    // Without it, dialogs do not show and close immediately. GetError()
    // returns 0 so I dont know what causes this. I was not able to reproduce
    // this behavior on Windows 7 and 10 but just in case, let it be here for
    // those versions too.
    // This hack is not required if other dialogs are used (they load comdlg32
    // automatically), only if message boxes are used.
    dll comdlg32("comdlg32.dll");

    // Using approach as shown here: https://stackoverflow.com/a/10444161
    UINT len = ::GetSystemDirectoryA(nullptr, 0);
    std::string sys_dir(len, '\0');
    ::GetSystemDirectoryA(&sys_dir[0], len);

    ACTCTXA act_ctx =
    {
        // Do not set flag ACTCTX_FLAG_SET_PROCESS_DEFAULT, since it causes a
        // crash with error default context is already set.
        sizeof(act_ctx),
        ACTCTX_FLAG_RESOURCE_NAME_VALID | ACTCTX_FLAG_ASSEMBLY_DIRECTORY_VALID,
        "shell32.dll", 0, 0, sys_dir.c_str(), (LPCSTR)124, nullptr, 0,
    };

    return ::CreateActCtxA(&act_ctx);
}
#endif // _WIN32

// dialog implementation

    inline bool internal::dialog::ready(int timeout /* = default_wait_timeout */) const
    {
        return m_async->ready(timeout);
    }

    inline bool internal::dialog::kill() const
    {
        return m_async->kill();
    }

    inline internal::dialog::dialog()
            : m_async(std::make_shared<executor>())
    {
    }

    inline std::vector<std::string> internal::dialog::desktop_helper() const
    {
#if __APPLE__
        return { "osascript" };
#else
        return { flags(flag::has_zenity) ? "zenity"
                                      : flags(flag::has_matedialog) ? "matedialog"
                                                                    : flags(flag::has_qarma) ?
"qarma"
                                                                                             :
flags(flag::has_kdialog) ? "kdialog"

: "echo" };
#endif
    }

    inline std::string internal::dialog::buttons_to_name(choice _choice)
    {
        switch (_choice)
        {
            case choice::ok_cancel: return "okcancel";
            case choice::yes_no: return "yesno";
            case choice::yes_no_cancel: return "yesnocancel";
            case choice::retry_cancel: return "retrycancel";
            case choice::abort_retry_ignore: return "abortretryignore";
                /* case choice::ok: */ default: return "ok";
        }
    }

    inline std::string internal::dialog::get_icon_name(icon _icon)
    {
        switch (_icon)
        {
            case icon::warning: return "warning";
            case icon::error: return "error";
            case icon::question: return "question";
                // Zenity wants "information" but WinForms wants "info"
                /* case icon::info: */ default:
#if _WIN32
                return "info";
#else
                return "information";
#endif
        }
    }

// This is only used for debugging purposes
    inline std::ostream& operator «(std::ostream &s, std::vector<std::string> const &v)
    {
        int not_first = 0;
        for (auto &e : v)
            s « (not_first++ ? " " : "") « e;
        return s;
    }

// Properly quote a string for Powershell: replace ’ or " with ” or ""
// FIXME: we should probably get rid of newlines!
// FIXME: the \" sequence seems unsafe, too!
// XXX: this is no longer used but I would like to keep it around just in case
    inline std::string internal::dialog::powershell_quote(std::string const &str) const
    {
        return "’" + std::regex_replace(str, std::regex("[’\"]"), "$&$&") + "’";
    }

// Properly quote a string for osascript: replace \ or " with \\ or \"
// XXX: this also used to replace ’ with \’ when popen was used, but it would be
// smarter to do shell_quote(osascript_quote(...)) if this is needed again.
    inline std::string internal::dialog::osascript_quote(std::string const &str) const
    {
        return "\"" + std::regex_replace(str, std::regex("[\\\\\"]"), "\\$&") + "\"";
    }

// Properly quote a string for the shell: just replace ’ with ’\”
// XXX: this is no longer used but I would like to keep it around just in case
    inline std::string internal::dialog::shell_quote(std::string const &str) const
    {
        return "’" + std::regex_replace(str, std::regex("’"), "’\\”") + "’";
    }

// file_dialog implementation

    inline internal::file_dialog::file_dialog(type in_type,
                                              std::string const &title,
                                              std::string const &default_path /* = "" */,
                                              std::vector<std::string> const &filters /* = {} */,
                                              opt options /* = opt::none */)
    {
#if _WIN32
        std::string filter_list;
    std::regex whitespace("  *");
    for (size_t i = 0; i + 1 < filters.size(); i += 2)
    {
        filter_list += filters[i] + ’\0’;
        filter_list += std::regex_replace(filters[i + 1], whitespace, ";") + ’\0’;
    }
    filter_list += ’\0’;

    m_async->start_func([this, in_type, title, default_path, filter_list,
                         options](int *exit_code) -> std::string
    {
        (void)exit_code;
        m_wtitle = internal::str2wstr(title);
        m_wdefault_path = internal::str2wstr(default_path);
        auto wfilter_list = internal::str2wstr(filter_list);

        // Initialise COM. This is required for the new folder selection window,
        // (see https://github.com/samhocevar/portable-file-dialogs/pull/21)
        // and to avoid random crashes with GetOpenFileNameW() (see
        // https://github.com/samhocevar/portable-file-dialogs/issues/51)
        ole32_dll ole32;

        // Folder selection uses a different method
        if (in_type == type::folder)
        {
#if PFD_HAS_IFILEDIALOG
            if (flags(flag::is_vista))
            {
                // On Vista and higher we should be able to use IFileDialog for folder selection
                IFileDialog *ifd;
                HRESULT hr = dll::proc<HRESULT WINAPI (REFCLSID, LPUNKNOWN, DWORD, REFIID, LPVOID
*)>(ole32, "CoCreateInstance")
                                 (CLSID_FileOpenDialog, nullptr, CLSCTX_INPROC_SERVER,
IID_PPV_ARGS(&ifd));

                // In case CoCreateInstance fails (which it should not), try legacy approach
                if (SUCCEEDED(hr))
                    return select_folder_vista(ifd, options & opt::force_path);
            }
#endif

            BROWSEINFOW bi;
            memset(&bi, 0, sizeof(bi));

            bi.lpfn = &bffcallback;
            bi.lParam = (LPARAM)this;

            if (flags(flag::is_vista))
            {
                if (ole32.is_initialized())
                    bi.ulFlags |= BIF_NEWDIALOGSTYLE;
                bi.ulFlags |= BIF_EDITBOX;
                bi.ulFlags |= BIF_STATUSTEXT;
            }

            auto *list = SHBrowseForFolderW(&bi);
            std::string ret;
            if (list)
            {
                auto buffer = new wchar_t[MAX_PATH];
                SHGetPathFromIDListW(list, buffer);
                dll::proc<void WINAPI (LPVOID)>(ole32, "CoTaskMemFree")(list);
                ret = internal::wstr2str(buffer);
                delete[] buffer;
            }
            return ret;
        }

        OPENFILENAMEW ofn;
        memset(&ofn, 0, sizeof(ofn));
        ofn.lStructSize = sizeof(OPENFILENAMEW);
        ofn.hwndOwner = GetActiveWindow();

        ofn.lpstrFilter = wfilter_list.c_str();

        auto woutput = std::wstring(MAX_PATH * 256, L'\0');
        ofn.lpstrFile = (LPWSTR)woutput.data();
        ofn.nMaxFile = (DWORD)woutput.size();
        if (!m_wdefault_path.empty())
        {
            // If a directory was provided, use it as the initial directory. If
            // a valid path was provided, use it as the initial file. Otherwise,
            // let the Windows API decide.
            auto path_attr = GetFileAttributesW(m_wdefault_path.c_str());
            if (path_attr != INVALID_FILE_ATTRIBUTES && (path_attr & FILE_ATTRIBUTE_DIRECTORY))
                ofn.lpstrInitialDir = m_wdefault_path.c_str();
            else if (m_wdefault_path.size() <= woutput.size())
                //second argument is size of buffer, not length of string
                StringCchCopyW(ofn.lpstrFile, MAX_PATH*256+1, m_wdefault_path.c_str());
            else
            {
                ofn.lpstrFileTitle = (LPWSTR)m_wdefault_path.data();
                ofn.nMaxFileTitle = (DWORD)m_wdefault_path.size();
            }
        }
        ofn.lpstrTitle = m_wtitle.c_str();
        ofn.Flags = OFN_NOCHANGEDIR | OFN_EXPLORER;

        dll comdlg32("comdlg32.dll");

        // Apply new visual style (required for windows XP)
        new_style_context ctx;

        if (in_type == type::save)
        {
            if (!(options & opt::force_overwrite))
                ofn.Flags |= OFN_OVERWRITEPROMPT;

            dll::proc<BOOL WINAPI (LPOPENFILENAMEW)> get_save_file_name(comdlg32, "GetSaveFileNameW");
            if (get_save_file_name(&ofn) == 0)
                return "";
            return internal::wstr2str(woutput.c_str());
        }
        else
        {
            if (options & opt::multiselect)
                ofn.Flags |= OFN_ALLOWMULTISELECT;
            ofn.Flags |= OFN_PATHMUSTEXIST;

            dll::proc<BOOL WINAPI (LPOPENFILENAMEW)> get_open_file_name(comdlg32, "GetOpenFileNameW");
            if (get_open_file_name(&ofn) == 0)
                return "";
        }

        std::string prefix;
        for (wchar_t const *p = woutput.c_str(); *p; )
        {
            auto filename = internal::wstr2str(p);
            p += wcslen(p);
            // In multiselect mode, we advance p one wchar further and
            // check for another filename. If there is one and the
            // prefix is empty, it means we just read the prefix.
            if ((options & opt::multiselect) && *++p && prefix.empty())
            {
                prefix = filename + "/";
                continue;
            }

            m_vector_result.push_back(prefix + filename);
        }

        return "";
    });
#elif __EMSCRIPTEN__
        // FIXME: do something
    (void)in_type;
    (void)title;
    (void)default_path;
    (void)filters;
    (void)options;
#else
        auto command = desktop_helper();

        if (is_osascript())
        {
            std::string script = "set ret to choose";
            switch (in_type)
            {
                case type::save:
                    script += " file name";
                    break;
                case type::open: default:
                    script += " file";
                    if (options & opt::multiselect)
                        script += " with multiple selections allowed";
                    break;
                case type::folder:
                    script += " folder";
                    break;
            }

            if (default_path.size())
            {
                if (in_type == type::folder || is_directory(default_path))
                    script += " default location ";
                else
                    script += " default name ";
                script += osascript_quote(default_path);
            }

            script += " with prompt " + osascript_quote(title);

            if (in_type == type::open)
            {
                // Concatenate all user-provided filter patterns
                std::string patterns;
                for (size_t i = 0; i < filters.size() / 2; ++i)
                    patterns += " " + filters[2 * i + 1];

                // Split the pattern list to check whether "*" is in there; if it
                // is, we have to disable filters because there is no mechanism in
                // OS X for the user to override the filter.
                std::regex sep("\\s+");
                std::string filter_list;
                bool has_filter = true;
                std::sregex_token_iterator iter(patterns.begin(), patterns.end(), sep, -1);
                std::sregex_token_iterator end;
                for ( ; iter != end; ++iter)
                {
                    auto pat = iter->str();
                    if (pat == "*" || pat == "*.*")
                        has_filter = false;
                    else if (internal::starts_with(pat, "*."))
                        filter_list += "," + osascript_quote(pat.substr(2, pat.size() - 2));
                }

                if (has_filter && filter_list.size() > 0)
                {
                    // There is a weird AppleScript bug where file extensions of length != 3 are
                    // ignored, e.g. type{"txt"} works, but type{"json"} does not. Fortunately if
                    // the whole list starts with a 3-character extension, everything works again.
                    // We use "///" for such an extension because we are sure it cannot appear in
                    // an actual filename.
                    script += " of type {\"///\"" + filter_list + "}";
                }
            }

            if (in_type == type::open && (options & opt::multiselect))
            {
                script += "\nset s to \"\"";
                script += "\nrepeat with i in ret";
                script += "\n  set s to s & (POSIX path of i) & \"\\n\"";
                script += "\nend repeat";
                script += "\ncopy s to stdout";
            }
            else
            {
                script += "\nPOSIX path of ret";
            }

            command.push_back("-e");
            command.push_back(script);
        }
        else if (is_zenity())
        {
            command.push_back("--file-selection");

            // If the default path is a directory, make sure it ends with "/" otherwise zenity will
            // open the file dialog in the parent directory.
            auto filename_arg = "--filename=" + default_path;
            if (in_type != type::folder && !ends_with(default_path, "/") &&
      internal::is_directory(default_path))
                filename_arg += "/";
            command.push_back(filename_arg);

            command.push_back("--title");
            command.push_back(title);
            command.push_back("--separator=\n");

            for (size_t i = 0; i < filters.size() / 2; ++i)
            {
                command.push_back("--file-filter");
                command.push_back(filters[2 * i] + "|" + filters[2 * i + 1]);
            }

            if (in_type == type::save)
                command.push_back("--save");
            if (in_type == type::folder)
                command.push_back("--directory");
            if (!(options & opt::force_overwrite))
                command.push_back("--confirm-overwrite");
            if (options & opt::multiselect)
                command.push_back("--multiple");
        }
        else if (is_kdialog())
        {
            switch (in_type)
            {
                case type::save: command.push_back("--getsavefilename"); break;
                case type::open: command.push_back("--getopenfilename"); break;
                case type::folder: command.push_back("--getexistingdirectory"); break;
            }
            if (options & opt::multiselect)
            {
                command.push_back("--multiple");
                command.push_back("--separate-output");
            }

            command.push_back(default_path);

            std::string filter;
            for (size_t i = 0; i < filters.size() / 2; ++i)
                filter += (i == 0 ? "" : " | ") + filters[2 * i] + "(" + filters[2 * i + 1] + ")";
            command.push_back(filter);

            command.push_back("--title");
            command.push_back(title);
        }

        if (flags(flag::is_verbose))
            std::cerr « "pfd: " « command « std::endl;

        m_async->start_process(command);
#endif
    }

    inline std::string internal::file_dialog::string_result()
    {
#if _WIN32
        return m_async->result();
#else
        auto ret = m_async->result();
        // Strip potential trailing newline (zenity). Also strip trailing slash
        // added by osascript for consistency with other backends.
        while (!ret.empty() && (ret.back() == '\n' || ret.back() == '/'))
            ret.pop_back();
        return ret;
#endif
    }

    inline std::vector<std::string> internal::file_dialog::vector_result()
    {
#if _WIN32
        m_async->result();
    return m_vector_result;
#else
        std::vector<std::string> ret;
        auto result = m_async->result();
        for (;;)
        {
            // Split result along newline characters
            auto i = result.find('\n');
            if (i == 0 || i == std::string::npos)
                break;
            ret.push_back(result.substr(0, i));
            result = result.substr(i + 1, result.size());
        }
        return ret;
#endif
    }

#if _WIN32
    // Use a static function to pass as BFFCALLBACK for legacy folder select
inline int CALLBACK internal::file_dialog::bffcallback(HWND hwnd, UINT uMsg,
                                                          LPARAM, LPARAM pData)
{
    auto inst = (file_dialog *)pData;
    switch (uMsg)
    {
        case BFFM_INITIALIZED:
            SendMessage(hwnd, BFFM_SETSELECTIONW, TRUE, (LPARAM)inst->m_wdefault_path.c_str());
            break;
    }
    return 0;
}

#if PFD_HAS_IFILEDIALOG
inline std::string internal::file_dialog::select_folder_vista(IFileDialog *ifd, bool force_path)
{
    std::string result;

    IShellItem *folder;

    // Load library at runtime so app doesn't link it at load time (which will fail on windows XP)
    dll shell32("shell32.dll");
    dll::proc<HRESULT WINAPI (PCWSTR, IBindCtx*, REFIID, void**)>
        create_item(shell32, "SHCreateItemFromParsingName");

    if (!create_item)
        return "";

    auto hr = create_item(m_wdefault_path.c_str(),
                          nullptr,
                          IID_PPV_ARGS(&folder));

    // Set default folder if found. This only sets the default folder. If
    // Windows has any info about the most recently selected folder, it
    // will display it instead. Generally, calling SetFolder() to set the
    // current directory is not a good or expected user experience and
    // should therefore be avoided:
```

```
    //
https://docs.microsoft.com/windows/win32/api/shobjidl_core/nf-shobjidl_core-ifiledialog-setfolder
    if (SUCCEEDED(hr))
    {
        if (force_path)
            ifd->SetFolder(folder);
        else
            ifd->SetDefaultFolder(folder);
        folder->Release();
    }

    // Set the dialog title and option to select folders
    ifd->SetOptions(FOS_PICKFOLDERS | FOS_FORCEFILESYSTEM);
    ifd->SetTitle(m_wtitle.c_str());

    hr = ifd->Show(GetActiveWindow());
    if (SUCCEEDED(hr))
    {
        IShellItem* item;
        hr = ifd->GetResult(&item);
        if (SUCCEEDED(hr))
        {
            wchar_t* wname = nullptr;
            // This is unlikely to fail because we use FOS_FORCEFILESYSTEM, but try
            // to output a debug message just in case.
            if (SUCCEEDED(item->GetDisplayName(SIGDN_FILESYSPATH, &wname)))
            {
                result = internal::wstr2str(std::wstring(wname));
                dll::proc<void WINAPI (LPVOID)>(ole32_dll(), "CoTaskMemFree")(wname);
            }
            else
            {
                if (SUCCEEDED(item->GetDisplayName(SIGDN_NORMALDISPLAY, &wname)))
                {
                    auto name = internal::wstr2str(std::wstring(wname));
                    dll::proc<void WINAPI (LPVOID)>(ole32_dll(), "CoTaskMemFree")(wname);
                    std::cerr « "pfd: failed to get path for " « name « std::endl;
                }
                else
                    std::cerr « "pfd: item of unknown type selected" « std::endl;
            }

            item->Release();
        }
    }

    ifd->Release();

    return result;
}
#endif
#endif

// notify implementation

    inline notify::notify(std::string const &title,
                          std::string const &message,
                          icon _icon /* = icon::info */)
    {
        if (_icon == icon::question) // Not supported by notifications
            _icon = icon::info;

#if _WIN32
        // Use a static shared pointer for notify_icon so that we can delete
    // it whenever we need to display a new one, and we can also wait
    // until the program has finished running.
    struct notify_icon_data : public NOTIFYICONDATAW
    {
        ~notify_icon_data() { Shell_NotifyIconW(NIM_DELETE, this); }
    };

    static std::shared_ptr<notify_icon_data> nid;

    // Release the previous notification icon, if any, and allocate a new
    // one. Note that std::make_shared() does value initialization, so there
    // is no need to memset the structure.
    nid = nullptr;
    nid = std::make_shared<notify_icon_data>();

    // For XP support
    nid->cbSize = NOTIFYICONDATAW_V2_SIZE;
    nid->hWnd = nullptr;
    nid->uID = 0;

    // Flag Description:
    // - NIF_ICON    The hIcon member is valid.
    // - NIF_MESSAGE The uCallbackMessage member is valid.
    // - NIF_TIP     The szTip member is valid.
    // - NIF_STATE   The dwState and dwStateMask members are valid.
    // - NIF_INFO    Use a balloon ToolTip instead of a standard ToolTip. The szInfo, uTimeout, szInfoTitle, and dwInfoFlags members are valid.
    // - NIF_GUID    Reserved.
    nid->uFlags = NIF_MESSAGE | NIF_ICON | NIF_INFO;

    // Flag Description
    // - NIIF_ERROR     An error icon.
    // - NIIF_INFO      An information icon.
    // - NIIF_NONE      No icon.
    // - NIIF_WARNING   A warning icon.
    // - NIIF_ICON_MASK Version 6.0. Reserved.
    // - NIIF_NOSOUND   Version 6.0. Do not play the associated sound. Applies only to balloon ToolTips
    switch (_icon)
    {
        case icon::warning: nid->dwInfoFlags = NIIF_WARNING; break;
        case icon::error: nid->dwInfoFlags = NIIF_ERROR; break;
        /* case icon::info: */ default: nid->dwInfoFlags = NIIF_INFO; break;
    }

    ENUMRESNAMEPROC icon_enum_callback = [](HMODULE, LPCTSTR, LPTSTR lpName, LONG_PTR lParam) -> BOOL
    {
        ((NOTIFYICONDATAW *)lParam)->hIcon = ::LoadIcon(GetModuleHandle(nullptr), lpName);
        return false;
    };

    nid->hIcon = ::LoadIcon(nullptr, IDI_APPLICATION);
    ::EnumResourceNames(nullptr, RT_GROUP_ICON, icon_enum_callback, (LONG_PTR)nid.get());

    nid->uTimeout = 5000;

    StringCchCopyW(nid->szInfoTitle, ARRAYSIZE(nid->szInfoTitle), internal::str2wstr(title).c_str());
    StringCchCopyW(nid->szInfo, ARRAYSIZE(nid->szInfo), internal::str2wstr(message).c_str());

    // Display the new icon
    Shell_NotifyIconW(NIM_ADD, nid.get());
#elif __EMSCRIPTEN__
        // FIXME: do something
    (void)title;
    (void)message;
#else
        auto command = desktop_helper();

        if (is_osascript())
        {
            command.push_back("-e");
            command.push_back("display notification " + osascript_quote(message) +
                              " with title " + osascript_quote(title));
        }
        else if (is_zenity())
        {
            command.push_back("--notification");
            command.push_back("--window-icon");
            command.push_back(get_icon_name(_icon));
            command.push_back("--text");
            command.push_back(title + "\n" + message);
        }
        else if (is_kdialog())
        {
            command.push_back("--icon");
            command.push_back(get_icon_name(_icon));
            command.push_back("--title");
            command.push_back(title);
            command.push_back("--passivepopup");
            command.push_back(message);
            command.push_back("5");
        }

        if (flags(flag::is_verbose))
            std::cerr « "pfd: " « command « std::endl;

        m_async->start_process(command);
#endif
    }

// message implementation

    inline message::message(std::string const &title,
                            std::string const &text,
                            choice _choice /* = choice::ok_cancel */,
                            icon _icon /* = icon::info */)
    {
#if _WIN32
        // Use MB_SYSTEMMODAL rather than MB_TOPMOST to ensure the message window is brought
    // to front. See https://github.com/samhocevar/portable-file-dialogs/issues/52
    UINT style = MB_SYSTEMMODAL;
    switch (_icon)
    {
        case icon::warning: style |= MB_ICONWARNING; break;
        case icon::error: style |= MB_ICONERROR; break;
        case icon::question: style |= MB_ICONQUESTION; break;
        /* case icon::info: */ default: style |= MB_ICONINFORMATION; break;
    }

    switch (_choice)
    {
        case choice::ok_cancel: style |= MB_OKCANCEL; break;
        case choice::yes_no: style |= MB_YESNO; break;
        case choice::yes_no_cancel: style |= MB_YESNOCANCEL; break;
        case choice::retry_cancel: style |= MB_RETRYCANCEL; break;
        case choice::abort_retry_ignore: style |= MB_ABORTRETRYIGNORE; break;
        /* case choice::ok: */ default: style |= MB_OK; break;
    }

    m_mappings[IDCANCEL] = button::cancel;
    m_mappings[IDOK] = button::ok;
    m_mappings[IDYES] = button::yes;
    m_mappings[IDNO] = button::no;
    m_mappings[IDABORT] = button::abort;
    m_mappings[IDRETRY] = button::retry;
    m_mappings[IDIGNORE] = button::ignore;

    m_async->start_func([text, title, style](int* exit_code) -> std::string
    {
        auto wtext = internal::str2wstr(text);
        auto wtitle = internal::str2wstr(title);
        // Apply new visual style (required for all Windows versions)
        new_style_context ctx;
        *exit_code = MessageBoxW(GetActiveWindow(), wtext.c_str(), wtitle.c_str(), style);
        return "";
    });

#elif __EMSCRIPTEN__
        std::string full_message;
    switch (_icon)
    {
        case icon::warning: full_message = ""; break;
        case icon::error: full_message = ""; break;
        case icon::question: full_message = ""; break;
        /* case icon::info: */ default: full_message = ""; break;
    }

    full_message += ' ' + title + "\n\n" + text;

    // This does not really start an async task; it just passes the
    // EM_ASM_INT return value to a fake start() function.
    m_async->start(EM_ASM_INT(
    {
        if ($1)
            return window.confirm(UTF8ToString($0)) ? 0 : -1;
        alert(UTF8ToString($0));
        return 0;
    }, full_message.c_str(), _choice == choice::ok_cancel));
#else
        auto command = desktop_helper();

        if (is_osascript())
        {
            std::string script = "display dialog " + osascript_quote(text) +
                                 " with title " + osascript_quote(title);
            auto if_cancel = button::cancel;
            switch (_choice)
            {
                case choice::ok_cancel:
                    script += "buttons {\"OK\", \"Cancel\"}"
                              " default button \"OK\""
                              " cancel button \"Cancel\"";
                    break;
                case choice::yes_no:
                    script += "buttons {\"Yes\", \"No\"}"
                              " default button \"Yes\""
                              " cancel button \"No\"";
                    if_cancel = button::no;
                    break;
                case choice::yes_no_cancel:
                    script += "buttons {\"Yes\", \"No\", \"Cancel\"}"
                              " default button \"Yes\""
                              " cancel button \"Cancel\"";
                    break;
                case choice::retry_cancel:
                    script += "buttons {\"Retry\", \"Cancel\"}"
                              " default button \"Retry\""
                              " cancel button \"Cancel\"";
                    break;
                case choice::abort_retry_ignore:
                    script += "buttons {\"Abort\", \"Retry\", \"Ignore\"}"
                              " default button \"Abort\""
                              " cancel button \"Retry\"";
                    if_cancel = button::retry;
                    break;
                case choice::ok: default:
                    script += "buttons {\"OK\"}"
                              " default button \"OK\""
                              " cancel button \"OK\"";
                    if_cancel = button::ok;
                    break;
            }
            m_mappings[1] = if_cancel;
            m_mappings[256] = if_cancel; // XXX: I think this was never correct
            script += " with icon ";
            switch (_icon)
            {
#define PFD_OSX_ICON(n) "alias ((path to library folder from system domain) as text " \
                "& \"CoreServices:CoreTypes.bundle:Contents:Resources:" n ".icns\")"
                case icon::info: default: script += PFD_OSX_ICON("ToolBarInfo"); break;
                case icon::warning: script += "caution"; break;
                case icon::error: script += "stop"; break;
                case icon::question: script += PFD_OSX_ICON("GenericQuestionMarkIcon"); break;
#undef PFD_OSX_ICON
            }

            command.push_back("-e");
            command.push_back(script);
        }
        else if (is_zenity())
        {
            switch (_choice)
            {
                case choice::ok_cancel:
                    command.insert(command.end(), { "--question", "--cancel-label=Cancel",
"--ok-label=OK" }); break;
                case choice::yes_no:
                    // Do not use standard --question because it causes No to return -1,
                    // which is inconsistent with the Yes/No/Cancel mode below.
                    command.insert(command.end(), { "--question", "--switch", "--extra-button=No",
"--extra-button=Yes" }); break;
                case choice::yes_no_cancel:
                    command.insert(command.end(), { "--question", "--switch", "--extra-button=Cancel",
"--extra-button=No", "--extra-button=Yes" }); break;
                case choice::retry_cancel:
                    command.insert(command.end(), { "--question", "--switch", "--extra-button=Cancel",
"--extra-button=Retry" }); break;
                case choice::abort_retry_ignore:
                    command.insert(command.end(), { "--question", "--switch", "--extra-button=Ignore",
"--extra-button=Abort", "--extra-button=Retry" }); break;
                case choice::ok:
                default:
                    switch (_icon)
                    {
                        case icon::error: command.push_back("--error"); break;
                        case icon::warning: command.push_back("--warning"); break;
                        default: command.push_back("--info"); break;
                    }
            }

            command.insert(command.end(), { "--title", title,
                                            "--width=300", "--height=0", // sensible defaults
                                            "--no-markup", // do not interpret text as Pango markup
                                            "--text", text,
                                            "--icon-name=dialog-" + get_icon_name(_icon) });
        }
        else if (is_kdialog())
        {
            if (_choice == choice::ok)
            {
                switch (_icon)
                {
                    case icon::error: command.push_back("--error"); break;
                    case icon::warning: command.push_back("--sorry"); break;
                    default: command.push_back("--msgbox"); break;
                }
            }
            else
            {
                std::string flag = "--";
                if (_icon == icon::warning || _icon == icon::error)
                    flag += "warning";
                flag += "yesno";
                if (_choice == choice::yes_no_cancel)
```

```
                    flag += "cancel";
                command.push_back(flag);
                if (_choice == choice::yes_no || _choice == choice::yes_no_cancel)
                {
                    m_mappings[0] = button::yes;
                    m_mappings[256] = button::no;
                }
            }

            command.push_back(text);
            command.push_back("--title");
            command.push_back(title);

            // Must be after the above part
            if (_choice == choice::ok_cancel)
                command.insert(command.end(), { "--yes-label", "OK", "--no-label", "Cancel" });
        }

        if (flags(flag::is_verbose))
            std::cerr « "pfd: " « command « std::endl;

        m_async->start_process(command);
#endif
    }

    inline button message::result()
    {
        int exit_code;
        auto ret = m_async->result(&exit_code);
        // osascript will say "button returned:Cancel\n"
        // and others will just say "Cancel\n"
        if (internal::ends_with(ret, "Cancel\n"))
            return button::cancel;
        if (internal::ends_with(ret, "OK\n"))
            return button::ok;
        if (internal::ends_with(ret, "Yes\n"))
            return button::yes;
        if (internal::ends_with(ret, "No\n"))
            return button::no;
        if (internal::ends_with(ret, "Abort\n"))
            return button::abort;
        if (internal::ends_with(ret, "Retry\n"))
            return button::retry;
        if (internal::ends_with(ret, "Ignore\n"))
            return button::ignore;
        if (m_mappings.count(exit_code) != 0)
            return m_mappings[exit_code];
        return exit_code == 0 ? button::ok : button::cancel;
    }

// open_file implementation

    inline open_file::open_file(std::string const &title,
                                std::string const &default_path /* = "" */,
                                std::vector<std::string> const &filters /* = { "All Files", "*" } */,
                                opt options /* = opt::none */)
            : file_dialog(type::open, title, default_path, filters, options)
    {
    }

    inline open_file::open_file(std::string const &title,
                                std::string const &default_path,
                                std::vector<std::string> const &filters,
                                bool allow_multiselect)
            : open_file(title, default_path, filters,
                        (allow_multiselect ? opt::multiselect : opt::none))
    {
    }

    inline std::vector<std::string> open_file::result()
    {
        return vector_result();
    }

// save_file implementation

    inline save_file::save_file(std::string const &title,
                                std::string const &default_path /* = "" */,
                                std::vector<std::string> const &filters /* = { "All Files", "*" } */,
                                opt options /* = opt::none */)
            : file_dialog(type::save, title, default_path, filters, options)
    {
    }

    inline save_file::save_file(std::string const &title,
                                std::string const &default_path,
                                std::vector<std::string> const &filters,
                                bool confirm_overwrite)
            : save_file(title, default_path, filters,
                        (confirm_overwrite ? opt::none : opt::force_overwrite))
    {
    }

    inline std::string save_file::result()
    {
        return string_result();
    }

// select_folder implementation

    inline select_folder::select_folder(std::string const &title,
                                        std::string const &default_path /* = "" */,
                                        opt options /* = opt::none */)
            : file_dialog(type::folder, title, default_path, {}, options)
    {
    }

    inline std::string select_folder::result()
    {
        return string_result();
    }

#endif // PFD_SKIP_IMPLEMENTATION

} // namespace pfd
```

8.13 include/core/LinkedList.cpp File Reference

```
#include "LinkedList.hpp"
```

8.14 LinkedList.cpp

Go to the documentation of this file.

```
//
// Created by dirii on 12/04/2023.
//

#include "LinkedList.hpp"

LinkedList::LinkedList(sf::RenderWindow* window, TypeLinkedList typeLinkedList) {
    this->window = window;
    this->typeLinkedList = typeLinkedList;
    this->highlighter = nullptr;
    this->delayTime = constants::LinkedList::DELAY_TIME;
    this->backArrow = new BackArrow(this->window, {0, 0}, {0, 0});

    if (this->typeLinkedList == TypeLinkedList::CIRCULAR)
        this->backArrow->show();
    else
        this->backArrow->hide();

    this->createLinkedList(0);
}

void LinkedList::clear() {
    for (auto &node : this->nodes)
        delete node;
    this->nodes.clear();
    this->size = 0;
}

void LinkedList::render() {
    if (this->size > 1) {
//        this->backArrow->toggleActiveColorNode();
        this->backArrow->render();
    }
    for (auto &node : this->nodes){
        node->render();
    }
}

```

```cpp
LinkedList::LinkedList(sf::RenderWindow* window, TypeLinkedList typeLinkedList, int size) {
    this->window = window;
    this->typeLinkedList = typeLinkedList;
    this->highlighter = nullptr;
    this->delayTime = constants::LinkedList::DELAY_TIME;
    this->backArrow = new BackArrow(this->window, {0, 0}, {0, 0});

    if (this->typeLinkedList == TypeLinkedList::CIRCULAR)
        this->backArrow->show();
    else
        this->backArrow->hide();

    this->createLinkedList(size);
}

LinkedList::LinkedList(sf::RenderWindow* window, TypeLinkedList typeLinkedList,
std::vector<std::string> values) {
    this->window = window;
    this->typeLinkedList = typeLinkedList;
    this->highlighter = nullptr;
    this->delayTime = constants::LinkedList::DELAY_TIME;
    this->backArrow = new BackArrow(this->window, {0, 0}, {0, 0});

    if (this->typeLinkedList == TypeLinkedList::CIRCULAR)
        this->backArrow->show();
    else
        this->backArrow->hide();

    this->createLinkedList(std::move(values));
}

void LinkedList::update() {
    if ((int)this->events.size() && (this->isDelay or this->clock.getElapsedTime().asSeconds() >
this->delayTime / this->speed))
        this->updateAnimation();
    this->isDelay = false;
}

void LinkedList::updateAnimation() {
    if (this->nodes.empty())
        return;

    // reset events of list
    for (auto &node : this->nodes){
        node->reset();
    }

    if (this->typeLinkedList == TypeLinkedList::CIRCULAR)
        this->backArrow->show();

    EventAnimation &event = this->events[this->currentEvent];
    for (auto &arrow: event.colorArrows)
        this->nodes[arrow.first]->toggleActiveColorArrow(arrow.second);
    for (auto &arrow : event.hiddenArrows)
        this->nodes[arrow.first]->hide(arrow.second);
    for (auto node : event.colorNodes)
        this->nodes[node]->toggleActiveColorNode();
    switch (event.statusChosenNode) {
        case NodeInfo::StatusNode::InChain:
            this->nodes[this->chosenNode]->setNodeInChain();
            break;
        case NodeInfo::StatusNode::OutChain:
            this->nodes[this->chosenNode]->setNodeOutside();
            break;
        case NodeInfo::StatusNode::Visible:
            this->nodes[this->chosenNode]->setNodeVisible();
            break;
    }
    if (event.isPrintPreVal)
        this->nodes[this->chosenNode]->setPrintPreVal();
    if (this->chosenNode < this->size - 1 && event.isPrintNormal)
        this->nodes[this->chosenNode + 1]->setPrintNormal();

    if (this->highlighter)
        this->highlighter->toggle(event.lines);

    this->calculateEffectivePositions();

    for (auto &node : this->nodes){
        node->updateNode();
    }

    for (auto &i : event.titleNodes) {
        this->nodes[i.first]->setTitle(i.second);
    }

    if (this->chosenNode < this->size - 1)
```

```
        this->nodes[this->chosenNode]->updateArrows(NodeInfo::ArrowType::RIGHT,
this->nodes[this->chosenNode + 1]->getPosition());
    if (this->chosenNode > 0)
        this->nodes[this->chosenNode]->updateArrows(NodeInfo::ArrowType::LEFT,
this->nodes[this->chosenNode - 1]->getPosition());

    if (event.indexBackArrow.first != -1 and event.indexBackArrow.second != -1)
        this->backArrow->setPosition(
                this->nodes[event.indexBackArrow.first]->getPosition(),
                this->nodes[event.indexBackArrow.second]->getPosition()
                );

    int lastInChain = 0;
    if (this->nodes[lastInChain]->getStatusNode() != NodeInfo::StatusNode::InChain){
        lastInChain++;
    }
    for (int i = lastInChain + 1; i < this->size; i++){
        if (this->nodes[i]->getStatusNode() == NodeInfo::StatusNode::InChain) {
            this->nodes[lastInChain]->updateArrows(NodeInfo::ArrowType::RIGHT,
this->nodes[i]->getPosition());
            this->nodes[i]->updateArrows(NodeInfo::ArrowType::LEFT,
this->nodes[lastInChain]->getPosition());
            lastInChain = i;
        }
    }
}

void LinkedList::calculateEffectivePositions() {
    if (this->size < 2) return;

    int lastInChain = 0;
    if (this->nodes[lastInChain]->getStatusNode() != NodeInfo::StatusNode::InChain){
        lastInChain++;
    }

    this->nodes[lastInChain]->setEffectivePosition(
            sf::Vector2f(
                    constants::NodeInfo::originNode.x,
                    constants::NodeInfo::originNode.y
            )
    );

    for (int i = lastInChain + 1; i < this->size; i++){
        if (this->nodes[i]->getStatusNode() == NodeInfo::StatusNode::InChain){
            this->nodes[i]->setEffectivePosition(
                    sf::Vector2f(
                            this->nodes[lastInChain]->getPosition().x + constants::NodeInfo::offsetX,
                            this->nodes[lastInChain]->getPosition().y
                    )
            );
            lastInChain = i;
        }
    }
}

void LinkedList::resetEvents() {
    delete this->highlighter;
    this->highlighter = nullptr;
    this->currentEvent = 0;
    this->events.clear();
    this->chosenNode = 0;

    if (this->deletedNode != -1){
        this->nodes.erase(this->nodes.begin() + this->deletedNode);
        --this->size;
        if (this->size && this->deletedNode == this->size)
            this->nodes.back()->destroyArrow(NodeInfo::ArrowType::RIGHT);
        if (this->size && this->deletedNode == 0)
            this->nodes[0]->destroyArrow(NodeInfo::ArrowType::LEFT);
    }
    this->deletedNode = -1;

    for (int i = 0;  i < this->size; i++){
        this->nodes[i]->reset();
        this->nodes[i]->reInitPos(i);
        this->nodes[i]->reInitPreVal();
    }
    if (this->size > 1)
        this->backArrow->setPosition(this->nodes.back()->getPosition(),
this->nodes[0]->getPosition());
}

void LinkedList::createLinkedList(int _size) {
    this->resetEvents();
    this->size = _size;
    for (auto &node : this->nodes)
        delete node;
    this->nodes.resize(_size);
    for (int i = 0; i < size; i++){
        this->nodes[i] = new NodeInfo(
                this->window,
                std::to_string(Random::randomInt(0, 99)),
                sf::Vector2f(
                        constants::NodeInfo::originNode.x + static_cast<float>(i) *
constants::NodeInfo::offsetX,
                        constants::NodeInfo::originNode.y
                ),
                this->typeLinkedList == TypeLinkedList::DOUBLY
        );
        if (i > 0){
            this->nodes[i - 1]->initArrow(
                    NodeInfo::ArrowType::RIGHT,
                    this->nodes[i - 1]->getPosition(),
                    this->nodes[i]->getPosition()
            );
            if (this->typeLinkedList == TypeLinkedList::DOUBLY)
                this->nodes[i]->initArrow(
                        NodeInfo::ArrowType::LEFT,
                        this->nodes[i]->getPosition(),
                        this->nodes[i - 1]->getPosition()
                );
        }
    }
    if (this->size > 1)
        this->backArrow->setPosition(this->nodes.back()->getPosition(),
this->nodes[0]->getPosition());
}

void LinkedList::createLinkedList(std::vector<std::string> values) {
    this->resetEvents();
    this->size = static_cast<int>(values.size());
    for (auto &node : this->nodes)
        delete node;
    this->nodes.resize(this->size);
    for (int i = 0; i < this->size; i++){
        this->nodes[i] = new NodeInfo(
                this->window,
                values[i],
                sf::Vector2f(
                        constants::NodeInfo::originNode.x + static_cast<float>(i) *
constants::NodeInfo::offsetX,
                        constants::NodeInfo::originNode.y
                ),
                this->typeLinkedList == TypeLinkedList::DOUBLY
        );
        if (i > 0){
            this->nodes[i - 1]->initArrow(
                    NodeInfo::ArrowType::RIGHT,
                    this->nodes[i - 1]->getPosition(),
                    this->nodes[i]->getPosition()
            );
            if (this->typeLinkedList == TypeLinkedList::DOUBLY)
                this->nodes[i]->initArrow(
                        NodeInfo::ArrowType::LEFT,
                        this->nodes[i]->getPosition(),
                        this->nodes[i - 1]->getPosition()
                );
        }
    }
    if (this->size > 1)
        this->backArrow->setPosition(this->nodes.back()->getPosition(),
this->nodes[0]->getPosition());
}

void LinkedList::initHighlighter(int linesCount, const char *codePath) {
    delete this->highlighter;
    this->highlighter = new Highlighter(
            this->window,
            linesCount,
            codePath
    );
}

void LinkedList::toggleLines(std::vector<int> lines) {
    this->highlighter->toggle(std::move(lines));
}

void LinkedList::renderHighlighter() {
    if (this->highlighter)
        this->highlighter->render();
}

void LinkedList::processControlMenu(ControlMenu::StatusCode status) {
    if (this->clock.getElapsedTime().asSeconds() < this->delayTime / this->speed)
```

```
        return;
    switch (status){
        case ControlMenu::StatusCode::PREVIOUS:
            if (this->currentEvent > 0)
                --this->currentEvent;
            break;
        case ControlMenu::StatusCode::PAUSE:
//            std::cout « "PAUSE" « std::endl;
            break;
        case ControlMenu::StatusCode::PLAY:
            if (this->currentEvent + 1 < this->events.size()) {
                this->isDelay = true;
                this->clock.restart();
            }
        case ControlMenu::StatusCode::NEXT:
            if (this->currentEvent + 1 < this->events.size())
                ++this->currentEvent;
            break;
        default:
            break;
    }
}

void LinkedList::setSpeed(float _speed) {
    this->speed = _speed;
}

int LinkedList::getSize() const {
    return this->size;
}

void LinkedList::addNode(int position, std::string value, const std::vector<EventAnimation>& listEvents) {
    if (position < 0 || position > this->size) return;

    sf::Vector2f newPosition(
            constants::NodeInfo::originNode.x + static_cast<float>(this->nodes.size()) * constants::NodeInfo::offsetX,
            constants::NodeInfo::originNode.y
    );
    if (this->size) {
        this->nodes.back()->initArrow(
                NodeInfo::ArrowType::RIGHT,
                this->nodes.back()->getPosition(),
                newPosition
        );
    }
    this->nodes.push_back(new NodeInfo(
            this->window,
            "10",
            newPosition,
            this->typeLinkedList == TypeLinkedList::DOUBLY
    ));
    ++this->size;
    if (this->typeLinkedList == TypeLinkedList::DOUBLY && this->size > 1)
        this->nodes.back()->initArrow(
                NodeInfo::ArrowType::LEFT,
                this->nodes.back()->getPosition(),
                this->nodes[this->nodes.size() - 2]->getPosition()
        );
    this->backArrow->setPosition(newPosition, this->nodes[0]->getPosition());
    for (int i = this->size - 1; i > position; --i) {
        this->nodes[i]->setValue(this->nodes[i - 1]->getValue());
        this->nodes[i]->reInitPreVal();
    }
    this->nodes[position]->setValue(std::move(value));
//    std::cout « "add node to the current list " « position « " " « this->nodes[position]->getValue() « std::endl;

    this->chosenNode = position;
    this->currentEvent = 0;

    for (auto &e : listEvents)
        this->events.emplace_back(e);
}

void LinkedList::deleteNode(int position, const std::vector<EventAnimation>& listEvents) {
    if (position < 0 || position >= this->size) return;

    this->deletedNode = position;
    this->chosenNode = position;
    this->currentEvent = 0;

    for (auto &e : listEvents)
        this->events.emplace_back(e);
}

```

```
void LinkedList::updateNode(int position, std::string value, const std::vector<EventAnimation>
&listEvents) {
    if (position < 0 || position >= this->size) return;

    this->nodes[position]->setValue(std::move(value));
    this->chosenNode = position;
    this->currentEvent = 0;

    for (auto &e : listEvents)
        this->events.emplace_back(e);
}

void LinkedList::searchNode(const std::vector<EventAnimation> &listEvents) {
    this->chosenNode = 0;
    this->currentEvent = 0;

    for (auto &e : listEvents)
        this->events.emplace_back(e);
}

int LinkedList::findValue(const std::string& value) {
    for (int i = 0; i < this->size; ++i)
        if (this->nodes[i]->getValue() == value)
            return i;
    return this->size;
}

sf::Vector2f LinkedList::getPosNode(int position) {
    if (position < 0 || position >= this->size) return {};
    return this->nodes[position]->getPosition();
}
```

8.15 include/core/LinkedList.hpp File Reference

```
#include "Random.h"
#include "draw/NodeInfo.hpp"
#include "draw/BackArrow.hpp"
#include "libScene/Highlighter.hpp"
#include "libScene/ControlMenu.hpp"
#include "EventAnimation.hpp"
#include "core/Vector.h"
```

Classes

- class LinkedList

8.16 LinkedList.hpp

Go to the documentation of this file.

```
//
// Created by dirii on 12/04/2023.
//

#ifndef VISUALGO_CS162_LINKEDLIST_HPP
#define VISUALGO_CS162_LINKEDLIST_HPP

#include "Random.h"
#include "draw/NodeInfo.hpp"
#include "draw/BackArrow.hpp"
#include "libScene/Highlighter.hpp"
#include "libScene/ControlMenu.hpp"
#include "EventAnimation.hpp"
#include "core/Vector.h"

class LinkedList {
public:
    enum class TypeLinkedList{
        SINGLY,
        DOUBLY,
        CIRCULAR
    };

    explicit LinkedList(sf::RenderWindow* window, TypeLinkedList typeLinkedList);
    LinkedList(sf::RenderWindow* window, TypeLinkedList typeLinkedList, int size);
    LinkedList(sf::RenderWindow* window, TypeLinkedList typeLinkedList, std::vector<std::string>
values);

    void setSpeed(float speed);
    int findValue(const std::string& value);
    sf::Vector2f getPosNode(int position);

    [[nodiscard]] int getSize() const;

    void update();
    void updateAnimation();
    void render();
    void renderHighlighter();
    void resetEvents();

    void calculateEffectivePositions();
    void clear();

    void processControlMenu(ControlMenu::StatusCode status);

    // operations of highlighter
    void initHighlighter(int linesCount, const char *codePath);
    void toggleLines(std::vector<int> lines);

    // operations of linked list
    void createLinkedList(int size);
    void createLinkedList(std::vector<std::string> values);
    void addNode(int position, std::string value, const std::vector<EventAnimation>& listEvents);
    void deleteNode(int position, const std::vector<EventAnimation>& listEvents);
    void updateNode(int position, std::string value, const std::vector<EventAnimation>& listEvents);
    void searchNode(const std::vector<EventAnimation>& listEvents);

private:
    sf::RenderWindow* window;
    sf::Clock clock;
    int chosenNode = 0, deletedNode = -1;
    TypeLinkedList typeLinkedList;

    Vector<NodeInfo*> nodes;
    int size;

    BackArrow* backArrow;

    Highlighter* highlighter;

    std::vector<EventAnimation> events;
    int currentEvent;

    float speed, delayTime;
    bool isDelay = false;
};

#endif //VISUALGO_CS162_LINKEDLIST_HPP
```

8.17 include/core/Random.h File Reference

```
#include <random>
```

Namespaces

- namespace Random

8.18 Random.h

Go to the documentation of this file.

```
//
// Created by dirii on 01/05/2023.
//

#ifndef VISUALGO_CS162_RANDOM_H
#define VISUALGO_CS162_RANDOM_H

#include <random>

namespace Random{
    static std::mt19937 rng(std::random_device{}());
    static int randomInt(int min, int max){
        std::uniform_int_distribution<int> dist(min, max);
        return dist(rng);
    }
}

#endif //VISUALGO_CS162_RANDOM_H
```

8.19 include/core/Vector.h File Reference

Classes

- class Vector< T >

8.20 Vector.h

Go to the documentation of this file.

```
//
// Created by dirii on 27/04/2023.
//

#ifndef VISUALGO_CS162_VECTOR_H
#define VISUALGO_CS162_VECTOR_H

template<class T> class Vector {
private:
    T* arr;
    int capacity{};
    int _size{};

public:
    Vector();
    explicit Vector(int capacity);
    Vector(const Vector<T>& other);
    ~Vector();

    void push_back(T data);
    void pop_back();
    void insert(int index, T data);
    void erase(int index);
    void erase(T* position);
    void clear();
    void resize(int capacity);
    void assign(int capacity, T data);
//    void reserve();
//    void shrink_to_fit();

    T& operator[](int index);
    Vector<T>& operator=(const Vector<T>& other);

    [[nodiscard]] int getCapacity() const;
    [[nodiscard]] int size() const;
    [[nodiscard]] bool empty() const;
    T& at(int index) const;
    T& front() const;
    T& back() const;
    T* data() const;
    T* begin();
    T* end();
};

template<class T>
void Vector<T>::assign(int _capacity, T data) {
    this->clear();
    this->resize(_capacity);
    for (int i = 0; i < capacity; ++i) {
        this->arr[i] = data;
    }

}

template<class T>
void Vector<T>::erase(T *position) {
    for (int i = 0; i < this->_size; ++i) {
        if (this->arr + i == position) {
            this->erase(i);
            break;
        }
    }
}

template<class T>
T *Vector<T>::end() {
    return this->arr + this->_size;
}

template<class T>
T *Vector<T>::begin() {
    return this->arr;
}

template<class T>
T *Vector<T>::data() const {
    return this->arr;
}

template<class T>
T &Vector<T>::back() const {
    return this->arr[this->_size - 1];
}

template<class T>
T &Vector<T>::front() const {
    return this->arr[0];
}

template<class T>
T &Vector<T>::at(int index) const {
    return this->arr[index];
}

template<class T>
bool Vector<T>::empty() const {
    return this->_size == 0;
}

template<class T>
int Vector<T>::size() const {
    return this->_size;
}

template<class T>
int Vector<T>::getCapacity() const {
    return this->capacity;
}

template<class T>
Vector<T> &Vector<T>::operator=(const Vector<T> &other) {
    if (this != &other) {
        this->capacity = other.capacity;
        this->_size = other._size;
        delete[] this->arr;
        this->arr = new T[this->capacity];
        for (int i = 0; i < this->_size; i++) {
            this->arr[i] = other.arr[i];
        }
    }
    return *this;
}

template<class T>
T &Vector<T>::operator[](int index) {
    return this->arr[index];
}

```

```
template<class T>
void Vector<T>::resize(int _capacity) {
    this->_size = _capacity;
    if (_capacity > 0) {
        this->capacity = _capacity;
        T* temp = new T[this->capacity];
        for (int i = 0; i < this->_size; i++) {
            temp[i] = this->arr[i];
        }
        delete[] this->arr;
        this->arr = temp;
    }
}

template<class T>
void Vector<T>::clear() {
    this->_size = 0;
}

template<class T>
void Vector<T>::erase(int index) {
    if (index >= 0 && index < this->_size) {
        for (int i = index; i < this->_size - 1; i++) {
            this->arr[i] = this->arr[i + 1];
        }
        this->_size--;
    }
}

template<class T>
void Vector<T>::insert(int index, T data) {
    if (index >= 0 && index <= this->_size) {
        if (this->_size >= this->capacity) {
            this->capacity *= 2;
            T* temp = new T[this->capacity];
            for (int i = 0; i < this->_size; i++) {
                temp[i] = this->arr[i];
            }
            delete[] this->arr;
            this->arr = temp;
        }
        for (int i = this->_size; i > index; i--) {
            this->arr[i] = this->arr[i - 1];
        }
        this->arr[index] = data;
        this->_size++;
    }
}

template<class T>
void Vector<T>::pop_back() {
    if (this->_size > 0) {
        this->_size--;
    }
}

template<class T>
void Vector<T>::push_back(T data) {
    if (this->_size >= this->capacity) {
        this->capacity *= 2;
        T* temp = new T[this->capacity];
        for (int i = 0; i < this->_size; i++) {
            temp[i] = this->arr[i];
        }
        delete[] this->arr;
        this->arr = temp;
    }
    this->arr[this->_size] = data;
    this->_size++;
}

template<class T>
Vector<T>::Vector(const Vector<T> &other) {
    this->capacity = other.capacity;
    this->_size = other._size;
    this->arr = new T[this->capacity];
    for (int i = 0; i < this->_size; i++) {
        this->arr[i] = other.arr[i];
    }
}

template<class T>
Vector<T>::~Vector() {
    delete[] this->arr;
}

template<class T>
```

```
Vector<T>::Vector(int capacity) {
    this->capacity = capacity;
    this->_size = capacity;
    this->arr = new T[this->capacity];
}

template<class T>
Vector<T>::Vector() {
    this->capacity = 10;
    this->_size = 0;
    this->arr = new T[this->capacity];
}

#endif //VISUALGO_CS162_VECTOR_H
```

8.21 include/draw/Arrow.cpp File Reference

```
#include "Arrow.hpp"
```

8.22 Arrow.cpp

Go to the documentation of this file.

```
//
// Created by dirii on 08/04/2023.
//

#include "Arrow.hpp"

Arrow::Arrow(sf::RenderWindow *window, sf::Vector2f start, sf::Vector2f end) : BaseDraw(window) {
    this->points[0] = start;
    this->points[1] = end;

    this->arrowTexture[0].loadFromFile("../assets/arrow/arrow_black.png");
    this->arrowTexture[1].loadFromFile("../assets/arrow/arrow_orange.png");

    this->arrowTexture[0].setSmooth(true);
    this->arrowTexture[1].setSmooth(true);

    this->arrowSprite.setTexture(this->arrowTexture[0]);
    sf::Vector2i topLeftCorner(
            static_cast<int>(this->arrowTexture[0].getSize().x / 2.0 - constants::Arrow::sizeArrow.x /
    2.0),
            static_cast<int>(this->arrowTexture[0].getSize().y / 2.0 - constants::Arrow::sizeArrow.y /
    2.0)
    );
    this->arrowSprite.setTextureRect(sf::IntRect(
            topLeftCorner.x,
            topLeftCorner.y,
            constants::Arrow::sizeArrow.x,
            constants::Arrow::sizeArrow.y
    ));

    this->autoScale();
    this->autoRotate();

//    this->rectangleTexture[0].loadFromFile("../assets/rectangle/rectangle_black.png");
//    this->rectangleTexture[1].loadFromFile("../assets/rectangle/rectangle_orange.png");
//    topLeftCorner = sf::Vector2i(
//            static_cast<int>(this->rectangleTexture[0].getSize().x / 2.0 -
constants::Arrow::sizeRectangle.x / 2.0),
//            static_cast<int>(this->rectangleTexture[0].getSize().y / 2.0 -
constants::Arrow::sizeRectangle.y / 2.0)
//    );
//    this->rectangleSprite.setTexture(this->rectangleTexture[0]);
//    this->rectangleSprite.setTextureRect(sf::IntRect(
//            topLeftCorner.x,
//            topLeftCorner.y,
//            constants::Arrow::sizeRectangle.x,
//            constants::Arrow::sizeRectangle.y
//    ));
//    this->rectangleSprite.setScale(
//            constants::Arrow::defaultScaleRectangle.x,
//            constants::Arrow::defaultScaleRectangle.y
//            );
//    this->rectangleSprite.setOrigin(
//            0,
//            this->rectangleSprite.getLocalBounds().height / 2.0f
//    );
//    this->rectangleSprite.setPosition(sf::Vector2f(50, 200));
//    this->rectangleSprite.setRotation(angle);

    this->hasSetMid = false;
}

void Arrow::render() {
    this->window->draw(this->arrowSprite);
//    this->window->draw(this->rectangleSprite);
}

void Arrow::toggleActiveColor() {
    this->arrowSprite.setTexture(this->arrowTexture[1]);
//    this->rectangleSprite.setTexture(this->rectangleTexture[1]);
}

void Arrow::resetColor() {
    this->arrowSprite.setTexture(this->arrowTexture[0]);
//    this->rectangleSprite.setTexture(this->rectangleTexture[0]);
}

void Arrow::setPositions(sf::Vector2f start, sf::Vector2f end, bool needSetMid) {
    this->points[0] = start;
    this->points[1] = end;
    if (needSetMid) {
        this->hasSetMid = false;
        this->setMid();
    }
    else {
        this->arrowSprite.setPosition(this->points[0]);
        this->autoScale();
        this->autoRotate();
    }
}

void Arrow::autoRotate() {
    sf::Vector2f vector2point = this->points[1] - this->points[0];
    auto angle = static_cast<float>(atan2(vector2point.y, vector2point.x) * 180 / M_PI);
    this->arrowSprite.setRotation(angle);
}

void Arrow::autoScale() {
    this->length = static_cast<float>(
            sqrt(
                    pow(this->points[1].x - this->points[0].x, 2) + pow(this->points[1].y -
this->points[0].y, 2)
            ) - constants::NodeInfo::radius - 2.f
            );
    this->arrowSprite.setScale(
            this->length / this->arrowSprite.getLocalBounds().width,
            constants::Arrow::defaultScaleArrow.y
    );
    this->arrowSprite.setOrigin(
            0,
            this->arrowSprite.getLocalBounds().height / 2.0f
    );
    this->arrowSprite.setPosition(this->points[0]);
}

void Arrow::setMid() {
    if (this->hasSetMid) return;
    this->hasSetMid = true;
    this->points[0] = sf::Vector2f(
            (this->points[0].x + this->points[1].x) / 2.0f,
            (this->points[0].y + this->points[1].y) / 2.0f
    );
    this->setStart(this->points[0], false);
}

void Arrow::setStart(sf::Vector2f start, bool needSetMid) {
    this->setPositions(start, this->points[1], needSetMid);
}

void Arrow::hide() {
    sf::Color tmp = this->arrowSprite.getColor();
    tmp.a = 0;
    this->arrowSprite.setColor(tmp);
}

void Arrow::show() {
    sf::Color tmp = this->arrowSprite.getColor();
    tmp.a = 255;
    this->arrowSprite.setColor(tmp);
```

```
00135 }
```

8.23 include/draw/Arrow.hpp File Reference

```
#include <cmath>
#include <SFML/Graphics.hpp>
#include "BaseDraw.hpp"
#include "Constants.hpp"
```

Classes

- class Arrow

8.24 Arrow.hpp

Go to the documentation of this file.

```
//
// Created by dirii on 08/04/2023.
//

#ifndef VISUALGO_CS162_ARROW_HPP
#define VISUALGO_CS162_ARROW_HPP

#include <cmath>
#include <SFML/Graphics.hpp>
#include "BaseDraw.hpp"
#include "Constants.hpp"

class Arrow : public BaseDraw{
protected:
    sf::Vector2f points[2];
    sf::Texture arrowTexture[2];
    sf::Sprite arrowSprite;
    float length;
    bool hasSetMid;

public:
    Arrow(sf::RenderWindow* window, sf::Vector2f start, sf::Vector2f end);
    void render() override;
    void toggleActiveColor();
    void resetColor();
    void setStart(sf::Vector2f start, bool needSetMid);
    void setPositions(sf::Vector2f start, sf::Vector2f end, bool needSetMid);
    void setMid();
    void autoRotate();
    void autoScale();

    void hide();
    void show();
};

#endif //VISUALGO_CS162_ARROW_HPP
```

8.25 include/draw/BackArrow.cpp File Reference

```
#include "BackArrow.hpp"
```

8.26 BackArrow.cpp

Go to the documentation of this file.

```
//
// Created by dirii on 26/04/2023.
//

#include "BackArrow.hpp"

BackArrow::BackArrow(sf::RenderWindow *window, sf::Vector2f start, sf::Vector2f end) :
BaseDraw(window) {
    this->isShow = false;

    this->points[0] = end;
    this->points[1] = start;
    this->points[2] = sf::Vector2f(
            this->points[0].x,
            this->points[0].y - constants::NodeInfo::offsetX
    );
    this->points[3] = sf::Vector2f(
            this->points[1].x,
            this->points[2].y
    );
    this->arrow = new Arrow(window, this->points[2], this->points[0]);

    this->rectangleTexture[0].loadFromFile("../assets/rectangle/rectangle_black.png");
    this->rectangleTexture[1].loadFromFile("../assets/rectangle/rectangle_orange.png");

    this->rectangleTexture[0].setRepeated(true);
    this->rectangleTexture[1].setRepeated(true);

    sf::Vector2i topLeftCorner = sf::Vector2i(
            static_cast<int>(this->rectangleTexture[0].getSize().x / 2.0 -
constants::Arrow::sizeRectangle.x / 2.0),
            static_cast<int>(this->rectangleTexture[0].getSize().y / 2.0 -
constants::Arrow::sizeRectangle.y / 2.0)
    );
    for (auto & rectangleSprite : this->rectangleSprites) {
        rectangleSprite.setTexture(this->rectangleTexture[0]);
        rectangleSprite.setTextureRect(sf::IntRect(
                topLeftCorner.x,
                topLeftCorner.y,
                constants::Arrow::sizeRectangle.x,
                constants::Arrow::sizeRectangle.y
        ));
    }

    this->setPosition(start, end);
}

void BackArrow::render() {
    if (this->isShow) {
        this->window->draw(this->rectangleSprites[0]);
        this->window->draw(this->rectangleSprites[1]);
        this->arrow->render();
    }
}

void BackArrow::show() {
    this->isShow = true;
}

void BackArrow::hide() {
    this->isShow = false;
}

void BackArrow::toggleActiveColorNode() {
    this->rectangleSprites[0].setTexture(this->rectangleTexture[1]);
    this->rectangleSprites[1].setTexture(this->rectangleTexture[1]);
    this->arrow->toggleActiveColor();
}

void BackArrow::resetColor() {
    this->rectangleSprites[0].setTexture(this->rectangleTexture[0]);
    this->rectangleSprites[1].setTexture(this->rectangleTexture[0]);
    this->arrow->resetColor();
}

void BackArrow::setPosition(sf::Vector2f start, sf::Vector2f end) {
    this->points[0] = end;
    this->points[1] = start;
    if (end == start) {
        this->hide();
        return;
    }
    this->points[2] = sf::Vector2f(
            this->points[0].x,
            this->points[0].y - constants::NodeInfo::offsetX
    );
    this->points[3] = sf::Vector2f(
            this->points[1].x,
            this->points[2].y
    );
    this->arrow->setPositions(this->points[2], this->points[0], false);
    this->autoRotate();
    this->autoScale();
}

void BackArrow::autoScale() {
    float length = sqrtf(
            powf(this->points[3].x - this->points[2].x, 2) + powf(this->points[3].y -
this->points[2].y, 2)
            );
    this->rectangleSprites[0].setScale(
            length / this->rectangleSprites[0].getLocalBounds().width,
            constants::Arrow::defaultScaleRectangle.y
    );
    length = sqrtf(
            powf(this->points[3].x - this->points[1].x, 2) + powf(this->points[3].y -
this->points[1].y, 2)
            );
    this->rectangleSprites[1].setScale(
            length / this->rectangleSprites[1].getLocalBounds().width,
            constants::Arrow::defaultScaleRectangle.y
    );
    this->rectangleSprites[0].setOrigin(
            this->rectangleSprites[0].getLocalBounds().width / 2.0f,
            0
    );
    this->rectangleSprites[1].setOrigin(
            this->rectangleSprites[1].getLocalBounds().width,
            this->rectangleSprites[1].getLocalBounds().height / 2.0f
    );
    this->rectangleSprites[0].setPosition(
            (this->points[3].x + this->points[2].x) / 2.0f,
            (this->points[3].y + this->points[2].y) / 2.0f
            );
    this->rectangleSprites[1].setPosition(this->points[1]);
}

void BackArrow::autoRotate() {
    sf::Vector2f vector2point = this->points[3] - this->points[2];
    float angle = atan2f(vector2point.y, vector2point.x) * 180.0f / (float)M_PI;
    this->rectangleSprites[0].setRotation(angle);
    vector2point = this->points[1] - this->points[3];
    angle = atan2f(vector2point.y, vector2point.x) * 180.0f / (float)M_PI;
    this->rectangleSprites[1].setRotation(angle);
}
```

8.27 include/draw/BackArrow.hpp File Reference

```
#include "Arrow.hpp"
#include "Constants.hpp"
```

Classes

- class BackArrow

8.28 BackArrow.hpp

Go to the documentation of this file.

```
//
// Created by dirii on 26/04/2023.
//
```

```

#ifndef VISUALGO_CS162_BACKARROW_HPP
#define VISUALGO_CS162_BACKARROW_HPP

#include "Arrow.hpp"
#include "Constants.hpp"

class BackArrow : public BaseDraw {
private:
    sf::Vector2f points[4];
    sf::Texture rectangleTexture[2];
    sf::Sprite rectangleSprites[2];
    Arrow* arrow;
    bool isShow;

public:
    BackArrow(sf::RenderWindow* window, sf::Vector2f start, sf::Vector2f end);
    void render() override;

    void autoScale();
    void autoRotate();

    void toggleActiveColorNode();
    void resetColor();

    void setPosition(sf::Vector2f start, sf::Vector2f end);

    void show();
    void hide();
};

#endif //VISUALGO_CS162_BACKARROW_HPP
```

8.29 include/draw/BaseDraw.cpp File Reference

```
#include "BaseDraw.hpp"
```

8.30 BaseDraw.cpp

Go to the documentation of this file.

```
//
// Created by dirii on 08/04/2023.
//

#include "BaseDraw.hpp"

BaseDraw::BaseDraw(sf::RenderWindow *window) {
    this->window = window;
}
```

8.31 include/draw/BaseDraw.hpp File Reference

```
#include <SFML/Graphics.hpp>
```

Classes

- class BaseDraw

8.32 BaseDraw.hpp

Go to the documentation of this file.

```
//
// Created by dirii on 08/04/2023.
//

#ifndef VISUALGO_CS162_BASEDRAW_HPP
#define VISUALGO_CS162_BASEDRAW_HPP

#include <SFML/Graphics.hpp>

class BaseDraw{
protected:
    sf::RenderWindow* window;

public:
    explicit BaseDraw(sf::RenderWindow* window);


    virtual void render() = 0;
};

#endif //VISUALGO_CS162_BASEDRAW_HPP
```

8.33 include/draw/NodeInfo.cpp File Reference

```
#include "NodeInfo.hpp"
```

8.34 NodeInfo.cpp

Go to the documentation of this file.

```
//
// Created by dirii on 08/04/2023.
//

#include "NodeInfo.hpp"

NodeInfo::NodeInfo(sf::RenderWindow *window, std::string value, sf::Vector2f position, bool _isDLL) :
      BaseDraw(window) {
    this->values[0] = value;
    this->values[1] = value;

    this->positions[(int)TypeNode::Normal] = position;
    this->positions[(int)TypeNode::Effective] = position;
    this->positions[(int)TypeNode::Outside] = sf::Vector2f(
            position.x,
            position.y + constants::NodeInfo::offsetY
            );

    this->isDLL = _isDLL;

    this->statusNode = StatusNode::InChain;

    this->node = new SingleNode(window, std::move(value), this->positions[(int)TypeNode::Normal]);

    for (auto &arrow : this->arrows)
        arrow[(int)ArrowType::LEFT] = arrow[(int)ArrowType::RIGHT] = nullptr;

    this->isPrintPreVal = this->isPrintNormal = false;

    this->title.setFont(this->node->font);
    this->title.setCharacterSize(constants::TitleNode::fontSize);
    this->title.setFillColor(constants::titleGreen);
    this->title.setString("");
}

void NodeInfo::render() {
    if (this->statusNode == StatusNode::Visible)
        return;

    if (this->isDLL && this->statusNode == StatusNode::InChain){
        if (this->arrows[1][(int)ArrowType::LEFT])
            this->arrows[1][(int)ArrowType::LEFT]->render();
        if (this->arrows[1][(int)ArrowType::RIGHT])
            this->arrows[1][(int)ArrowType::RIGHT]->render();
    } else {
        if (this->arrows[0][(int)ArrowType::LEFT])
            this->arrows[0][(int)ArrowType::LEFT]->render();
        if (this->arrows[0][(int)ArrowType::RIGHT])
            this->arrows[0][(int)ArrowType::RIGHT]->render();
    }
    this->node->render();
    this->window->draw(this->title);
}

void NodeInfo::initArrow(NodeInfo::ArrowType type, sf::Vector2f start, sf::Vector2f end) {
    this->arrows[1][(int)type] = new Arrow(this->window, start, end);
    this->arrows[1][(int)type]->setMid();
    this->arrows[0][(int)type] = new Arrow(this->window, start, end);
}

void NodeInfo::toggleActiveColorNode() {
    this->node->toggleActiveColor();
}

void NodeInfo::toggleActiveColorArrow(NodeInfo::ArrowType type) {
    if (this->arrows[0][(int)type])
        this->arrows[0][(int)type]->toggleActiveColor();
    if (this->arrows[1][(int)type])
        this->arrows[1][(int)type]->toggleActiveColor();
}

void NodeInfo::resetColorNode() {
    this->node->resetColor();
}

void NodeInfo::resetColorArrow(NodeInfo::ArrowType type) {
    if (this->arrows[0][(int)type])
        this->arrows[0][(int)type]->resetColor();
    if (this->arrows[1][(int)type])
        this->arrows[1][(int)type]->resetColor();
}

void NodeInfo::reset() {
    this->resetColorNode();
    this->resetColorArrow(ArrowType::LEFT);
    this->resetColorArrow(ArrowType::RIGHT);
    this->resetTitle();
    this->isPrintNormal = this->isPrintPreVal = false;
    this->statusNode = StatusNode::InChain;
    this->show(ArrowType::LEFT);
    this->show(ArrowType::RIGHT);
}

// require update() before calling this function
sf::Vector2f NodeInfo::getPosition() {
    this->updateNode(); // ?
    return this->node->getPosition();
}

void NodeInfo::reInitPos(int index) {
    this->positions[(int)TypeNode::Normal] = sf::Vector2f(
            constants::NodeInfo::originNode.x + static_cast<float>(index) *
constants::NodeInfo::offsetX,
            constants::NodeInfo::originNode.y
    );
    this->positions[(int)TypeNode::Outside] = sf::Vector2f(
            this->positions[(int)TypeNode::Effective].x,
            this->positions[(int)TypeNode::Effective].y + constants::NodeInfo::offsetY
            );
}

void NodeInfo::setPrintPreVal() {
    this->isPrintPreVal = true;
}

void NodeInfo::setPrintNormal() {
    this->isPrintNormal = true;
}

void NodeInfo::setNodeOutside() {
    this->statusNode = StatusNode::OutChain;
}

void NodeInfo::setNodeInChain() {
    this->statusNode = StatusNode::InChain;
}
```

```

void NodeInfo::setNodeVisible() {
    this->statusNode = StatusNode::Visible;
}

// require calculate effective positions of a chain before calling this function
void NodeInfo::updateNode() {
    if (this->statusNode == StatusNode::Visible)
        return;

    if (this->statusNode == StatusNode::InChain) {
        if (this->isPrintNormal) {
            this->node->setPosition(this->positions[(int)TypeNode::Normal]);
        } else {
            this->node->setPosition(this->positions[(int)TypeNode::Effective]);
        }
    } else {
        this->node->setPosition(this->positions[(int)TypeNode::Outside]);
    }

    if (this->isPrintPreVal) {
        this->node->setText(this->values[1]);
    } else {
        this->node->setText(this->values[0]);
    }
}

void NodeInfo::updateArrows(ArrowType type, sf::Vector2f end){
    if (this->arrows[0][(int)type])
        this->arrows[0][(int)type]->setPositions(this->node->getPosition(), end, false);

    if (this->arrows[1][(int)type])
        this->arrows[1][(int)type]->setPositions(this->node->getPosition(), end, true);
}

void NodeInfo::reInitPreVal() {
    this->values[1] = this->values[0];
}

NodeInfo::StatusNode NodeInfo::getStatusNode() {
    return this->statusNode;
}

void NodeInfo::setEffectivePosition(sf::Vector2f start) {
    this->positions[(int)TypeNode::Effective] = start;
}

void NodeInfo::setArrows(NodeInfo::ArrowType type, sf::Vector2f start, sf::Vector2f end) {
    if (this->arrows[0][(int)type])
        this->arrows[0][(int)type]->setPositions(start, end, false);
    if (this->arrows[1][(int)type])
        this->arrows[1][(int)type]->setPositions(start, end, true);
}

void NodeInfo::hide(NodeInfo::ArrowType type) {
    if (this->arrows[0][(int)type])
        this->arrows[0][(int)type]->hide();
    if (this->arrows[1][(int)type])
        this->arrows[1][(int)type]->hide();
}

void NodeInfo::show(NodeInfo::ArrowType type) {
    if (this->arrows[0][(int)type])
        this->arrows[0][(int)type]->show();
    if (this->arrows[1][(int)type])
        this->arrows[1][(int)type]->show();
}

NodeInfo::~NodeInfo() {
    delete this->node;
    for (auto & arrow : this->arrows) {
        for (auto & j : arrow) {
            delete j;
        }
    }
}

void NodeInfo::setValue(std::string value) {
    this->values[0] = std::move(value);
}

std::string NodeInfo::getValue() {
    return this->values[0];
}

void NodeInfo::setTitle(const std::string& _title) {
    std::string preTitle = this->title.getString();
    if (!preTitle.empty())
        preTitle += "|";
    preTitle += _title;
    this->title.setString(preTitle);
    sf::Vector2f pos = this->node->getPosition();
    this->title.setOrigin(
            this->title.getGlobalBounds().width / 2,
            this->title.getGlobalBounds().height / 2
            );
    this->title.setPosition(
            pos.x,
            pos.y + constants::TitleNode::offsetY
            );
}

void NodeInfo::resetTitle() {
    this->title.setString("");
}

void NodeInfo::destroyArrow(NodeInfo::ArrowType type) {
    if (this->arrows[0][(int)type])
        delete this->arrows[0][(int)type];
    if (this->arrows[1][(int)type])
        delete this->arrows[1][(int)type];
    this->arrows[0][(int)type] = nullptr;
    this->arrows[1][(int)type] = nullptr;
}
```

8.35 include/draw/NodeInfo.hpp File Reference

```
#include "BaseDraw.hpp"
#include "SingleNode.hpp"
#include "Arrow.hpp"
```

Classes

- class NodeInfo

8.36 NodeInfo.hpp

Go to the documentation of this file.

```
//
// Created by dirii on 08/04/2023.
//

#ifndef VISUALGO_CS162_NODEINFO_HPP
#define VISUALGO_CS162_NODEINFO_HPP

#include "BaseDraw.hpp"
#include "SingleNode.hpp"
#include "Arrow.hpp"

class NodeInfo : public BaseDraw {
public:
    enum class ArrowType {
        LEFT,
        RIGHT
    };

    enum class StatusNode{
        InChain,
        OutChain,
        Visible
    };

    enum class TypeNode{
        Normal,
        Outside,
        Effective
    };

    NodeInfo(sf::RenderWindow* window, std::string value, sf::Vector2f position, bool _isDLL);
    ~NodeInfo();
    void updateNode();
    void updateArrows(ArrowType type, sf::Vector2f end);
    void render() override;

    void initArrow(ArrowType type, sf::Vector2f start, sf::Vector2f end);
    void destroyArrow(ArrowType type);

    void reInitPos(int index);
    void reInitPreVal();

    void setEffectivePosition(sf::Vector2f start);
    void setArrows(ArrowType type, sf::Vector2f start, sf::Vector2f end);
    void setValue(std::string value);

    sf::Vector2f getPosition();
    std::string getValue();

    void toggleActiveColorNode();
    void toggleActiveColorArrow(ArrowType type);

    void setPrintPreVal();
    void setPrintNormal();

    void setNodeInChain();
    void setNodeOutside();
    void setNodeVisible();

    void setTitle(const std::string& title);

    void hide(ArrowType type);
    void show(ArrowType type);

    StatusNode getStatusNode();

    void resetColorNode();
    void resetColorArrow(ArrowType type);
    void resetTitle();

    void reset();

private:
    sf::Vector2f positions[3];
    SingleNode* node;
    Arrow* arrows[2][2];
    std::string values[2];
    sf::Text title;

    StatusNode statusNode;
    bool isPrintPreVal, isDLL, isPrintNormal;
};

#endif //VISUALGO_CS162_NODEINFO_HPP
```

8.37 include/draw/SingleNode.cpp File Reference

```
#include "SingleNode.hpp"
```

8.38 SingleNode.cpp

Go to the documentation of this file.

```
//
// Created by dirii on 10/04/2023.
//

#include "SingleNode.hpp"

SingleNode::SingleNode(sf::RenderWindow *window, std::string value, sf::Vector2f position) :
BaseDraw(window) {
    this->value = std::move(value);

    this->circle.setRadius(constants::NodeInfo::radius);
    this->circle.setFillColor(sf::Color::White);
    this->circle.setOutlineThickness(constants::NodeInfo::outlineThickness);
    this->circle.setOutlineColor(sf::Color::Black);
    this->circle.setPointCount(constants::NodeInfo::pointCount);
    sf::FloatRect bounds = this->circle.getLocalBounds();
    this->circle.setOrigin(bounds.left + bounds.width / 2.0f,bounds.top + bounds.height / 2.0f);
    this->circle.setPosition(position);

    this->font.loadFromFile(constants::fontPath);
    this->label.setFont(this->font);
    this->label.setString(this->value);
    this->label.setCharacterSize(constants::NodeInfo::fontSize);
    this->label.setFillColor(sf::Color::Black);
    bounds = this->label.getLocalBounds();
    this->label.setOrigin(bounds.left + bounds.width / 2.0f, bounds.top + bounds.height / 2.0f);
    this->label.setPosition(position);
}

void SingleNode::render() {
    this->window->draw(this->circle);
    this->window->draw(this->label);
}

void SingleNode::toggleActiveColor() {
    this->circle.setOutlineColor(constants::normalGreen);
}

void SingleNode::resetColor() {
    this->circle.setOutlineColor(sf::Color::Black);
}

void SingleNode::setPosition(sf::Vector2f position) {
    this->circle.setPosition(position);
    this->label.setPosition(position);
}

sf::Vector2f SingleNode::getPosition() {
    return this->circle.getPosition();
}

void SingleNode::setText(std::string _value) {
    this->value = std::move(_value);
    this->label.setString(this->value);
    sf::FloatRect bounds = this->label.getLocalBounds();
    this->label.setOrigin(bounds.left + bounds.width / 2.0f, bounds.top + bounds.height / 2.0f);
    this->label.setPosition(this->circle.getPosition());
}
```

8.39 include/draw/SingleNode.hpp File Reference

```
#include "Constants.hpp"
#include "BaseDraw.hpp"
#include <iostream>
```

Classes

- class SingleNode

8.40 SingleNode.hpp

Go to the documentation of this file.

```
//
// Created by dirii on 10/04/2023.
//

#ifndef VISUALGO_CS162_SINGLENODE_HPP
#define VISUALGO_CS162_SINGLENODE_HPP

```

```
#include "Constants.hpp"
#include "BaseDraw.hpp"
#include <iostream>

class SingleNode : public BaseDraw{
private:
    sf::CircleShape circle;
    sf::Text label;
    std::string value;

public:
    sf::Font font;

    SingleNode(sf::RenderWindow* window, std::string value, sf::Vector2f position);
    void render() override;
    void toggleActiveColor();
    void resetColor();
    void setText(std::string _value);
    void setPosition(sf::Vector2f position);
    sf::Vector2f getPosition();
};

#endif //VISUALGO_CS162_SINGLENODE_HPP
```

8.41 include/draw/Square.cpp File Reference

```
#include "Square.hpp"
```

8.42 Square.cpp

Go to the documentation of this file.

```
//
// Created by dirii on 28/04/2023.
//

#include "Square.hpp"

Square::Square(sf::RenderWindow *window, std::string value, sf::Vector2f position)
        : BaseDraw(window) {
    this->value = std::move(value);

    this->square.setSize(sf::Vector2f(constants::Square::length, constants::Square::length));
    this->square.setFillColor(sf::Color::White);
    this->square.setOutlineThickness(constants::Square::outlineThickness);
    this->square.setOutlineColor(sf::Color::Black);
    sf::FloatRect bounds = this->square.getLocalBounds();
    this->square.setOrigin(bounds.left + bounds.width / 2.0f,bounds.top + bounds.height / 2.0f);
    this->square.setPosition(position);

    this->font.loadFromFile(constants::fontPath);
    this->label.setFont(this->font);
    this->label.setString(this->value);
    this->label.setCharacterSize(constants::Square::fontSize);
    this->label.setFillColor(sf::Color::Black);
    bounds = this->label.getLocalBounds();
    this->label.setOrigin(bounds.left + bounds.width / 2.0f, bounds.top + bounds.height / 2.0f);
    this->label.setPosition(position);
}

void Square::render() {
    switch (this->status) {
        case Status::active:
            this->square.setOutlineColor(constants::normalGreen);
            break;
        case Status::inactive:
            this->square.setOutlineColor(sf::Color::Black);
            break;
        case Status::chosen:
            this->square.setOutlineColor(constants::clickGreen);
            break;
    }
    this->window->draw(this->square);
    this->window->draw(this->label);
}
```

```

void Square::resetColor() {
    this->status = Status::inactive;
}

void Square::setText(std::string _value) {
    this->value = std::move(_value);
    this->label.setString(this->value);
    sf::FloatRect bounds = this->label.getLocalBounds();
    this->label.setOrigin(bounds.left + bounds.width / 2.0f, bounds.top + bounds.height / 2.0f);
    this->label.setPosition(this->square.getPosition());
}

void Square::setPosition(sf::Vector2f position) {
    this->square.setPosition(position);
    this->label.setPosition(position);
}

sf::Vector2f Square::getPosition() {
    return this->square.getPosition();
}

void Square::setStatus(Square::Status _status) {
    this->status = _status;
}

Square::Status Square::getStatus() {
    return this->status;
}
```

8.43 include/draw/Square.hpp File Reference

```
#include "Constants.hpp"
#include "BaseDraw.hpp"
```

Classes

- class Square

8.44 Square.hpp

Go to the documentation of this file.

```
//
// Created by dirii on 28/04/2023.
//

#ifndef VISUALGO_CS162_SQUARE_HPP
#define VISUALGO_CS162_SQUARE_HPP

#include "Constants.hpp"
#include "BaseDraw.hpp"

class Square : BaseDraw{
public:
    enum class Status{
        inactive,
        active,
        chosen,
        hidden
    };

    sf::Font font;

    Square(sf::RenderWindow *window, std::string value, sf::Vector2f position);
    void render() override;

    void setStatus(Status _status);
    void resetColor();
    Status getStatus();
    void setText(std::string _value);
    void setPosition(sf::Vector2f position);
    sf::Vector2f getPosition();

private:
    sf::RectangleShape square;
    sf::Text label;
    std::string value;
    Status status = Status::inactive;
};

#endif //VISUALGO_CS162_SQUARE_HPP
```

8.45 include/draw/SquareInfo.cpp File Reference

```
#include "SquareInfo.hpp"
```

8.46 SquareInfo.cpp

Go to the documentation of this file.

```
//
// Created by dirii on 28/04/2023.
//

#include "SquareInfo.hpp"

SquareInfo::SquareInfo(sf::RenderWindow *window, std::string value, sf::Vector2f position) :
BaseDraw(window) {
    this->position = position;
    this->square = new Square(window, value, position);
    this->values[0] = std::move(value);
    this->values[1] = "";
    this->isPrintPreVal = false;

    this->title.setFont(this->square->font);
    this->title.setCharacterSize(20);
    this->title.setFillColor(sf::Color::Black);
}

void SquareInfo::render() {
    if (this->square->getStatus() != Square::Status::hidden) {
        this->square->render();
        this->window->draw(this->title);
    }
}

void SquareInfo::setValue(std::string value) {
    this->values[1] = this->values[0];
    this->values[0] = std::move(value);
}

void SquareInfo::update() {
    if (this->isPrintPreVal)
        this->square->setText(this->values[1]);
    else
        this->square->setText(this->values[0]);
}

void SquareInfo::setTitle(const std::string& _title) {
    this->title.setString(_title);
    sf::FloatRect bounds = this->title.getLocalBounds();
    this->title.setOrigin(bounds.left + bounds.width / 2.0f, bounds.top + bounds.height / 2.0f);
    this->title.setPosition(this->position.x, this->position.y + constants::TitleNode::offsetY);
}

void SquareInfo::resetTitle() {
    this->title.setString("");
}

void SquareInfo::reset() {
    this->resetTitle();
    this->square->resetColor();
    this->isPrintPreVal = false;
}

void SquareInfo::setStatus(Square::Status _status) {
    this->square->setStatus(_status);
}

std::string SquareInfo::getValue() {
    return this->values[0];
}

void SquareInfo::setPrintPreVal(bool _isPrintPreVal) {
    this->isPrintPreVal = _isPrintPreVal;
}

Square::Status SquareInfo::getStatus() {
    return this->square->getStatus();
}
```

8.47 include/draw/SquareInfo.hpp File Reference

```
#include "Square.hpp"
```

Classes

- class SquareInfo

8.48 SquareInfo.hpp

Go to the documentation of this file.

```
//
// Created by dirii on 28/04/2023.
//

#ifndef VISUALGO_CS162_SQUAREINFO_HPP
#define VISUALGO_CS162_SQUAREINFO_HPP

#include "Square.hpp"

class SquareInfo : public BaseDraw {
public:
    SquareInfo(sf::RenderWindow *window, std::string value, sf::Vector2f position);
    ~SquareInfo() = default;
    void update();
    void render() override;

    void setValue(std::string value);
    void setTitle(const std::string& _title);
    void setStatus(Square::Status _status);
    void setPrintPreVal(bool _isPrintPreVal);

    std::string getValue();
    Square::Status getStatus();

    void resetTitle();
    void reset();

private:
    sf::Vector2f position;
    Square* square;
    std::string values[2];
    sf::Text title;

    bool isPrintPreVal;
};

#endif //VISUALGO_CS162_SQUAREINFO_HPP
```

8.49 include/libScene/AllScenes.hpp File Reference

```
#include "MainMenu.hpp"
#include "SLLScene.hpp"
#include "DLLScene.hpp"
#include "CLLScene.hpp"
#include "StackScene.hpp"
#include "QueueScene.hpp"
#include "StaticArrayScene.hpp"
#include "DynamicArrayScene.hpp"
```

8.50 AllScenes.hpp

Go to the documentation of this file.

```
//
// Created by dirii on 29/03/2023.
//

#ifndef VISUALGO_CS162_ALLSCENES_HPP
#define VISUALGO_CS162_ALLSCENES_HPP

#include "MainMenu.hpp"
#include "SLLScene.hpp"
#include "DLLScene.hpp"
#include "CLLScene.hpp"
#include "StackScene.hpp"
#include "QueueScene.hpp"
#include "StaticArrayScene.hpp"
#include "DynamicArrayScene.hpp"

#endif //VISUALGO_CS162_ALLSCENES_HPP
```

8.51 include/libScene/BaseScene.cpp File Reference

```
#include "BaseScene.hpp"
```

8.52 BaseScene.cpp

Go to the documentation of this file.

```
//
// Created by dirii on 25/03/2023.
//

#include "BaseScene.hpp"

void BaseScene::setWindow(sf::RenderWindow *window) {
    this->window = window;
}

void BaseScene::createModeButton(sf::Vector2f position, std::string textString) {
    this->modeButton = new Button(
            this->window,
            position,
            constants::modeButtonSize,
            textString,
            textString,
            constants::sizeTextModeButton,
            sf::Color::Black,
            constants::normalGray,
            constants::hoverGray,
            constants::clickGray
            );
}

BaseScene::BaseScene(sf::RenderWindow *window) {
    this->setWindow(window);
    this->isMenuOpen = false;
    this->isDemoCodeOpen = false;

    this->controlMenu = new ControlMenu(this->window);
}
```

8.53 include/libScene/BaseScene.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include "stuff/button.hpp"
#include "ControlMenu.hpp"
```

Classes

- class BaseScene

8.54 BaseScene.hpp

Go to the documentation of this file.

```
//
// Created by dirii on 23/03/2023.
//

#ifndef VISUALGO_CS162_BASESCENE_HPP
#define VISUALGO_CS162_BASESCENE_HPP

#include <SFML/Graphics.hpp>
#include "stuff/button.hpp"
#include "ControlMenu.hpp"

class BaseScene{
protected:
    sf::RenderWindow* window{};
    ControlMenu* controlMenu;

    void setWindow(sf::RenderWindow* window);
public:
    Button* modeButton{};
    bool isMenuOpen{}, isDemoCodeOpen{};

    explicit BaseScene(sf::RenderWindow* window);

    void createModeButton(sf::Vector2f position, std::string textString);

    virtual void pollEvent(sf::Event event, sf::Vector2f mousePosView) = 0;
    virtual void update() = 0;
    virtual void render() = 0;
};

#endif //VISUALGO_CS162_BASESCENE_HPP
```

8.55 include/libScene/CLLScene.cpp File Reference

```
#include "CLLScene.hpp"
```

8.56 CLLScene.cpp

Go to the documentation of this file.

```
//
// Created by dirii on 28/03/2023.
//

#include "CLLScene.hpp"

CLLScene::CLLScene(sf::RenderWindow *window) : BaseScene(window) {
    this->init();
}

void CLLScene::pollEvent(sf::Event event, sf::Vector2f mousePosView) {
    if (this->isMenuOpen)
        this->menu->pollEvents(event, mousePosView);

    this->controlMenu->pollEvents(event, mousePosView);
}

void CLLScene::update() {
    if (this->isMenuOpen) {
        this->menu->update();

        constants::MenuLinkedList::Button status = this->menu->getActiveOptionMenu();
        constants::MenuLinkedList::CreateMode::Button createMode;
        switch (status){
            case constants::MenuLinkedList::Button::CREATE_BUTTON:
                createMode = this->menu->getActiveCreateMode();
                if (createMode == constants::MenuLinkedList::CreateMode::Button::RANDOM_BUTTON) {
                    if (this->menu->createModeValue[0] == "None")
                        break;
                    if (this->menu->createModeValue[0].empty())
                        this->menu->createModeValue[0] = "0";
                    int size = std::stoi(this->menu->createModeValue[0]);
                    this->linkedList->createLinkedList(size);
                } else if (createMode == constants::MenuLinkedList::CreateMode::Button::DEFINED_LIST_BUTTON) {
                    if (this->menu->createModeValue[1] == "None")
                        break;
                    std::vector<std::string> values;
                    std::string value = this->menu->createModeValue[1];
                    std::stringstream ss(value);
                    std::string token;
                    while (std::getline(ss, token, ',')) {
                        values.push_back(token);
                    }
                    this->linkedList->createLinkedList(values);
                } else if (createMode == constants::MenuLinkedList::CreateMode::Button::FILE_BUTTON) {
                    if (this->menu->createModeValue[2] == "None")
                        break;
                    std::vector<std::string> values;
                    std::string value = this->menu->createModeValue[2];
                    std::stringstream ss(value);
                    std::string token;
                    while (std::getline(ss, token, ','))
                        values.push_back(token);
                    this->linkedList->createLinkedList(values);
                    this->menu->createModeValue[2] = "None";
                }
                this->controlMenu->reset();
                break;
            case constants::MenuLinkedList::Button::ADD_BUTTON:
                if (this->menu->addModeValue[0] == "None" || this->menu->addModeValue[1] == "None" || this->menu->addModeValue[0].empty())
                    break;

                this->linkedList->addNode(
                        std::stoi(this->menu->addModeValue[0]),
                        this->menu->addModeValue[1],
                        this->addModeEvents(std::stoi(this->menu->addModeValue[0]))
                );

                std::cout « "Add: " « this->menu->addModeValue[0] « " " « this->menu->addModeValue[1] « std::endl;
                this->menu->addModeValue[0] = this->menu->addModeValue[1] = "None";
                this->controlMenu->reset();
                break;
            case constants::MenuLinkedList::Button::DELETE_BUTTON:
                if (this->menu->deleteModeValue == "None" || this->menu->deleteModeValue.empty())
                    break;

                this->linkedList->deleteNode(
                        std::stoi(this->menu->deleteModeValue),
                        this->deleteModeEvents(std::stoi(this->menu->deleteModeValue))
                );

                std::cout « "Delete: " « this->menu->deleteModeValue « std::endl;
                this->menu->deleteModeValue = "None";
                this->controlMenu->reset();
                break;
            case constants::MenuLinkedList::Button::UPDATE_BUTTON:
                if (this->menu->updateModeValue[0] == "None" || this->menu->updateModeValue[1] ==
"None" || this->menu->updateModeValue[0].empty())
                    break;

                this->linkedList->updateNode(
                        std::stoi(this->menu->updateModeValue[0]),
                        this->menu->updateModeValue[1],
                        this->updateModeEvents(std::stoi(this->menu->updateModeValue[0]))
                );

                std::cout « "Update: " « this->menu->updateModeValue[0] « " " «
this->menu->updateModeValue[1] « std::endl;
                this->menu->updateModeValue[0] = this->menu->updateModeValue[1] = "None";
                this->controlMenu->reset();
                break;
            case constants::MenuLinkedList::Button::SEARCH_BUTTON:
                if (this->menu->searchModeValue == "None" || this->menu->searchModeValue.empty())
                    break;

                this->linkedList->searchNode(

this->searchModeEvents(this->linkedList->findValue(this->menu->searchModeValue))
                );

                std::cout « "Search: " « this->menu->searchModeValue « std::endl;
                this->menu->searchModeValue = "None";
                this->controlMenu->reset();
                break;
        }
    }

    this->controlMenu->update();

    this->linkedList->processControlMenu(this->controlMenu->getStatus());
    this->linkedList->setSpeed(this->controlMenu->getSpeed());

    this->linkedList->update();
}

void CLLScene::render() {
    if (this->isMenuOpen)
        this->menu->render();

    if (this->isDemoCodeOpen)
        this->linkedList->renderHighlighter();

    this->controlMenu->render();
    this->linkedList->render();
}

void CLLScene::init() {
    this->menu = new MenuLinkedList(this->window);
    this->linkedList = new LinkedList(this->window, LinkedList::TypeLinkedList::CIRCULAR);
}

void CLLScene::reset() {
    this->menu->resetActiveOptionMenu();
}

std::vector<EventAnimation> CLLScene::addModeEvents(int chosenNode) {
    this->linkedList->resetEvents();
    if (chosenNode < 0 || chosenNode > this->linkedList->getSize())
        return {};

    this->linkedList->initHighlighter(
            constants::Highlighter::SLL::CODES_PATH[0].second,
            constants::Highlighter::SLL::CODES_PATH[0].first
    );

    std::vector<EventAnimation> events;
    EventAnimation event;

    if (chosenNode) {
        event.titleNodes = {
                {0,          "head"},
                {chosenNode, "temp"}
        };
        event.indexBackArrow.second = 0;
    }
    else {
        event.titleNodes.emplace_back(chosenNode, "temp");
        if (this->linkedList->getSize()) {
            event.titleNodes.emplace_back(1, "head");
            event.indexBackArrow.second = 1;
        }
    }
    event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
    if (chosenNode && chosenNode == this->linkedList->getSize())
        event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
    event.colorNodes.push_back(chosenNode);
    event.statusChosenNode = NodeInfo::StatusNode::OutChain;
    event.indexBackArrow.first = this->linkedList->getSize();
    event.lines = {0};

    events.emplace_back(event);

    if (chosenNode == 0) {
        if (this->linkedList->getSize()) {
            event.reset();
            event.titleNodes = {
                    {1, "head"},
                    {chosenNode, "temp"}
            };
            event.colorNodes = std::vector<int>{0};
            event.colorArrows.emplace_back(0, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.isPrintNormal = true;
            event.indexBackArrow = {this->linkedList->getSize(), 1};
            event.lines = {1, 2};

            events.emplace_back(event);
        }

        event.reset();
        event.titleNodes.emplace_back(chosenNode, "head|temp");
        event.lines = {3};
        event.statusChosenNode = NodeInfo::StatusNode::InChain;
        event.indexBackArrow = {this->linkedList->getSize(), 0};
        events.emplace_back(event);
    } else {
        event.reset();
        event.titleNodes = {
                {0, "head|current"},
                {chosenNode, "temp"}
        };
        event.colorNodes.push_back(0);
        event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
        if (chosenNode == this->linkedList->getSize())
            event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
        event.statusChosenNode = NodeInfo::StatusNode::OutChain;
        event.indexBackArrow = {this->linkedList->getSize(), 0};
        event.lines = {5};

        events.emplace_back(event);

        for (int i = 0; i < chosenNode; ++i) {
            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {chosenNode, "temp"},
                    {i, "current"}
            };
            event.colorNodes.push_back(i);
            event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
            if (chosenNode == this->linkedList->getSize())
                event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.indexBackArrow = {this->linkedList->getSize(), 0};
            event.lines = {6};

            events.emplace_back(event);

            if (i == chosenNode - 1) break;

            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {chosenNode, "temp"},
                    {i, "current"}
            };
            event.colorNodes.push_back(i);
            event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
            event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
            if (chosenNode == this->linkedList->getSize())
                event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.indexBackArrow = {this->linkedList->getSize(), 0};
            event.lines = {7};

            events.emplace_back(event);
        }

        if (chosenNode != this->linkedList->getSize()) {
            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {chosenNode, "temp"},
                    {chosenNode - 1, "current"}
            };
            event.colorNodes.push_back(chosenNode);
            event.colorArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.isPrintNormal = true;
            event.indexBackArrow = {this->linkedList->getSize(), 0};
            event.lines = {8};

            events.emplace_back(event);
        }

        event.reset();
        event.titleNodes = {
                {0, "head"},
                {chosenNode, "temp"}
        };
        event.statusChosenNode = NodeInfo::StatusNode::InChain;
        event.indexBackArrow = {this->linkedList->getSize(), 0};
        event.lines = {9};

        events.emplace_back(event);
    }

    return events;
}

std::vector<EventAnimation> CLLScene::deleteModeEvents(int chosenNode) {
    this->linkedList->resetEvents();
    if (chosenNode < 0 || chosenNode >= this->linkedList->getSize())
        return {};

    this->linkedList->initHighlighter(
            constants::Highlighter::SLL::CODES_PATH[1].second,
            constants::Highlighter::SLL::CODES_PATH[1].first
    );

    std::vector<EventAnimation> events;
    EventAnimation event;

    if (!chosenNode) {
        event.titleNodes.emplace_back(chosenNode, "head|temp");
        event.colorNodes.push_back(chosenNode);
        event.statusChosenNode = NodeInfo::StatusNode::InChain;
        event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
        event.lines = {0, 1};

        events.emplace_back(event);

        if (this->linkedList->getSize() > 1) {
            event.reset();
            event.titleNodes = {
                    {chosenNode, "temp"},
                    {1, "head"}
            };
            event.colorNodes.push_back(1);
            event.colorArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
            event.isPrintNormal = true;
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.indexBackArrow = {this->linkedList->getSize() - 1, 1};
            event.lines = {2};

            events.emplace_back(event);
        }

        event.reset();
        event.titleNodes.emplace_back(1, "head");
        event.statusChosenNode = NodeInfo::StatusNode::Visible;
        event.indexBackArrow = {this->linkedList->getSize() - 1, 1};
        event.lines = {3};

        events.emplace_back(event);
    } else {
        event.reset();
        event.titleNodes.emplace_back(0, "head|current");
        event.colorNodes.push_back(0);
        event.statusChosenNode = NodeInfo::StatusNode::InChain;
        event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
        event.lines = {5};

        events.emplace_back(event);

        for (int i = 0; i < chosenNode; ++i) {
            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {i, "current"}
            };
            event.colorNodes.push_back(i);
            event.statusChosenNode = NodeInfo::StatusNode::InChain;
            event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
            event.lines = {6};

            events.emplace_back(event);

            if (i == chosenNode - 1) break;

            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {i, "current"}
            };
            event.colorNodes.push_back(i);
            event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::InChain;
            event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
            event.lines = {7};

            events.emplace_back(event);
        }

        event.reset();
        event.titleNodes = {
                {0, "head"},
                {chosenNode, "temp"},
                {chosenNode - 1, "current"}
        };
        event.colorNodes.push_back(chosenNode);
        event.colorArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
        event.statusChosenNode = NodeInfo::StatusNode::InChain;
        event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
        event.lines = {8};

        events.emplace_back(event);

        if (chosenNode != this->linkedList->getSize() - 1) {
            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {chosenNode, "temp"},
                    {chosenNode - 1, "current"}
            };
            event.colorNodes.push_back(chosenNode);
            event.colorArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
            event.isPrintNormal = true;
            event.lines = {9};

            events.emplace_back(event);

            event.reset();
            event.titleNodes.emplace_back(0, "head");
            event.statusChosenNode = NodeInfo::StatusNode::Visible;
            event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
            event.lines = {10};

            events.emplace_back(event);
        } else {
            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {chosenNode, "temp"},
                    {chosenNode - 1, "current"}
            };
            event.colorNodes.push_back(chosenNode);
            event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.indexBackArrow = {chosenNode - 1, 0};
            event.lines = {9};

            events.emplace_back(event);

            event.reset();
            event.titleNodes.emplace_back(0, "head");
            event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::Visible;
            event.indexBackArrow = {chosenNode - 1, 0};
            event.lines = {10};

            events.emplace_back(event);
        }
    }

    return events;
}

std::vector<EventAnimation> CLLScene::updateModeEvents(int chosenNode) {
    this->linkedList->resetEvents();
    if (chosenNode < 0 || chosenNode >= this->linkedList->getSize())
        return {};

    this->linkedList->initHighlighter(
            constants::Highlighter::SLL::CODES_PATH[2].second,
            constants::Highlighter::SLL::CODES_PATH[2].first
    );

    std::vector<EventAnimation> events;
    EventAnimation event;

    event.titleNodes.emplace_back(0, "head|current");
    event.colorNodes.push_back(0);
    event.isPrintPreVal = true;
    event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
    event.lines = {0};

    events.emplace_back(event);

    if (chosenNode) {
        for (int i = 0; i <= chosenNode; ++i) {
            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {i, "current"}
            };
            event.colorNodes.push_back(i);
            event.isPrintPreVal = true;
            event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
            event.lines = {1};

            events.emplace_back(event);

            if (i == chosenNode) break;

            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {i, "current"}
            };
            event.colorNodes.push_back(i);
            event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
            event.isPrintPreVal = true;
            event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
            event.lines = {2};

            events.emplace_back(event);
        }
    }

    event.reset();
    if (chosenNode == 0)
        event.titleNodes.emplace_back(0, "head|current");
    else
        event.titleNodes = {
                {0, "head"},
                {chosenNode, "current"}
        };
    event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
    event.lines = {3};

    events.emplace_back(event);

    return events;
}

std::vector<EventAnimation> CLLScene::searchModeEvents(int chosenNode) {
    this->linkedList->resetEvents();
    this->linkedList->initHighlighter(
            constants::Highlighter::SLL::CODES_PATH[3].second,
            constants::Highlighter::SLL::CODES_PATH[3].first
    );

    std::vector<EventAnimation> events;
    EventAnimation event;

    event.titleNodes.emplace_back(0, "head|current");
    event.colorNodes.push_back(0);
    event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
    event.lines = {0};

    events.emplace_back(event);

    for (int i = 0; i <= chosenNode; ++i) {
        if (i == chosenNode && chosenNode == this->linkedList->getSize())
            break;

        event.reset();
        event.titleNodes = {
                {0, "head"},
                {i, "current"}
        };
        event.colorNodes.push_back(i);
        event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
        event.lines = {1};

        events.emplace_back(event);

        if (i == chosenNode) break;

        event.reset();
        event.titleNodes = {
                {0, "head"},
                {i, "current"}
        };
        event.colorNodes.push_back(i);
        event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
        event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
        event.lines = {4};

        events.emplace_back(event);
    }

    if (chosenNode == this->linkedList->getSize()) {
        event.reset();
        event.titleNodes.emplace_back(0, "head");
        event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
        event.lines = {5};

        events.emplace_back(event);
    } else {
        event.reset();
        event.titleNodes = {
                {0, "head"},
                {chosenNode, "current"}
        };
        event.colorNodes.push_back(chosenNode);
        event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
        event.lines = {2, 3};

        events.emplace_back(event);
    }

    return events;
}
```

8.57 include/libScene/CLLScene.hpp File Reference

```
#include "BaseScene.hpp"
#include "MenuLinkedList.hpp"
#include "core/LinkedList.hpp"
```

Classes

- class CLLScene

8.58 CLLScene.hpp

Go to the documentation of this file.

```
//
// Created by dirii on 28/03/2023.
//

#ifndef VISUALGO_CS162_CLLSCENE_HPP
#define VISUALGO_CS162_CLLSCENE_HPP

#include "BaseScene.hpp"
#include "MenuLinkedList.hpp"
#include "core/LinkedList.hpp"

class CLLScene : public BaseScene{
private:
    MenuLinkedList* menu;
    LinkedList* linkedList;

    void init();

public:
    explicit CLLScene(sf::RenderWindow* window);

    void reset();

    void pollEvent(sf::Event event, sf::Vector2f mousePosView) override;
    void update() override;
    void render() override;

    std::vector<EventAnimation> addModeEvents(int chosenNode);
    std::vector<EventAnimation> deleteModeEvents(int chosenNode);
    std::vector<EventAnimation> updateModeEvents(int chosenNode);
    std::vector<EventAnimation> searchModeEvents(int chosenNode);
};

#endif //VISUALGO_CS162_CLLSCENE_HPP
```

8.59 include/libScene/ControlMenu.cpp File Reference

```
#include "ControlMenu.hpp"
```

8.60 ControlMenu.cpp

Go to the documentation of this file.

```
//
// Created by dirii on 14/04/2023.
//

#include "ControlMenu.hpp"

ControlMenu::ControlMenu(sf::RenderWindow *window) {
    this->window = window;

    for (int i = 0; i < constants::ControlMenu::BUTTON_COUNT; ++i) {
        buttons[i] = new Button(
                this->window,
                constants::ControlMenu::buttonPos[i],
                constants::ControlMenu::buttonSize,
                constants::ControlMenu::BUTTON_NAMES[i],
                constants::ControlMenu::BUTTON_NAMES[i],
                constants::ControlMenu::BUTTON_NAME_SIZE,
                sf::Color::Black,
                constants::normalGray,
                constants::hoverGray,
                constants::clickGray
                );
    }

    this->font.loadFromFile(constants::fontPath);
    this->textSpeed.setFont(font);
    this->textSpeed.setString(to_string_with_precision(this->speed));
    this->textSpeed.setCharacterSize(constants::ControlMenu::TEXT_SIZE);
    this->textSpeed.setFillColor(sf::Color::Black);
    this->textSpeed.setOrigin(
            this->textSpeed.getLocalBounds().width / 2.0f,
            this->textSpeed.getLocalBounds().height / 2.0f
            );
    this->textSpeed.setPosition(
            constants::ControlMenu::buttonPos[3].x + constants::ControlMenu::buttonSize.x * 2,
            constants::ControlMenu::buttonPos[3].y + constants::ControlMenu::buttonSize.y / 2.0f
            );

    this->status = StatusCode::None;
    this->speed = 1;
}

void ControlMenu::pollEvents(sf::Event event, sf::Vector2f mousePosView) {
    for (int i = 0; i < constants::ControlMenu::BUTTON_COUNT; ++i) {
        if (buttons[i]->pollEvent(mousePosView)) {
            switch (i) {
                case 0:
                    this->status = StatusCode::PREVIOUS;
                    break;
                case 1:
                    if (this->status == StatusCode::PLAY)
                        this->status = StatusCode::PAUSE;
                    else
                        this->status = StatusCode::PLAY;
                    break;
                case 2:
                    this->status = StatusCode::NEXT;
                    break;
                case 3:
                    if (this->speed > 0.25)
                        this->speed -= 0.25;
                    break;
                case 4:
                    if (this->speed < 2)
                        this->speed += 0.25;
                    break;
                default:
                    this->status = StatusCode::None;
                    break;
            }
        }
    }
}

void ControlMenu::update() {
    for (auto &button : buttons) {
        button->update();
    }
    this->textSpeed.setString(to_string_with_precision(this->speed));
}

void ControlMenu::render() {
    for (auto &button : buttons) {
        button->render();
    }
    this->window->draw(this->textSpeed);
}

ControlMenu::StatusCode ControlMenu::getStatus() {
    ControlMenu::StatusCode temp = this->status;
    if (this->status == StatusCode::PREVIOUS || this->status == StatusCode::NEXT)
        this->status = StatusCode::PAUSE;
    return temp;
}

float ControlMenu::getSpeed() const {
    return this->speed;
}

void ControlMenu::reset() {
    this->status = StatusCode::None;
}
```

8.61 include/libScene/ControlMenu.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include "stuff/button.hpp"
```

```
#include "stuff/ToStringWithPrecision.hpp"
#include "Constants.hpp"
```

Classes

- class ControlMenu

8.62 ControlMenu.hpp

Go to the documentation of this file.

```
//
// Created by dirii on 14/04/2023.
//

#ifndef VISUALGO_CS162_CONTROLMENU_HPP
#define VISUALGO_CS162_CONTROLMENU_HPP

#include <SFML/Graphics.hpp>
#include "stuff/button.hpp"
#include "stuff/ToStringWithPrecision.hpp"
#include "Constants.hpp"

class ControlMenu {
private:
    sf::RenderWindow* window;

    Button* buttons[constants::ControlMenu::BUTTON_COUNT];
    sf::Font font;
    sf::Text textSpeed;
    float speed;

public:
    enum class StatusCode {
        PREVIOUS,
        PAUSE,
        PLAY,
        NEXT,
        None
    } status;

    explicit ControlMenu(sf::RenderWindow* window);
    ~ControlMenu() = default;

    void pollEvents(sf::Event event, sf::Vector2f mousePosView);
    void update();
    void render();
    void reset();

    ControlMenu::StatusCode getStatus();
    [[nodiscard]] float getSpeed() const;
};

#endif //VISUALGO_CS162_CONTROLMENU_HPP
```

8.63 include/libScene/DLLScene.cpp File Reference

```
#include "DLLScene.hpp"
```

8.64 DLLScene.cpp

Go to the documentation of this file.

```
//
// Created by dirii on 27/03/2023.
//

#include "DLLScene.hpp"

DLLScene::DLLScene(sf::RenderWindow *window) : BaseScene(window) {
    this->init();
}

void DLLScene::pollEvent(sf::Event event, sf::Vector2f mousePosView) {
    if (this->isMenuOpen)
        this->menu->pollEvents(event, mousePosView);

    this->controlMenu->pollEvents(event, mousePosView);
}

void DLLScene::update() {
    if (this->isMenuOpen) {
        this->menu->update();

        constants::MenuLinkedList::Button status = this->menu->getActiveOptionMenu();
        constants::MenuLinkedList::CreateMode::Button createMode;
        switch (status){
            case constants::MenuLinkedList::Button::CREATE_BUTTON:
                createMode = this->menu->getActiveCreateMode();
                if (createMode == constants::MenuLinkedList::CreateMode::Button::RANDOM_BUTTON) {
                    if (this->menu->createModeValue[0] == "None")
                        break;
                    if (this->menu->createModeValue[0].empty())
                        this->menu->createModeValue[0] = "0";
                    int size = std::stoi(this->menu->createModeValue[0]);
                    this->linkedList->createLinkedList(size);
                } else if (createMode ==
constants::MenuLinkedList::CreateMode::Button::DEFINED_LIST_BUTTON) {
                    if (this->menu->createModeValue[1] == "None")
                        break;
                    std::vector<std::string> values;
                    std::string value = this->menu->createModeValue[1];
                    std::stringstream ss(value);
                    std::string token;
                    while (std::getline(ss, token, ',')) {
                        values.push_back(token);
                    }
                    this->linkedList->createLinkedList(values);
                } else if (createMode == constants::MenuLinkedList::CreateMode::Button::FILE_BUTTON) {
                    if (this->menu->createModeValue[2] == "None")
                        break;
                    std::vector<std::string> values;
                    std::string value = this->menu->createModeValue[2];
                    std::stringstream ss(value);
                    std::string token;
                    while (std::getline(ss, token, ','))
                        values.push_back(token);
                    this->linkedList->createLinkedList(values);
                    this->menu->createModeValue[2] = "None";
                }
                this->controlMenu->reset();
                break;
            case constants::MenuLinkedList::Button::ADD_BUTTON:
                if (this->menu->addModeValue[0] == "None" || this->menu->addModeValue[1] == "None" ||
this->menu->addModeValue[0].empty())
                    break;

                this->linkedList->addNode(
                        std::stoi(this->menu->addModeValue[0]),
                        this->menu->addModeValue[1],
                        this->addModeEvents(std::stoi(this->menu->addModeValue[0]))
                );

                std::cout « "Add: " « this->menu->addModeValue[0] « " " « this->menu->addModeValue[1]
« std::endl;
                this->menu->addModeValue[0] = this->menu->addModeValue[1] = "None";
                this->controlMenu->reset();
                break;
            case constants::MenuLinkedList::Button::DELETE_BUTTON:
                if (this->menu->deleteModeValue == "None" || this->menu->deleteModeValue.empty())
                    break;

                this->linkedList->deleteNode(
                        std::stoi(this->menu->deleteModeValue),
                        this->deleteModeEvents(std::stoi(this->menu->deleteModeValue))
                );

                std::cout « "Delete: " « this->menu->deleteModeValue « std::endl;
                this->menu->deleteModeValue = "None";
                this->controlMenu->reset();
                break;
            case constants::MenuLinkedList::Button::UPDATE_BUTTON:
                if (this->menu->updateModeValue[0] == "None" || this->menu->updateModeValue[1] ==
"None" || this->menu->updateModeValue[0].empty())
                    break;

                this->linkedList->updateNode(
                        std::stoi(this->menu->updateModeValue[0]),
                        this->menu->updateModeValue[1],
                        this->updateModeEvents(std::stoi(this->menu->updateModeValue[0]))
                );

                std::cout « "Update: " « this->menu->updateModeValue[0] « " " «
this->menu->updateModeValue[1] « std::endl;
                this->menu->updateModeValue[0] = this->menu->updateModeValue[1] = "None";
                this->controlMenu->reset();
                break;
            case constants::MenuLinkedList::Button::SEARCH_BUTTON:
                if (this->menu->searchModeValue == "None" || this->menu->searchModeValue.empty())
                    break;

                this->linkedList->searchNode(

this->searchModeEvents(this->linkedList->findValue(this->menu->searchModeValue))
                );

                std::cout « "Search: " « this->menu->searchModeValue « std::endl;
                this->menu->searchModeValue = "None";
                this->controlMenu->reset();
                break;
        }
    }

    this->controlMenu->update();

    this->linkedList->processControlMenu(this->controlMenu->getStatus());
    this->linkedList->setSpeed(this->controlMenu->getSpeed());

    this->linkedList->update();
}

void DLLScene::render() {
    if (this->isMenuOpen)
        this->menu->render();

    if (this->isDemoCodeOpen)
        this->linkedList->renderHighlighter();

    this->controlMenu->render();
    this->linkedList->render();
}

void DLLScene::init() {
    this->menu = new MenuLinkedList(this->window);
    this->linkedList = new LinkedList(this->window, LinkedList::TypeLinkedList::DOUBLY);
}

void DLLScene::reset() {
    this->menu->resetActiveOptionMenu();
}

std::vector<EventAnimation> DLLScene::addModeEvents(int chosenNode) {
    this->linkedList->resetEvents();
    if (chosenNode < 0 || chosenNode > this->linkedList->getSize())
        return {};

    std::vector<EventAnimation> events;
    EventAnimation event;
    int size = this->linkedList->getSize();

    if (chosenNode == 0) {
        this->linkedList->initHighlighter(
                constants::Highlighter::DLL::CODES_PATH[0].second,
                constants::Highlighter::DLL::CODES_PATH[0].first
        );

        event.titleNodes.emplace_back(chosenNode, "temp");
        if (size == 1)
            event.titleNodes.emplace_back(1, "head|tail");
        else if (size > 1){
            event.titleNodes.emplace_back(1, "head");
            event.titleNodes.emplace_back(size, "tail");
        }
        if (size)
            event.hiddenArrows.emplace_back(1, NodeInfo::ArrowType::LEFT);
        event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
        event.colorNodes.emplace_back(chosenNode);
        event.statusChosenNode = NodeInfo::StatusNode::OutChain;
        event.lines = {0, 1};

        events.emplace_back(event);

        event.reset();

        event.titleNodes.emplace_back(chosenNode, "temp");
        if (size == 1)
            event.titleNodes.emplace_back(1, "head|tail");
        else if (size > 1){
            event.titleNodes.emplace_back(1, "head");
            event.titleNodes.emplace_back(size, "tail");
        }
        if (size)
            event.hiddenArrows.emplace_back(1, NodeInfo::ArrowType::LEFT);
        event.colorArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
        event.colorNodes.emplace_back(chosenNode);
        event.statusChosenNode = NodeInfo::StatusNode::OutChain;
        event.isPrintNormal = true;
        event.lines = {2};

        events.emplace_back(event);

        if (size) {
            event.reset();
            event.titleNodes.emplace_back(chosenNode, "temp");
            if (size == 1)
                event.titleNodes.emplace_back(1, "head|tail");
            else if (size > 1){
                event.titleNodes.emplace_back(1, "head");
                event.titleNodes.emplace_back(size, "tail");
            }
            event.colorArrows = {
//                    {chosenNode, NodeInfo::ArrowType::RIGHT},
                    {chosenNode + 1, NodeInfo::ArrowType::LEFT}
            };
            event.colorNodes.emplace_back(chosenNode + 1);
            event.statusChosenNode = NodeInfo::StatusNode::InChain;
            event.lines = {3, 4};

            events.emplace_back(event);
        }

        event.reset();
        if (size) {
            event.titleNodes = {
                    {chosenNode, "head"},
                    {size,       "tail"}
            };
            event.lines = {7};
        }
        else {
            event.titleNodes.emplace_back(chosenNode, "head|tail");
            event.lines = {5, 6, 7};
        }
        event.colorNodes = {chosenNode};

        events.emplace_back(event);
    }
    else if (chosenNode == size) {
        this->linkedList->initHighlighter(
                constants::Highlighter::DLL::CODES_PATH[1].second,
                constants::Highlighter::DLL::CODES_PATH[1].first
        );

        event.titleNodes.emplace_back(chosenNode, "temp");
        if (size == 1)
            event.titleNodes.emplace_back(0, "head|tail");
        else if (size > 1){
            event.titleNodes.emplace_back(0, "head");
            event.titleNodes.emplace_back(size - 1, "tail");
        }
        event.hiddenArrows.emplace_back(size - 1, NodeInfo::ArrowType::RIGHT);
        event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::LEFT);
        event.colorNodes.emplace_back(chosenNode);
        event.statusChosenNode = NodeInfo::StatusNode::OutChain;
        event.lines = {0, 1};

        events.emplace_back(event);

        event.reset();

        event.titleNodes.emplace_back(chosenNode, "temp");
        if (size == 1)
            event.titleNodes.emplace_back(0, "head|tail");
        else if (size > 1){
            event.titleNodes.emplace_back(0, "head");
            event.titleNodes.emplace_back(size - 1, "tail");
        }
        event.hiddenArrows.emplace_back(size - 1, NodeInfo::ArrowType::RIGHT);
        event.colorArrows.emplace_back(chosenNode, NodeInfo::ArrowType::LEFT);
        event.colorNodes.emplace_back(chosenNode);
        event.statusChosenNode = NodeInfo::StatusNode::OutChain;
        event.lines = {2};

        events.emplace_back(event);

        event.reset();
        event.titleNodes.emplace_back(chosenNode, "temp");
        if (size == 1)
            event.titleNodes.emplace_back(0, "head|tail");
        else if (size > 1){
            event.titleNodes.emplace_back(0, "head");
            event.titleNodes.emplace_back(size - 1, "tail");
        }
        event.colorArrows = {
//                  {chosenNode, NodeInfo::ArrowType::LEFT},
                {chosenNode - 1, NodeInfo::ArrowType::RIGHT}
        };
        event.colorNodes.emplace_back(chosenNode - 1);
        event.statusChosenNode = NodeInfo::StatusNode::InChain;
        event.lines = {3};

        events.emplace_back(event);

        event.reset();
        event.titleNodes = {
                {chosenNode, "tail"},
                {0, "head"}
        };
        event.colorNodes = {chosenNode};
        event.lines = {4};

        events.emplace_back(event);
    }
    else {
        this->linkedList->initHighlighter(
                constants::Highlighter::DLL::CODES_PATH[2].second,
                constants::Highlighter::DLL::CODES_PATH[2].first
        );

        event.titleNodes = {
                {chosenNode, "temp"},
                {0,          "head"},
                {size,       "tail"}
        };
        event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
        event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::LEFT);
        event.colorNodes.emplace_back(chosenNode);
        event.statusChosenNode = NodeInfo::StatusNode::OutChain;
        event.lines = {0, 1};

        events.emplace_back(event);

        event.reset();
        event.titleNodes = {
                {chosenNode, "temp"},
                {0,          "head|current"},
                {size,       "tail"}
        };
        event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
        event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::LEFT);
        event.colorNodes.emplace_back(0);
        event.statusChosenNode = NodeInfo::StatusNode::OutChain;
        event.lines = {2};

        events.emplace_back(event);

        for (int i = 0; i < chosenNode; ++i) {
            event.reset();
            event.titleNodes = {
                    {chosenNode, "temp"},
                    {0,          "head"},
                    {size,       "tail"},
                    {i, "current"}
            };
            event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
            event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::LEFT);
            event.colorNodes.emplace_back(i);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.lines = {3};

            events.emplace_back(event);

            if (i == chosenNode - 1)
                break;

            event.reset();
            event.titleNodes = {
                    {chosenNode, "temp"},
                    {0,          "head"},
                    {size,       "tail"},
                    {i, "current"}
            };
            event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
            event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::LEFT);
            event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
//              event.colorArrows.emplace_back(i + 1 + (i + 1 == chosenNode),
NodeInfo::ArrowType::LEFT);
            event.colorNodes.emplace_back(i);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.lines = {4};

            events.emplace_back(event);
        }

        event.reset();
        event.titleNodes = {
                {chosenNode, "temp"},
                {0,          "head"},
                {size,       "tail"},
                {chosenNode - 1, "current"}
        };
        event.colorArrows = {
                {chosenNode, NodeInfo::ArrowType::RIGHT},
                {chosenNode, NodeInfo::ArrowType::LEFT}
        };
        event.colorNodes.emplace_back(chosenNode);
        event.statusChosenNode = NodeInfo::StatusNode::OutChain;
        event.isPrintNormal = true;
        event.lines = {5, 6};

        events.emplace_back(event);

        event.reset();
        event.titleNodes = {
                {chosenNode, "temp"},
                {0,          "head"},
                {size,       "tail"}
        };
        event.colorNodes.emplace_back(chosenNode);
        event.statusChosenNode = NodeInfo::StatusNode::InChain;
        event.lines = {7, 8};

        events.emplace_back(event);
    }

    return events;
}

std::vector<EventAnimation> DLLScene::deleteModeEvents(int chosenNode) {
    this->linkedList->resetEvents();
    if (chosenNode < 0 || chosenNode >= this->linkedList->getSize())
        return {};

    std::vector<EventAnimation> events;
    EventAnimation event;
    int size = this->linkedList->getSize();

    if (chosenNode == 0) {
        this->linkedList->initHighlighter(
                constants::Highlighter::DLL::CODES_PATH[3].second,
                constants::Highlighter::DLL::CODES_PATH[3].first
        );

        if (size == 1) {
            event.titleNodes.emplace_back(chosenNode, "head|tail|temp");
            event.colorNodes.emplace_back(chosenNode);
            event.statusChosenNode = NodeInfo::StatusNode::InChain;
            event.lines = {0, 1, 2};

            events.emplace_back(event);

            event.reset();
            event.statusChosenNode = NodeInfo::StatusNode::Visible;
            event.lines = {5, 6, 7};

            events.emplace_back(event);
        }
        else {
            event.titleNodes = {
                    {chosenNode, "head|temp"},
                    {size - 1, "tail"}
            };
            event.colorNodes.emplace_back(chosenNode);
            event.lines = {0, 1};

            events.emplace_back(event);

            event.reset();
            if (size == 2)
                event.titleNodes.emplace_back(size - 1, "head|tail");
            else
                event.titleNodes = {
                    {size - 1, "tail"},
                    {chosenNode + 1, "head" }
            };
            event.titleNodes.emplace_back(chosenNode, "temp");
            event.colorNodes.emplace_back(chosenNode + 1);
//            event.isPrintNormal = true;
//            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.lines = {2};

            events.emplace_back(event);

            event.reset();
            if (size == 2)
                event.titleNodes.emplace_back(size - 1, "head|tail");
            else
                event.titleNodes = {
                        {size - 1, "tail"},
                        {chosenNode + 1, "head" }
                };
            event.titleNodes.emplace_back(chosenNode, "temp");
            event.colorNodes.emplace_back(chosenNode);
            event.hiddenArrows = {
//                    {chosenNode, NodeInfo::ArrowType::RIGHT},
                    {chosenNode + 1, NodeInfo::ArrowType::LEFT}
            };
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.isPrintNormal = true;
            event.lines = {3, 4};

            events.emplace_back(event);

            event.reset();
            if (size == 2)
                event.titleNodes.emplace_back(size - 1, "head|tail");
            else
                event.titleNodes = {
                        {size - 1, "tail"},
                        {chosenNode + 1, "head" }
                };
            event.hiddenArrows = {
                    {chosenNode + 1, NodeInfo::ArrowType::LEFT}
            };
            event.statusChosenNode = NodeInfo::StatusNode::Visible;
            event.lines = {7};

            events.emplace_back(event);
        }
    }
    else if (chosenNode == size - 1) {
        this->linkedList->initHighlighter(
                constants::Highlighter::DLL::CODES_PATH[4].second,
                constants::Highlighter::DLL::CODES_PATH[4].first
                );

        event.titleNodes = {
                {0, "head"},
                {chosenNode, "tail|temp"}
        };
        event.colorNodes.emplace_back(chosenNode);
        event.lines = {0, 1};

        events.emplace_back(event);

        event.reset();
        if (size == 2)
            event.titleNodes.emplace_back(0, "head|tail");
        else
            event.titleNodes = {
                    {chosenNode - 1, "tail"},
                    {0, "head" }
            };
        event.titleNodes.emplace_back(chosenNode, "temp");
        event.colorNodes.emplace_back(chosenNode - 1);
        event.lines = {2};

        events.emplace_back(event);

        event.reset();
        if (size == 2)
            event.titleNodes.emplace_back(0, "head|tail");
        else
            event.titleNodes = {
                    {chosenNode - 1, "tail"},
                    {0, "head" }
            };
        event.titleNodes.emplace_back(chosenNode, "temp");
        event.colorNodes.emplace_back(chosenNode);
        event.hiddenArrows = {
                {chosenNode - 1, NodeInfo::ArrowType::RIGHT}
        };
        event.statusChosenNode = NodeInfo::StatusNode::OutChain;
        event.isPrintNormal = true;
        event.lines = {3};

        events.emplace_back(event);

        event.reset();
        if (size == 2)
            event.titleNodes.emplace_back(0, "head|tail");
        else
            event.titleNodes = {
                    {chosenNode - 1, "tail"},
                    {0, "head" }
            };
        event.hiddenArrows = {
                {chosenNode - 1, NodeInfo::ArrowType::RIGHT}
        };
        event.statusChosenNode = NodeInfo::StatusNode::Visible;
        event.lines = {4};

        events.emplace_back(event);
    }
    else {
        this->linkedList->initHighlighter(
                constants::Highlighter::DLL::CODES_PATH[5].second,
                constants::Highlighter::DLL::CODES_PATH[5].first
        );

        event.titleNodes = {
                {0, "head|temp"},
                {size - 1, "tail"}
        };
        event.colorNodes.emplace_back(0);
        event.lines = {0, 1};

        events.emplace_back(event);

        for (int i = 0; i <= chosenNode; ++i) {
            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {i, "temp"},
                    {size - 1, "tail"}
            };
            event.colorNodes.emplace_back(i);
            event.lines = {2};

            events.emplace_back(event);

            if (i == chosenNode)
                break;

            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {i, "temp"},
                    {size - 1, "tail"}
            };
            event.colorNodes.emplace_back(i);
            event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
            event.lines = {3};

            events.emplace_back(event);
        }

        event.reset();
        event.titleNodes = {
                {0, "head"},
                {chosenNode, "temp"},
                {size - 1, "tail"}
        };
        event.colorNodes.emplace_back(chosenNode);
        event.colorArrows = {
                {chosenNode - 1, NodeInfo::ArrowType::RIGHT},
                {chosenNode + 1, NodeInfo::ArrowType::LEFT}
        };
        event.statusChosenNode = NodeInfo::StatusNode::OutChain;
        event.isPrintNormal = true;
        event.lines = {4, 5};

        events.emplace_back(event);

        event.reset();
        event.titleNodes = {
                {0, "head"},
                {size - 1, "tail"}
        };
        event.statusChosenNode = NodeInfo::StatusNode::Visible;
        event.lines = {6};

        events.emplace_back(event);
    }

    return events;
}

std::vector<EventAnimation> DLLScene::updateModeEvents(int chosenNode) {
    this->linkedList->resetEvents();
    if (chosenNode < 0 || chosenNode >= this->linkedList->getSize())
        return {};

    this->linkedList->initHighlighter(
            constants::Highlighter::DLL::CODES_PATH[6].second,
            constants::Highlighter::DLL::CODES_PATH[6].first
    );

    std::vector<EventAnimation> events;
    EventAnimation event;
    int size = this->linkedList->getSize();

    if (size > 1)
        event.titleNodes = {
                {0, "head|current"},
                {size - 1, "tail"}
        };
    else
        event.titleNodes = {
                {0, "head|tail|current"}
        };
    event.colorNodes.push_back(0);
    event.isPrintPreVal = true;
    event.lines = {0};

    events.emplace_back(event);

    if (chosenNode) {
        for (int i = 0; i <= chosenNode; ++i) {
            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {size - 1, "tail"},
                    {i, "current"},
            };
            event.colorNodes.push_back(i);
            event.isPrintPreVal = true;
            event.lines = {1};

            events.emplace_back(event);

            if (i == chosenNode) break;

            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {i, "current"},
                    {size - 1, "tail"}
            };
            event.colorNodes.push_back(i);
            event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
            event.isPrintPreVal = true;
            event.lines = {2};

            events.emplace_back(event);
        }
    }

    event.reset();
    if (size == 1)
        event.titleNodes = {
                {0, "head|tail|current"}
        };
    else if (chosenNode == size - 1)
        event.titleNodes = {
                {0, "head"},
                {chosenNode, "current|tail"}
        };
    else if (chosenNode == 0)
        event.titleNodes = {
                {0, "head|current"},
                {size - 1, "tail"}
        };
    else
        event.titleNodes = {
                {0,          "head"},
                {chosenNode, "current"},
                {size - 1,   "tail"}
        };
    event.lines = {3};

    events.emplace_back(event);

    return events;
}

std::vector<EventAnimation> DLLScene::searchModeEvents(int chosenNode) {
    this->linkedList->resetEvents();
    this->linkedList->initHighlighter(
            constants::Highlighter::DLL::CODES_PATH[7].second,
            constants::Highlighter::DLL::CODES_PATH[7].first
    );

    std::vector<EventAnimation> events;
    EventAnimation event;
    int size = this->linkedList->getSize();

    if (size > 1)
        event.titleNodes = {
                {0, "head|current"},
                {size - 1, "tail"}
        };
    else
        event.titleNodes = {
                {0, "head|tail|current"}
        };
    event.colorNodes.push_back(0);
    event.lines = {0};

    events.emplace_back(event);

    for (int i = 0; i <= chosenNode; ++i) {
        if (i == chosenNode && chosenNode == this->linkedList->getSize())
            break;

        event.reset();
        event.titleNodes = {
                {0, "head"},
                {size - 1, "tail"},
                {i, "current"}
        };
        event.colorNodes.push_back(i);
        event.lines = {1};

        events.emplace_back(event);

        if (i == chosenNode) break;

        event.reset();
        event.titleNodes = {
                {0, "head"},
                {size - 1, "tail"},
                {i, "current"}
        };
        event.colorNodes.push_back(i);
        event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
        event.lines = {4};

        events.emplace_back(event);
    }

    if (chosenNode == this->linkedList->getSize()) {
        event.reset();
        event.titleNodes.emplace_back(0, "head");
        event.titleNodes.emplace_back(size - 1, "tail");
        event.lines = {5};

        events.emplace_back(event);
    } else {
        event.reset();
        event.titleNodes = {
                {0, "head"},
                {size - 1, "tail"},
                {chosenNode, "current"}
        };
        event.colorNodes.push_back(chosenNode);
        event.lines = {2, 3};

        events.emplace_back(event);
    }

    return events;
}
```

8.65 include/libScene/DLLScene.hpp File Reference

```
#include "BaseScene.hpp"
#include "MenuLinkedList.hpp"
#include "core/LinkedList.hpp"
```

Classes

- class DLLScene

8.66 DLLScene.hpp

Go to the documentation of this file.

```
//
// Created by dirii on 27/03/2023.
//

#ifndef VISUALGO_CS162_DLLSCENE_HPP
#define VISUALGO_CS162_DLLSCENE_HPP

#include "BaseScene.hpp"
#include "MenuLinkedList.hpp"
#include "core/LinkedList.hpp"

class DLLScene : public BaseScene {
private:
    MenuLinkedList* menu;
    LinkedList* linkedList;

    void init();

public:
    explicit DLLScene(sf::RenderWindow* window);

    void reset();

    void pollEvent(sf::Event event, sf::Vector2f mousePosView) override;
    void update() override;
    void render() override;

    std::vector<EventAnimation> addModeEvents(int chosenNode);
    std::vector<EventAnimation> deleteModeEvents(int chosenNode);
    std::vector<EventAnimation> updateModeEvents(int chosenNode);
    std::vector<EventAnimation> searchModeEvents(int chosenNode);
};

#endif //VISUALGO_CS162_DLLSCENE_HPP
```

8.67 include/libScene/DynamicArrayScene.cpp File Reference

```
#include "DynamicArrayScene.hpp"
```

8.68 DynamicArrayScene.cpp

Go to the documentation of this file.

```
//
// Created by dirii on 27/03/2023.
//

#include "DynamicArrayScene.hpp"

DynamicArrayScene::DynamicArrayScene(sf::RenderWindow *window) : BaseScene(window) {
    this->init();
}

void DynamicArrayScene::update() {
    if (this->isMenuOpen) {
        this->menu->update();

        constants::MenuArray::Button status = this->menu->getActiveOptionMenu();
        constants::MenuArray::CreateMode::Button createMode;
        switch (status){
            case constants::MenuArray::Button::CREATE_BUTTON:
                createMode = this->menu->getActiveCreateMode();
                if (createMode == constants::MenuArray::CreateMode::Button::RANDOM_BUTTON) {
                    if (this->menu->createModeValue[0] == "None")
                        break;
                    if (this->menu->createModeValue[0].empty())
                        this->menu->createModeValue[0] = "0";
                    int size = std::stoi(this->menu->createModeValue[0]);
                    this->array->createArray(size);
                } else if (createMode ==
constants::MenuArray::CreateMode::Button::DEFINED_LIST_BUTTON) {
                    if (this->menu->createModeValue[1] == "None")
                        break;
                    std::vector<std::string> values;
                    std::string value = this->menu->createModeValue[1];
                    std::stringstream ss(value);
                    std::string token;
                    while (std::getline(ss, token, ',')) {
                        values.push_back(token);
                    }
                    this->array->createArray(values);
                } else if (createMode == constants::MenuArray::CreateMode::Button::FILE_BUTTON) {
                    if (this->menu->createModeValue[2] == "None")
                        break;
                    std::vector<std::string> values;
                    std::string value = this->menu->createModeValue[2];
                    std::stringstream ss(value);
                    std::string token;
                    while (std::getline(ss, token, ','))
                        values.push_back(token);
                    this->array->createArray(values);
                    this->menu->createModeValue[2] = "None";
                }
                this->controlMenu->reset();
                break;
            case constants::MenuArray::Button::ADD_BUTTON:
                if (this->menu->addModeValue[0] == "None" || this->menu->addModeValue[1] == "None" ||
this->menu->addModeValue[0].empty())
                    break;

                this->array->addSquare(
                        std::stoi(this->menu->addModeValue[0]),
                        this->menu->addModeValue[1],
                        this->addModeEvents(std::stoi(this->menu->addModeValue[0]))
                );

                std::cout « "Add: " « this->menu->addModeValue[0] « " " « this->menu->addModeValue[1]
« std::endl;
                this->menu->addModeValue[0] = this->menu->addModeValue[1] = "None";
                this->controlMenu->reset();
                break;
            case constants::MenuArray::Button::DELETE_BUTTON:
                if (this->menu->deleteModeValue == "None" || this->menu->deleteModeValue.empty())
```

```
                    break;

                this->array->deleteSquare(
                        std::stoi(this->menu->deleteModeValue),
                        this->deleteModeEvents(std::stoi(this->menu->deleteModeValue))
                );

                std::cout « "Delete: " « this->menu->deleteModeValue « std::endl;
                this->menu->deleteModeValue = "None";
                this->controlMenu->reset();
                break;
            case constants::MenuArray::Button::UPDATE_BUTTON:
                if (this->menu->updateModeValue[0] == "None" || this->menu->updateModeValue[1] == "None" || this->menu->updateModeValue[0].empty())
                    break;

                this->array->updateSquare(
                        std::stoi(this->menu->updateModeValue[0]),
                        this->menu->updateModeValue[1],
                        this->updateModeEvents(std::stoi(this->menu->updateModeValue[0]))
                );

                std::cout « "Update: " « this->menu->updateModeValue[0] « " " « this->menu->updateModeValue[1] « std::endl;
                this->menu->updateModeValue[0] = this->menu->updateModeValue[1] = "None";
                this->controlMenu->reset();
                break;
            case constants::MenuArray::Button::SEARCH_BUTTON:
                if (this->menu->searchModeValue == "None" || this->menu->searchModeValue.empty())
                    break;

                this->array->searchSquare(
                        this->searchModeEvents(this->array->findValue(this->menu->searchModeValue))
                );

                std::cout « "Search: " « this->menu->searchModeValue « std::endl;
                this->menu->searchModeValue = "None";
                this->controlMenu->reset();
                break;
            case constants::MenuArray::Button::ALLOCATE_BUTTON:
                if (this->menu->allocateModeValue == "None" || this->menu->allocateModeValue.empty())
                    break;

                this->array->allocateSquare(
                        std::stoi(this->menu->allocateModeValue),
                        this->allocateModeEvents(std::stoi(this->menu->allocateModeValue))
                );

                std::cout « "Allocate: " « this->menu->allocateModeValue « std::endl;
                this->menu->allocateModeValue = "None";
                this->controlMenu->reset();
                break;
        }
    }

    this->controlMenu->update();

    this->array->processControlMenu(this->controlMenu->getStatus());
    this->array->setSpeed(this->controlMenu->getSpeed());

    this->array->update();
}

void DynamicArrayScene::render() {
    if (this->isMenuOpen)
        this->menu->render();

    if (this->isDemoCodeOpen)
        this->array->renderHighlighter();

    this->controlMenu->render();
    this->array->render();
}

void DynamicArrayScene::pollEvent(sf::Event event, sf::Vector2f mousePosView) {
    if (this->isMenuOpen)
        this->menu->pollEvents(event, mousePosView);

    this->controlMenu->pollEvents(event, mousePosView);
}

void DynamicArrayScene::init() {
    this->menu = new MenuArray(this->window, constants::MenuArray::Type::DYNAMIC);
    this->array = new Array(this->window, Array::TypeArray::DYNAMIC);
}

void DynamicArrayScene::reset() {
    this->menu->resetActiveOptionMenu();
}

std::vector<EventAnimation> DynamicArrayScene::addModeEvents(int chosenNode) {
    this->array->resetEvents();
    if (chosenNode < 0 || chosenNode > this->array->getSize())
        return {};

    // init highlighter
    // ...

    int size = this->array->getSize() + 1,
        squaresSize = this->array->getSquaresSize();
    std::vector<EventAnimation> events;
    EventAnimation event;

    if (size > squaresSize) {
        ++squaresSize;
        event.eventSquares.assign(squaresSize, EventSquare());
        event.eventSquaresTemp.assign(squaresSize, EventSquare());
        for (auto &square : event.eventSquares) {
            square.status = Square::Status::active;
            square.isPrintPreVal = true;
        }
        event.eventSquares.back().status = Square::Status::hidden;
        if (size > 1)
            event.eventSquares[size - 2].title = "n";
        for (auto &square : event.eventSquaresTemp) {
            square.status = Square::Status::inactive;
            square.isPrintPreVal = true;
        }

        events.emplace_back(event);

        for (int i = 0; i < size - 1; ++i) {
            event = EventAnimation();
            event.eventSquares.assign(squaresSize, EventSquare());
            event.eventSquaresTemp.assign(squaresSize, EventSquare());
            for (auto &square : event.eventSquares) {
                square.status = Square::Status::active;
                square.isPrintPreVal = true;
            }
            event.eventSquares.back().status = Square::Status::hidden;
            if (size > 1)
                event.eventSquares[size - 2].title = "n";
            for (auto &square : event.eventSquaresTemp) {
                square.status = Square::Status::inactive;
                square.isPrintPreVal = true;
            }
            for (int j = 0; j < i; ++j) {
                event.eventSquaresTemp[j].status = Square::Status::active;
                event.eventSquaresTemp[j].isPrintPreVal = false;
            }
            event.eventSquaresTemp[i].status = Square::Status::chosen;
            event.eventSquaresTemp[i].title = "m";

            events.emplace_back(event);

            event.eventSquaresTemp[i].isPrintPreVal = false;
            event.eventSquares[i].status = Square::Status::chosen;

            events.emplace_back(event);
        }
    }

    event = EventAnimation();
    event.eventSquares.assign(squaresSize, EventSquare());
    event.eventSquaresTemp.assign(squaresSize, EventSquare());
    for (auto &square : event.eventSquares) {
        square.status = Square::Status::active;
        square.isPrintPreVal = true;
    }
    for (int i = size - 1; i < squaresSize; ++i)
        event.eventSquares[i].status = Square::Status::inactive;
    if (size > 1)
        event.eventSquares[size - 2].title = "n";
    for (auto &square : event.eventSquaresTemp) {
        square.status = Square::Status::hidden;
    }

    events.emplace_back(event);

    event = EventAnimation();
    event.eventSquares.assign(squaresSize, EventSquare());
    event.eventSquaresTemp.assign(squaresSize, EventSquare());
    for (auto &square : event.eventSquares) {
        square.status = Square::Status::active;
```

```
            square.isPrintPreVal = true;
        }
        for (int i = size; i < squaresSize; ++i)
            event.eventSquares[i].status = Square::Status::inactive;
        event.eventSquares[size - 1].title = "n";
        for (auto &square : event.eventSquaresTemp)
            square.status = Square::Status::hidden;

        events.emplace_back(event);

        for (int i = size - 1; i >= chosenNode; --i) {
            event = EventAnimation();
            event.eventSquares.assign(squaresSize, EventSquare());
            event.eventSquaresTemp.assign(squaresSize, EventSquare());
            for (auto &square: event.eventSquares) {
                square.status = Square::Status::active;
                square.isPrintPreVal = true;
            }
            for (int j = size; j < squaresSize; ++j)
                event.eventSquares[j].status = Square::Status::inactive;
            event.eventSquares[size - 1].title = "n";
            for (int j = size - 1; j > i; --j)
                event.eventSquares[j].isPrintPreVal = false;
            event.eventSquares[i].status = Square::Status::chosen;
            for (auto &square : event.eventSquaresTemp)
                square.status = Square::Status::hidden;

            events.emplace_back(event);

            event.eventSquares[i].isPrintPreVal = false;
            if (i > chosenNode)
                event.eventSquares[i - 1].status = Square::Status::chosen;

            events.emplace_back(event);
        }

        return events;
    }

    std::vector<EventAnimation> DynamicArrayScene::deleteModeEvents(int chosenNode) {
        this->array->resetEvents();
        if (chosenNode < 0 || chosenNode >= this->array->getSize())
            return {};

        // init highlighter
        // ...

        int size = this->array->getSize(),
            squaresSize = this->array->getSquaresSize();
        std::vector<EventAnimation> events;
        EventAnimation event;

        for (int i = chosenNode; i < size - 1; ++i) {
            event = EventAnimation();
            event.eventSquares.assign(squaresSize, EventSquare());
            for (auto &square : event.eventSquares) {
                square.status = Square::Status::active;
                square.isPrintPreVal = true;
            }
            for (int j = size; j < squaresSize; ++j)
                event.eventSquares[j].status = Square::Status::inactive;
            for (int j = 0; j < i; ++j)
                event.eventSquares[j].isPrintPreVal = false;
            event.eventSquares[i].status = Square::Status::chosen;
            for (auto &square : event.eventSquaresTemp)
                square.status = Square::Status::hidden;
            event.eventSquares[size - 1].title = "n";

            events.emplace_back(event);

            event.eventSquares[i].isPrintPreVal = false;
            event.eventSquares[i + 1].status = Square::Status::chosen;

            events.emplace_back(event);
        }

        event = EventAnimation();
        event.eventSquares.assign(squaresSize, EventSquare());
        for (int i = 0; i < size - 1; ++i) {
            event.eventSquares[i].status = Square::Status::active;
            if (i == size - 2)
                event.eventSquares[i].title = "n";
        }
        for (int i = size - 1; i < squaresSize; ++i)
            event.eventSquares[i].status = Square::Status::inactive;

        events.emplace_back(event);
```

```cpp

    return events;
}

std::vector<EventAnimation> DynamicArrayScene::updateModeEvents(int chosenNode) {
    this->array->resetEvents();
    if (chosenNode < 0 || chosenNode >= this->array->getSize())
        return {};

    // init highlighter
    // ...

    std::vector<EventAnimation> events;
    EventAnimation event;

    event = EventAnimation();
    event.eventSquares.assign(this->array->getSquaresSize(), EventSquare());
    for (int i = 0; i < this->array->getSize(); ++i) {
        event.eventSquares[i].status = Square::Status::active;
        if (i == this->array->getSize() - 1)
            event.eventSquares[this->array->getSize() - 1].title = "n";
    }
    event.eventSquares[chosenNode].status = Square::Status::chosen;
    event.eventSquares[chosenNode].isPrintPreVal = true;

    events.emplace_back(event);

    event.eventSquares[chosenNode].isPrintPreVal = false;

    events.emplace_back(event);

    return events;
}

std::vector<EventAnimation> DynamicArrayScene::searchModeEvents(int chosenNode) {
    this->array->resetEvents();

    // init highlighter
    // ...

    int size = this->array->getSize(),
        squaresSize = this->array->getSquaresSize();
    std::vector<EventAnimation> events;
    EventAnimation event;

    for (int i = 0; i <= chosenNode; ++i) {
        if (i == size) break;

        event = EventAnimation();
        event.eventSquares.assign(squaresSize, EventSquare());
        for (int j = 0; j < size; ++j) {
            event.eventSquares[j].status = Square::Status::active;
            if (j == size - 1)
                event.eventSquares[size - 1].title = "n";
        }
        event.eventSquares[i].status = Square::Status::chosen;

        events.emplace_back(event);
    }

    if (chosenNode == size) {
        event = EventAnimation();
        event.eventSquares.assign(squaresSize, EventSquare());
        for (int j = 0; j < size; ++j) {
            event.eventSquares[j].status = Square::Status::active;
            if (j == size - 1)
                event.eventSquares[size - 1].title = "n";
        }

        events.emplace_back(event);
    }

    return events;
}

std::vector<EventAnimation> DynamicArrayScene::allocateModeEvents(int newSize) {
    this->array->resetEvents();

    // init highlighter
    // ...

    int size = this->array->getSize(),
        oldSize = this->array->getSquaresSize(),
        squaresSize = std::max(oldSize, newSize);

    std::vector<EventAnimation> events;
    EventAnimation event;

    event.eventSquares.assign(squaresSize, EventSquare());
    event.eventSquaresTemp.assign(newSize, EventSquare());
    for (int i = 0; i < size; ++i) {
        event.eventSquares[i].status = Square::Status::active;
        if (i == size - 1)
            event.eventSquares[i].title = "n";
    }
    for (int i = size; i < oldSize; ++i) {
        event.eventSquares[i].status = Square::Status::inactive;
    }
    for (int i = oldSize; i < newSize; ++i) {
        event.eventSquares[i].status = Square::Status::hidden;
    }
    for (auto &square : event.eventSquaresTemp) {
        square.status = Square::Status::inactive;
        square.isPrintPreVal = true;
    }

    events.emplace_back(event);

    for (int i = 0; i < std::min(size, newSize); ++i) {
        event = EventAnimation();
        event.eventSquares.assign(squaresSize, EventSquare());
        event.eventSquaresTemp.assign(newSize, EventSquare());
        for (int j = 0; j < size; ++j) {
            event.eventSquares[j].status = Square::Status::active;
            if (j == size - 1)
                event.eventSquares[j].title = "n";
        }
        for (int j = size; j < oldSize; ++j) {
            event.eventSquares[j].status = Square::Status::inactive;
        }
        for (int j = oldSize; j < newSize; ++j) {
            event.eventSquares[j].status = Square::Status::hidden;
        }
        for (auto &square : event.eventSquaresTemp) {
            square.status = Square::Status::inactive;
            square.isPrintPreVal = true;
        }
        for (int j = 0; j < i; ++j) {
            event.eventSquaresTemp[j].status = Square::Status::active;
            event.eventSquaresTemp[j].isPrintPreVal = false;
        }
        event.eventSquaresTemp[i].title = "m";
        event.eventSquaresTemp[i].status = Square::Status::chosen;

        events.emplace_back(event);

        event.eventSquaresTemp[i].isPrintPreVal = false;
        event.eventSquares[i].status = Square::Status::chosen;

        events.emplace_back(event);
    }

    event = EventAnimation();
    event.eventSquares.assign(squaresSize, EventSquare());
    event.eventSquaresTemp.assign(newSize, EventSquare());

    for (int i = 0; i < std::min(size, newSize); ++i) {
        event.eventSquares[i].status = Square::Status::active;
        if (i == std::min(size, newSize) - 1)
            event.eventSquares[i].title = "n";
    }
    for (int i = size; i < newSize; ++i) {
        event.eventSquares[i].status = Square::Status::inactive;
    }
    for (int i = newSize; i < oldSize; ++i) {
        event.eventSquares[i].status = Square::Status::hidden;
    }
    for (auto &square : event.eventSquaresTemp) {
        square.status = Square::Status::hidden;
    }

    events.emplace_back(event);

    return events;
}
```

8.69 include/libScene/DynamicArrayScene.hpp File Reference

```
#include "BaseScene.hpp"
#include "MenuArray.hpp"
#include "core/Array.hpp"
```

Classes

- class DynamicArrayScene

8.70 DynamicArrayScene.hpp

Go to the documentation of this file.

```
//
// Created by dirii on 27/03/2023.
//

#ifndef VISUALGO_CS162_DYNAMICARRAYSCENE_HPP
#define VISUALGO_CS162_DYNAMICARRAYSCENE_HPP

#include "BaseScene.hpp"
#include "MenuArray.hpp"
#include "core/Array.hpp"

class DynamicArrayScene : public BaseScene{
private:
    MenuArray* menu;
    Array* array;

    void init();

public:
    explicit DynamicArrayScene(sf::RenderWindow* window);

    void reset();

    void pollEvent(sf::Event event, sf::Vector2f mousePosView) override;
    void update() override;
    void render() override;

    std::vector<EventAnimation> addModeEvents(int chosenNode);
    std::vector<EventAnimation> deleteModeEvents(int chosenNode);
    std::vector<EventAnimation> updateModeEvents(int chosenNode);
    std::vector<EventAnimation> searchModeEvents(int chosenNode);
    std::vector<EventAnimation> allocateModeEvents(int newSize);
};

#endif //VISUALGO_CS162_DYNAMICARRAYSCENE_HPP
```

8.71 include/libScene/Highlighter.cpp File Reference

```
#include "Highlighter.hpp"
```

8.72 Highlighter.cpp

Go to the documentation of this file.

```
//
// Created by dirii on 15/04/2023.
//

#include "Highlighter.hpp"

Highlighter::Highlighter(sf::RenderWindow *window, int linesCount, const char *codePath) {
    this->window = window;
    this->linesCount = linesCount;

    this->codeTexture.loadFromFile(codePath);
    this->codeSprite.setTexture(this->codeTexture);
    this->codeSprite.setScale(constants::Highlighter::codeScale);

    this->codeSprite.setOrigin(
            this->codeSprite.getLocalBounds().width,
            this->codeSprite.getLocalBounds().height
            );

    this->codeSprite.setPosition(constants::Highlighter::codePos);

    float heightTop = 43;

    this->rectSize = sf::Vector2f(
            this->codeSprite.getGlobalBounds().width,
            ((this->codeSprite.getLocalBounds().height - heightTop * 2) /
static_cast<float>(this->linesCount)) * constants::Highlighter::codeScale.y
            );

    for (int i = 0; i < this->linesCount; ++i) {
        sf::RectangleShape rect(this->rectSize);
        rect.setOrigin(rect.getLocalBounds().width, rect.getLocalBounds().height);
        rect.setFillColor(constants::transparentGreen);
        rect.setPosition(
                this->codeSprite.getPosition().x,
                this->codeSprite.getPosition().y - (heightTop * constants::Highlighter::codeScale.y) -
static_cast<float>(this->linesCount - 1 - i) * this->rectSize.y
                );
        this->lines.push_back(rect);
    }
}

void Highlighter::toggle(std::vector<int> linesList) {
    this->toggleLines = std::move(linesList);
}

void Highlighter::render() {
    this->window->draw(this->codeSprite);

    for (auto &i : this->toggleLines) {
        this->window->draw(this->lines[i]);
    }
}

void Highlighter::resetToggle() {
    this->toggleLines.clear();
}
```

8.73 include/libScene/Highlighter.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include "Constants.hpp"
```

Classes

- class Highlighter

8.74 Highlighter.hpp

Go to the documentation of this file.

```
//
// Created by dirii on 15/04/2023.
//

#ifndef VISUALGO_CS162_HIGHLIGHTER_HPP
#define VISUALGO_CS162_HIGHLIGHTER_HPP

#include <SFML/Graphics.hpp>
#include "Constants.hpp"

class Highlighter {
private:
    sf::RenderWindow* window;

    sf::Texture codeTexture;
    sf::Sprite codeSprite;

    int linesCount;

    std::vector<sf::RectangleShape> lines;
    std::vector<int> toggleLines;

    sf::Vector2f rectSize;

public:
    Highlighter(sf::RenderWindow* window, int linesCount, const char* codePath);

    void toggle(std::vector<int> lines);
    void resetToggle();
    void render();
};

#endif //VISUALGO_CS162_HIGHLIGHTER_HPP
```

8.75 include/libScene/MainMenu.cpp File Reference

```
#include "MainMenu.hpp"
```

8.76 MainMenu.cpp

Go to the documentation of this file.

```
//
// Created by dirii on 30/03/2023.
//

#include "MainMenu.hpp"

MainMenu::MainMenu(sf::RenderWindow *window) : BaseScene(window) {
    this->modeButton = new Button;
}

void MainMenu::pollEvent(sf::Event event, sf::Vector2f mousePosView) {

}

void MainMenu::update() {

}

void MainMenu::render() {

}
```

8.77 include/libScene/MainMenu.hpp File Reference

```
#include "BaseScene.hpp"
```

Classes

- class MainMenu

8.78 MainMenu.hpp

Go to the documentation of this file.

```
//
// Created by dirii on 30/03/2023.
//

#ifndef VISUALGO_CS162_MAINMENU_HPP
#define VISUALGO_CS162_MAINMENU_HPP

#include "BaseScene.hpp"

class MainMenu : public BaseScene{
public:
    explicit MainMenu(sf::RenderWindow* window);

    void pollEvent(sf::Event event, sf::Vector2f mousePosView) override;
    void update() override;
    void render() override;
};

#endif //VISUALGO_CS162_MAINMENU_HPP
```

8.79 include/libScene/MenuArray.cpp File Reference

```
#include "MenuArray.hpp"
```

8.80 MenuArray.cpp

Go to the documentation of this file.

```
//
// Created by dirii on 01/05/2023.
//

#include "MenuArray.hpp"

MenuArray::MenuArray(sf::RenderWindow *window, constants::MenuArray::Type _typeArray) {
    this->window = window;
    this->typeArray = _typeArray;
    this->init();
}

void MenuArray::init() {
    this->initButtons();
    this->initCreateMode();
    this->initAddMode();
    this->initDeleteMode();
    this->initUpdateMode();
    this->initSearchMode();
    this->initAllocateMode();

    this->activeOptionMenu = constants::MenuArray::Button::NONE;
}

void MenuArray::initButtons() {
    for (int i = 0; i < constants::MenuArray::BUTTON_COUNT; i++) {
        sf::Vector2f position = sf::Vector2f(
                constants::sideButtonSize.x + constants::distance2ModeButtons,
                constants::submenuButtonPos.y + (constants::optionButtonSize.y +
constants::distance2ModeButtons / 10) * static_cast<float>(i)
        );
        if (i == constants::MenuArray::BUTTON_COUNT - 1 && this->typeArray ==
constants::MenuArray::Type::STATIC)
```

```
            position = sf::Vector2f(-100, -100);
        this->buttons[i] = new Button(
                this->window,
                position,
                constants::optionButtonSize,
                constants::MenuArray::BUTTON_NAMES[i],
                constants::MenuArray::BUTTON_NAMES[i],
                constants::MenuArray::BUTTON_NAME_SIZE,
                sf::Color::Black,
                constants::normalGray,
                constants::hoverGray,
                constants::clickGray
        );
    }
}

void MenuArray::resetActiveOptionMenu() {
    this->activeOptionMenu = constants::MenuArray::Button::NONE;
    this->activeCreateMode = constants::MenuArray::CreateMode::Button::NONE;
}

void MenuArray::pollEvents(sf::Event event, sf::Vector2f mousePosView) {
    if (this->activeOptionMenu != constants::MenuArray::Button::NONE)
        this->buttons[this->activeOptionMenu]->setColor(constants::normalGray);

    for (int i = 0; i < constants::MenuArray::BUTTON_COUNT; ++i) {
        if (this->buttons[i]->pollEvent(mousePosView)) {
            std::cout « "Button " « i « " is clicked" « std::endl;
            this->activeOptionMenu = static_cast<constants::MenuArray::Button>(i);
            this->activeAddMode = constants::MenuArray::AddMode::Textbox::NONE;
        }
    }

    switch (this->activeOptionMenu) {
        case constants::MenuArray::Button::CREATE_BUTTON:
            this->pollEventCreateMode(event, mousePosView);
            break;
        case constants::MenuArray::Button::ADD_BUTTON:
            this->pollEventAddMode(event, mousePosView);
            break;
        case constants::MenuArray::Button::DELETE_BUTTON:
            this->pollEventDeleteMode(event, mousePosView);
            break;
        case constants::MenuArray::Button::UPDATE_BUTTON:
            this->pollEventUpdateMode(event, mousePosView);
            break;
        case constants::MenuArray::Button::SEARCH_BUTTON:
            this->pollEventSearchMode(event, mousePosView);
            break;
        case constants::MenuArray::Button::ALLOCATE_BUTTON:
            this->pollEventAllocateMode(event, mousePosView);
            break;
        case constants::MenuArray::Button::NONE:
            break;
    }
}

void MenuArray::update() {
    if (this->activeOptionMenu != constants::MenuArray::Button::NONE)
        this->buttons[this->activeOptionMenu]->setColor(constants::clickGreen);

    for (Button* button : this->buttons) {
        button->update();
    }

    switch (this->activeOptionMenu) {
        case constants::MenuArray::Button::CREATE_BUTTON:
            this->updateCreateMode();
            break;
        case constants::MenuArray::Button::ADD_BUTTON:
            this->updateAddMode();
            break;
        case constants::MenuArray::Button::DELETE_BUTTON:
            this->updateDeleteMode();
            break;
        case constants::MenuArray::Button::UPDATE_BUTTON:
            this->updateUpdateMode();
            break;
        case constants::MenuArray::Button::SEARCH_BUTTON:
            this->updateSearchMode();
            break;
        case constants::MenuArray::Button::ALLOCATE_BUTTON:
            this->updateAllocateMode();
            break;
        case constants::MenuArray::Button::NONE:
            break;
    }
```

```
}

void MenuArray::render() {
    for (Button* button : this->buttons) {
        button->render();
    }

    switch (this->activeOptionMenu) {
        case constants::MenuArray::Button::CREATE_BUTTON:
            this->renderCreateMode();
            break;
        case constants::MenuArray::Button::ADD_BUTTON:
            this->renderAddMode();
            break;
        case constants::MenuArray::Button::DELETE_BUTTON:
            this->renderDeleteMode();
            break;
        case constants::MenuArray::Button::UPDATE_BUTTON:
            this->renderUpdateMode();
            break;
        case constants::MenuArray::Button::SEARCH_BUTTON:
            this->renderSearchMode();
            break;
        case constants::MenuArray::Button::ALLOCATE_BUTTON:
            this->renderAllocateMode();
            break;
        case constants::MenuArray::Button::NONE:
            break;
    }
}

Button *MenuArray::getButton(int index) {
    return this->buttons[index];
}

constants::MenuArray::Button MenuArray::getActiveOptionMenu() {
    return this->activeOptionMenu;
}

constants::MenuArray::CreateMode::Button MenuArray::getActiveCreateMode() {
    return this->activeCreateMode;
}

void MenuArray::initCreateMode() {
    // init stuff for create mode
    this->activeCreateMode = constants::MenuArray::CreateMode::Button::NONE;
    for (int i = 0; i < constants::MenuArray::CreateMode::BUTTON_COUNT; i++) {
        sf::Vector2f position = sf::Vector2f(
                this->buttons[0]->getPosition().x + (constants::optionButtonSize.x + constants::distance2ModeButtons) * static_cast<float>(i + 1),
                this->buttons[0]->getPosition().y
        );
        this->subCreateMode[i] = new Button(
                this->window,
                position,
                constants::optionButtonSize,
                constants::MenuArray::CreateMode::BUTTON_NAMES[i],
                constants::MenuArray::CreateMode::BUTTON_NAMES[i],
                constants::MenuArray::CreateMode::NAME_SIZE,
                sf::Color::Black,
                constants::normalGray,
                constants::hoverGray,
                constants::clickGray
        );
        if (i < 2)
            this->createTextbox[i] = new CustomTextbox{
                    this->window,
                    sf::Vector2f(
                            this->subCreateMode[0]->getPosition().x,
                            this->subCreateMode[0]->getPosition().y + constants::optionButtonSize.y + constants::distance2ModeButtons
                    ),
                    20,
                    constants::MenuArray::CreateMode::TEXTBOX_NAMES[i],
                    constants::MenuArray::CreateMode::TEXTBOX_LENGTH[i],
            };
        this->createModeValue[i] = "None";
    }
    this->isOpenFileDialog = false;
}
void MenuArray::pollEventCreateMode(sf::Event event, sf::Vector2f mousePosView) {
    if (this->activeCreateMode != constants::MenuArray::CreateMode::Button::NONE)
        this->subCreateMode[this->activeCreateMode]->setColor(constants::normalGray);

    for (int i = 0; i < constants::MenuArray::CreateMode::BUTTON_COUNT; i++) {
        if (this->subCreateMode[i]->pollEvent(mousePosView)) {
            this->activeCreateMode = static_cast<constants::MenuArray::CreateMode::Button>(i);
            if (i == constants::MenuArray::CreateMode::Button::FILE_BUTTON)
                this->isOpenFileDialog = true;
            std::cout « "Button " « i « " is clicked" « std::endl;
        }
    }

//    this->testTextbox->pollEvent(event);
    if (this->activeCreateMode < constants::MenuArray::CreateMode::TEXTBOX_COUNT)
        this->createTextbox[this->activeCreateMode]->pollEvent(event, mousePosView);
}
void MenuArray::updateCreateMode() {
    if (this->activeCreateMode != constants::MenuArray::CreateMode::Button::NONE)
        this->subCreateMode[this->activeCreateMode]->setColor(constants::clickGreen);

    for (Button* button : this->subCreateMode) {
        button->update();
    }

//    this->testTextbox->update();
    if (this->activeCreateMode < constants::MenuArray::CreateMode::TEXTBOX_COUNT) {
        this->createTextbox[this->activeCreateMode]->update();
        std::string inputUser = this->createTextbox[this->activeCreateMode]->getTextString();
        if (inputUser != "None") {
            std::cout « inputUser « std::endl;
            this->createTextbox[this->activeCreateMode]->resetInput();
        }
        this->createModeValue[this->activeCreateMode] = inputUser;
    } else if (this->activeCreateMode == constants::MenuArray::CreateMode::FILE_BUTTON) {
        if (this->isOpenFileDialog) {
            auto f = pfd::open_file("Choose files to read", pfd::path::home(),
                                    {"Text Files (.txt .text)", "*.txt *.text",
                                     "All Files", "*"});

            // wait for the user to select a file unless the window will be not responsive
            while (!f.ready(100)) {
                sf::Event event{};
                this->window->pollEvent(event);
            }

            if (!f.result().empty()) {
                std::ifstream file(f.result()[0]);
                std::string line;
                file » line;
                this->createModeValue[this->activeCreateMode] = line;
            }
        }
        this->isOpenFileDialog = false;
    }
}
void MenuArray::renderCreateMode() {
    for (Button* button : this->subCreateMode) {
        button->render();
    }

//    this->testTextbox->render();
    if (this->activeCreateMode < constants::MenuArray::CreateMode::TEXTBOX_COUNT)
        this->createTextbox[this->activeCreateMode]->render();
}

void MenuArray::initAddMode() {
    //init stuff for add mode
    this->activeAddMode = constants::MenuArray::AddMode::Textbox::NONE;
    for (int i = 0; i < constants::MenuArray::AddMode::TEXTBOX_COUNT; i++) {
        sf::Vector2f position = sf::Vector2f(
                this->buttons[1]->getPosition().x + (constants::optionButtonSize.x +
constants::distance2ModeButtons),
                this->buttons[1]->getPosition().y
        );
        this->addTextbox[i] = new CustomTextbox{
                this->window,
                position,
                20,
                constants::MenuArray::AddMode::TEXTBOX_NAMES[i],
                constants::MenuArray::AddMode::TEXTBOX_LENGTH[i],
        };
        this->addModeValue[i] = "None";
    }
}
void MenuArray::pollEventAddMode(sf::Event event, sf::Vector2f mousePosView) {
    if (this->activeAddMode == constants::MenuArray::AddMode::NONE)
        this->activeAddMode = constants::MenuArray::AddMode::POSITION_TEXTBOX;

    this->addTextbox[this->activeAddMode]->pollEvent(event, mousePosView);
}
void MenuArray::updateAddMode() {
    if (this->activeAddMode == constants::MenuArray::AddMode::NONE)
        this->activeAddMode = constants::MenuArray::AddMode::POSITION_TEXTBOX;

    this->addTextbox[this->activeAddMode]->update();

    std::string inputUser = this->addTextbox[this->activeAddMode]->getTextString();
    // check if input is number
    bool isValid = true;
    for (char i : inputUser)
        if (!std::isdigit(i))
            isValid = false;
    if (isValid && inputUser != "None") {
        this->addModeValue[this->activeAddMode] = inputUser;
        std::cout « inputUser « std::endl;
        this->addTextbox[this->activeAddMode]->resetInput();
        this->activeAddMode =
static_cast<constants::MenuArray::AddMode::Textbox>(!this->activeAddMode);
    }
}
void MenuArray::renderAddMode() {
    this->addTextbox[this->activeAddMode]->render();
}

void MenuArray::initDeleteMode() {
    sf::Vector2f position = sf::Vector2f(
            this->buttons[2]->getPosition().x + (constants::optionButtonSize.x +
constants::distance2ModeButtons),
            this->buttons[2]->getPosition().y
    );
    this->deleteTextbox = new CustomTextbox{
            this->window,
            position,
            20,
            constants::MenuArray::DeleteMode::TEXTBOX_NAME,
            constants::MenuArray::DeleteMode::TEXTBOX_LENGTH,
    };
    this->deleteModeValue = "None";
}
void MenuArray::pollEventDeleteMode(sf::Event event, sf::Vector2f mousePosView) {
    this->deleteTextbox->pollEvent(event, mousePosView);
}
void MenuArray::updateDeleteMode() {
    this->deleteTextbox->update();

    std::string inputUser = this->deleteTextbox->getTextString();
    // check if input is number
    bool isValid = true;
    for (char i : inputUser)
        if (!std::isdigit(i))
            isValid = false;
    if (isValid && inputUser != "None") {
        this->deleteModeValue = inputUser;
        std::cout « inputUser « std::endl;
        this->deleteTextbox->resetInput();
    }
}
void MenuArray::renderDeleteMode() {
    this->deleteTextbox->render();
}

void MenuArray::initUpdateMode() {
    // init stuff for update mode
    this->activeUpdateMode = constants::MenuArray::UpdateMode::Textbox::NONE;
    for (int i = 0; i < constants::MenuArray::UpdateMode::TEXTBOX_COUNT; i++) {
        sf::Vector2f position = sf::Vector2f(
                this->buttons[3]->getPosition().x + (constants::optionButtonSize.x +
constants::distance2ModeButtons),
                this->buttons[3]->getPosition().y
        );
        this->updateTextbox[i] = new CustomTextbox{
                this->window,
                position,
                20,
                constants::MenuArray::UpdateMode::TEXTBOX_NAMES[i],
                constants::MenuArray::UpdateMode::TEXTBOX_LENGTH[i],
        };
        this->updateModeValue[i] = "None";
    }
}
void MenuArray::pollEventUpdateMode(sf::Event event, sf::Vector2f mousePosView) {
    if (this->activeUpdateMode == constants::MenuArray::UpdateMode::NONE)
        this->activeUpdateMode = constants::MenuArray::UpdateMode::POSITION_TEXTBOX;

    this->updateTextbox[this->activeUpdateMode]->pollEvent(event, mousePosView);
}
void MenuArray::updateUpdateMode() {
    if (this->activeUpdateMode == constants::MenuArray::UpdateMode::NONE)
        this->activeUpdateMode = constants::MenuArray::UpdateMode::POSITION_TEXTBOX;

    this->updateTextbox[this->activeUpdateMode]->update();

    std::string inputUser = this->updateTextbox[this->activeUpdateMode]->getTextString();
    // check if input is number
    bool isValid = true;
    for (char i : inputUser)
        if (!std::isdigit(i))
            isValid = false;
    if (isValid && inputUser != "None") {
        this->updateModeValue[this->activeUpdateMode] = inputUser;
        std::cout « inputUser « std::endl;
        this->updateTextbox[this->activeUpdateMode]->resetInput();
        this->activeUpdateMode = static_cast<constants::MenuArray::UpdateMode::Textbox>(!this->activeUpdateMode);
    }
}
void MenuArray::renderUpdateMode() {
    this->updateTextbox[this->activeUpdateMode]->render();
}

void MenuArray::initSearchMode() {
    sf::Vector2f position = sf::Vector2f(
            this->buttons[4]->getPosition().x + (constants::optionButtonSize.x + constants::distance2ModeButtons),
            this->buttons[4]->getPosition().y
    );
    this->searchTextbox = new CustomTextbox{
            this->window,
            position,
            20,
            constants::MenuArray::SearchMode::TEXTBOX_NAME,
            constants::MenuArray::SearchMode::TEXTBOX_LENGTH,
    };
    this->searchModeValue = "None";
}
void MenuArray::pollEventSearchMode(sf::Event event, sf::Vector2f mousePosView) {
    this->searchTextbox->pollEvent(event, mousePosView);
}
void MenuArray::updateSearchMode() {
    this->searchTextbox->update();

    std::string inputUser = this->searchTextbox->getTextString();
    // check if input is number
    bool isValid = true;
    for (char i : inputUser)
        if (!std::isdigit(i))
            isValid = false;
    if (isValid && inputUser != "None") {
        this->searchModeValue = inputUser;
        std::cout « inputUser « std::endl;
        this->searchTextbox->resetInput();
    }
}
void MenuArray::renderSearchMode() {
    this->searchTextbox->render();
}

void MenuArray::initAllocateMode() {
    sf::Vector2f position = sf::Vector2f(
            this->buttons[5]->getPosition().x + (constants::optionButtonSize.x + constants::distance2ModeButtons),
            this->buttons[5]->getPosition().y
    );
    this->allocateTextbox = new CustomTextbox{
            this->window,
            position,
            20,
            constants::MenuArray::AllocateMode::TEXTBOX_NAME,
            constants::MenuArray::AllocateMode::TEXTBOX_LENGTH,
    };
    this->allocateModeValue = "None";
}
void MenuArray::pollEventAllocateMode(sf::Event event, sf::Vector2f mousePosView) {
    this->allocateTextbox->pollEvent(event, mousePosView);
}
void MenuArray::updateAllocateMode() {
    this->allocateTextbox->update();

    std::string inputUser = this->allocateTextbox->getTextString();
    // check if input is number
    bool isValid = true;
    for (char i : inputUser)
        if (!std::isdigit(i))
            isValid = false;
    if (isValid && inputUser != "None") {
        this->allocateModeValue = inputUser;
        std::cout « inputUser « std::endl;
        this->allocateTextbox->resetInput();
    }
}
void MenuArray::renderAllocateMode() {
    this->allocateTextbox->render();
}
```

8.81 include/libScene/MenuArray.hpp File Reference

```
#include <fstream>
#include "Constants.hpp"
#include "stuff/button.hpp"
#include "stuff/CustomTextbox.hpp"
#include "core/FileDialog.h"
```

Classes

- class MenuArray

8.82 MenuArray.hpp

Go to the documentation of this file.

```
//
// Created by dirii on 01/05/2023.
//

#ifndef VISUALGO_CS162_MENUARRAY_HPP
#define VISUALGO_CS162_MENUARRAY_HPP

#include <fstream>
#include "Constants.hpp"
#include "stuff/button.hpp"
#include "stuff/CustomTextbox.hpp"
#include "core/FileDialog.h"

class MenuArray {
private:
    sf::RenderWindow* window;
    Button* buttons[constants::MenuArray::BUTTON_COUNT];
    constants::MenuArray::Type typeArray;

    constants::MenuArray::Button activeOptionMenu;

    // stuff for create mode
    Button* subCreateMode[constants::MenuArray::CreateMode::BUTTON_COUNT];
    CustomTextbox* createTextbox[constants::MenuArray::CreateMode::BUTTON_COUNT];
    constants::MenuArray::CreateMode::Button activeCreateMode;
    bool isOpenFileDialog = false;

    void initCreateMode();
    void pollEventCreateMode(sf::Event event, sf::Vector2f mousePosView);
    void updateCreateMode();
    void renderCreateMode();

    // stuff for add mode
    CustomTextbox* addTextbox[constants::MenuArray::AddMode::TEXTBOX_COUNT];
    constants::MenuArray::AddMode::Textbox activeAddMode;

    void initAddMode();
    void pollEventAddMode(sf::Event event, sf::Vector2f mousePosView);
    void updateAddMode();
    void renderAddMode();

    // stuff for delete mode
    CustomTextbox* deleteTextbox;

    void initDeleteMode();
    void pollEventDeleteMode(sf::Event event, sf::Vector2f mousePosView);
    void updateDeleteMode();
    void renderDeleteMode();

    // stuff for update mode
    CustomTextbox* updateTextbox[constants::MenuArray::UpdateMode::TEXTBOX_COUNT];
    constants::MenuArray::UpdateMode::Textbox activeUpdateMode;

    void initUpdateMode();
    void pollEventUpdateMode(sf::Event event, sf::Vector2f mousePosView);
    void updateUpdateMode();
    void renderUpdateMode();

    // stuff for search mode
    CustomTextbox* searchTextbox;

    void initSearchMode();
    void pollEventSearchMode(sf::Event event, sf::Vector2f mousePosView);
    void updateSearchMode();
    void renderSearchMode();

    // stuff for allocate mode
    CustomTextbox* allocateTextbox;

    void initAllocateMode();
    void pollEventAllocateMode(sf::Event event, sf::Vector2f mousePosView);
    void updateAllocateMode();
    void renderAllocateMode();

    void init();
    void initButtons();

public:
    // stuff public for create mode
    std::string createModeValue[constants::MenuArray::CreateMode::BUTTON_COUNT];
    constants::MenuArray::CreateMode::Button getActiveCreateMode();

    // stuff public for add mode
    std::string addModeValue[constants::MenuArray::AddMode::TEXTBOX_COUNT];

    // stuff public for delete mode
    std::string deleteModeValue;

    // stuff public for update mode
    std::string updateModeValue[constants::MenuArray::UpdateMode::TEXTBOX_COUNT];

    // stuff public for search mode
    std::string searchModeValue;

    // stuff public for allocate mode
    std::string allocateModeValue;

    explicit MenuArray(sf::RenderWindow* window, constants::MenuArray::Type _typeArray);
    ~MenuArray() = default;

    void resetActiveOptionMenu();

    void pollEvents(sf::Event event, sf::Vector2f mousePosView);
    void update();
    void render();

    Button* getButton(int index);
    constants::MenuArray::Button getActiveOptionMenu();
};

#endif //VISUALGO_CS162_MENUARRAY_HPP
```

8.83 include/libScene/MenuDataStructure.cpp File Reference

```
#include "MenuDataStructure.hpp"
```

8.84 MenuDataStructure.cpp

Go to the documentation of this file.

```
//
```

```
// Created by dirii on 26/04/2023.
//

#include "MenuDataStructure.hpp"

MenuDataStructure::MenuDataStructure(sf::RenderWindow *window) {
    this->window = window;
    this->init();
}

void MenuDataStructure::init() {
    this->initButtons();
    this->initCreateMode();
    this->initPushMode();

    this->activeOptionMenu = constants::MenuDataStructure::Button::NONE;
}

void MenuDataStructure::initButtons() {
    for (int i = 0; i < constants::MenuDataStructure::BUTTON_COUNT; i++) {
        sf::Vector2f position = sf::Vector2f(
                constants::sideButtonSize.x + constants::distance2ModeButtons,
                constants::submenuButtonPos.y + (constants::optionButtonSize.y +
constants::distance2ModeButtons / 10) * static_cast<float>(i)
        );
        this->buttons[i] = new Button(
                this->window,
                position,
                constants::optionButtonSize,
                constants::MenuDataStructure::BUTTON_NAMES[i],
                constants::MenuDataStructure::BUTTON_NAMES[i],
                constants::MenuDataStructure::BUTTON_NAME_SIZE,
                sf::Color::Black,
                constants::normalGray,
                constants::hoverGray,
                constants::clickGray
        );
    }
}

void MenuDataStructure::pollEvents(sf::Event event, sf::Vector2f mousePosView) {
    if (this->activeOptionMenu != constants::MenuDataStructure::Button::NONE)
        this->buttons[this->activeOptionMenu]->setColor(constants::normalGray);

    for (int i = 0; i < constants::MenuDataStructure::BUTTON_COUNT; i++) {
        if (this->buttons[i]->pollEvent(mousePosView)) {
            std::cout « "Button " « i « " is clicked" « std::endl;
            this->activeOptionMenu = static_cast<constants::MenuDataStructure::Button>(i);
        }
    }

    if (this->activeOptionMenu == constants::MenuDataStructure::Button::CREATE_BUTTON) {
        this->pollEventCreateMode(event, mousePosView);
    } else if (this->activeOptionMenu == constants::MenuDataStructure::Button::PUSH_BUTTON) {
        this->pollEventPushMode(event, mousePosView);
    }
}

void MenuDataStructure::update() {
    if (this->activeOptionMenu < constants::MenuDataStructure::Button::POP_BUTTON)
        this->buttons[this->activeOptionMenu]->setColor(constants::clickGreen);

    for (Button* button : this->buttons) {
        button->update();
    }

    if (this->activeOptionMenu == constants::MenuDataStructure::Button::CREATE_BUTTON) {
        this->updateCreateMode();
    } else if (this->activeOptionMenu == constants::MenuDataStructure::Button::PUSH_BUTTON) {
        this->updatePushMode();
    }
}

void MenuDataStructure::render() {
    for (Button* button : this->buttons) {
        button->render();
    }

    if (this->activeOptionMenu == constants::MenuDataStructure::Button::CREATE_BUTTON) {
        this->renderCreateMode();
    } else if (this->activeOptionMenu == constants::MenuDataStructure::Button::PUSH_BUTTON) {
        this->renderPushMode();
    }
}

Button *MenuDataStructure::getButton(int index) {
    return this->buttons[index];
```

```
}

constants::MenuDataStructure::Button MenuDataStructure::getActiveOptionMenu() {
    return this->activeOptionMenu;
}

constants::MenuDataStructure::CreateMode::Button MenuDataStructure::getActiveCreateMode() {
    return this->activeCreateMode;
}

void MenuDataStructure::initCreateMode() {
// init stuff for create mode
    this->activeCreateMode = constants::MenuDataStructure::CreateMode::Button::NONE;
    for (int i = 0; i < constants::MenuDataStructure::CreateMode::BUTTON_COUNT; i++) {
        sf::Vector2f position = sf::Vector2f(
                this->buttons[0]->getPosition().x + (constants::optionButtonSize.x +
constants::distance2ModeButtons) * static_cast<float>(i + 1),
                this->buttons[0]->getPosition().y
        );
        this->subCreateMode[i] = new Button(
                this->window,
                position,
                constants::optionButtonSize,
                constants::MenuDataStructure::CreateMode::BUTTON_NAMES[i],
                constants::MenuDataStructure::CreateMode::BUTTON_NAMES[i],
                constants::MenuDataStructure::CreateMode::NAME_SIZE,
                sf::Color::Black,
                constants::normalGray,
                constants::hoverGray,
                constants::clickGray
        );
        if (i < 2)
            this->createTextbox[i] = new CustomTextbox{
                    this->window,
                    sf::Vector2f(
                            this->subCreateMode[0]->getPosition().x,
                            this->subCreateMode[0]->getPosition().y + constants::optionButtonSize.y +
constants::distance2ModeButtons
                    ),
                    20,
                    constants::MenuDataStructure::CreateMode::TEXTBOX_NAMES[i],
                    constants::MenuDataStructure::CreateMode::TEXTBOX_LENGTH[i],
            };
        this->createModeValue[i] = "None";
    }
    this->isOpenFileDialog = false;
}

void MenuDataStructure::pollEventCreateMode(sf::Event event, sf::Vector2f mousePosView) {
    if (this->activeCreateMode != constants::MenuDataStructure::CreateMode::Button::NONE)
        this->subCreateMode[this->activeCreateMode]->setColor(constants::normalGray);

    for (int i = 0; i < constants::MenuDataStructure::CreateMode::BUTTON_COUNT; i++) {
        if (this->subCreateMode[i]->pollEvent(mousePosView)) {
            this->activeCreateMode = static_cast<constants::MenuDataStructure::CreateMode::Button>(i);
            if (i == constants::MenuDataStructure::CreateMode::Button::FILE_BUTTON)
                this->isOpenFileDialog = true;
            std::cout « "Button " « i « " is clicked" « std::endl;
        }
    }

    if (this->activeCreateMode < constants::MenuDataStructure::CreateMode::TEXTBOX_COUNT)
        this->createTextbox[this->activeCreateMode]->pollEvent(event, mousePosView);
}

void MenuDataStructure::updateCreateMode() {
    if (this->activeCreateMode != constants::MenuDataStructure::CreateMode::Button::NONE)
        this->subCreateMode[this->activeCreateMode]->setColor(constants::clickGreen);

    for (Button* button : this->subCreateMode) {
        button->update();
    }

//    this->testTextbox->update();
    if (this->activeCreateMode < constants::MenuDataStructure::CreateMode::TEXTBOX_COUNT) {
        this->createTextbox[this->activeCreateMode]->update();
        std::string inputUser = this->createTextbox[this->activeCreateMode]->getTextString();
        if (inputUser != "None") {
            std::cout « inputUser « std::endl;
            this->createTextbox[this->activeCreateMode]->resetInput();
        }
        this->createModeValue[this->activeCreateMode] = inputUser;
    } else if (this->activeCreateMode == constants::MenuDataStructure::CreateMode::FILE_BUTTON) {
        if (this->isOpenFileDialog) {
            auto f = pfd::open_file("Choose files to read", pfd::path::home(),
                                    {"Text Files (.txt .text)", "*.txt *.text",
                                     "All Files", "*"});

            // wait for the user to select a file unless the window will be not responsive
            while (!f.ready(100)) {
                sf::Event event{};
                this->window->pollEvent(event);
            }

            if (!f.result().empty()) {
                std::ifstream file(f.result()[0]);
                std::string line;
                file » line;
                this->createModeValue[this->activeCreateMode] = line;
            }
        }
        this->isOpenFileDialog = false;
    }
}

void MenuDataStructure::renderCreateMode() {
    for (Button* button : this->subCreateMode) {
        button->render();
    }

    if (this->activeCreateMode < constants::MenuDataStructure::CreateMode::TEXTBOX_COUNT)
        this->createTextbox[this->activeCreateMode]->render();
}

void MenuDataStructure::initPushMode() {
    sf::Vector2f position = sf::Vector2f(
            this->buttons[1]->getPosition().x + (constants::optionButtonSize.x +
constants::distance2ModeButtons),
            this->buttons[1]->getPosition().y
    );
    this->pushTextbox = new CustomTextbox{
            this->window,
            position,
            20,
            constants::MenuDataStructure::PushMode::TEXTBOX_NAME,
            constants::MenuDataStructure::PushMode::TEXTBOX_LENGTH,
    };
    this->pushModeValue = "None";
}

void MenuDataStructure::pollEventPushMode(sf::Event event, sf::Vector2f mousePosView) {
    this->pushTextbox->pollEvent(event, mousePosView);
}

void MenuDataStructure::updatePushMode() {
    this->pushTextbox->update();

    std::string inputUser = this->pushTextbox->getTextString();
    // check if input is number
    bool isValid = true;
    for (char i : inputUser)
        if (!std::isdigit(i))
            isValid = false;
    if (isValid && inputUser != "None") {
        this->pushModeValue = inputUser;
        std::cout « inputUser « std::endl;
        this->pushTextbox->resetInput();
    }
}

void MenuDataStructure::renderPushMode() {
    this->pushTextbox->render();
}

void MenuDataStructure::resetActiveOptionMenu() {
    this->activeOptionMenu = constants::MenuDataStructure::Button::NONE;
    this->activeCreateMode = constants::MenuDataStructure::CreateMode::Button::NONE;
}

void MenuDataStructure::resetActiveOptionMenuOnly(){
    this->activeOptionMenu = constants::MenuDataStructure::Button::NONE;
}
```

8.85 include/libScene/MenuDataStructure.hpp File Reference

```
#include <fstream>
#include "Constants.hpp"
```

```
#include "stuff/button.hpp"
#include "stuff/CustomTextbox.hpp"
#include "core/FileDialog.h"
```

Classes

- class MenuDataStructure

8.86 MenuDataStructure.hpp

Go to the documentation of this file.

```
//
// Created by dirii on 26/04/2023.
//

#ifndef VISUALGO_CS162_MENUDATASTRUCTURE_HPP
#define VISUALGO_CS162_MENUDATASTRUCTURE_HPP

#include <fstream>
#include "Constants.hpp"
#include "stuff/button.hpp"
#include "stuff/CustomTextbox.hpp"
#include "core/FileDialog.h"

class MenuDataStructure {
private:
    sf::RenderWindow* window;
    Button* buttons[constants::MenuDataStructure::BUTTON_COUNT];

    constants::MenuDataStructure::Button activeOptionMenu;

    // stuff for create mode
    Button* subCreateMode[constants::MenuDataStructure::CreateMode::BUTTON_COUNT];
    CustomTextbox* createTextbox[constants::MenuDataStructure::CreateMode::BUTTON_COUNT];
    constants::MenuDataStructure::CreateMode::Button activeCreateMode;
    bool isOpenFileDialog = false;

    void initCreateMode();
    void pollEventCreateMode(sf::Event event, sf::Vector2f mousePosView);
    void updateCreateMode();
    void renderCreateMode();

    // stuff for push mode
    CustomTextbox* pushTextbox;

    void initPushMode();
    void pollEventPushMode(sf::Event event, sf::Vector2f mousePosView);
    void updatePushMode();
    void renderPushMode();

    void init();
    void initButtons();

public:
    // stuff public for create mode
    std::string createModeValue[constants::MenuDataStructure::CreateMode::BUTTON_COUNT];
    constants::MenuDataStructure::CreateMode::Button getActiveCreateMode();

    // stuff public for push mode
    std::string pushModeValue;

    explicit MenuDataStructure(sf::RenderWindow* window);
    ~MenuDataStructure() = default;

    void resetActiveOptionMenu();
    void resetActiveOptionMenuOnly();

    void pollEvents(sf::Event event, sf::Vector2f mousePosView);
    void update();
    void render();

    Button* getButton(int index);
    constants::MenuDataStructure::Button getActiveOptionMenu();
};

#endif //VISUALGO_CS162_MENUDATASTRUCTURE_HPP
```

8.87 include/libScene/MenuLinkedList.cpp File Reference

```
#include "MenuLinkedList.hpp"
```

8.88 MenuLinkedList.cpp

Go to the documentation of this file.

```
//
// Created by dirii on 30/03/2023.
//

#include "MenuLinkedList.hpp"

void MenuLinkedList::init() {
    this->initButtons();
    this->initCreateMode();
    this->initAddMode();
    this->initDeleteMode();
    this->initUpdateMode();
    this->initSearchMode();

    this->activeOptionMenu = constants::MenuLinkedList::Button::NONE;
}

void MenuLinkedList::initButtons() {
    for (int i = 0; i < constants::MenuLinkedList::BUTTON_COUNT; i++) {
        sf::Vector2f position = sf::Vector2f(
                constants::sideButtonSize.x + constants::distance2ModeButtons,
                constants::submenuButtonPos.y + (constants::optionButtonSize.y +
constants::distance2ModeButtons / 10) * static_cast<float>(i)
                );
        this->buttons[i] = new Button(
                this->window,
                position,
                constants::optionButtonSize,
                constants::MenuLinkedList::BUTTON_NAMES[i],
                constants::MenuLinkedList::BUTTON_NAMES[i],
                constants::MenuLinkedList::BUTTON_NAME_SIZE,
                sf::Color::Black,
                constants::normalGray,
                constants::hoverGray,
                constants::clickGray
        );
    }
}

MenuLinkedList::MenuLinkedList(sf::RenderWindow *window) {
    this->window = window;
    this->init();
}

void MenuLinkedList::pollEvents(sf::Event event, sf::Vector2f mousePosView) {
    if (this->activeOptionMenu != constants::MenuLinkedList::Button::NONE)
        this->buttons[this->activeOptionMenu]->setColor(constants::normalGray);

    for (int i = 0; i < constants::MenuLinkedList::BUTTON_COUNT; i++) {
        if (this->buttons[i]->pollEvent(mousePosView)) {
            std::cout « "Button " « i « " is clicked" « std::endl;
            this->activeOptionMenu = static_cast<constants::MenuLinkedList::Button>(i);
            this->activeAddMode = constants::MenuLinkedList::AddMode::Textbox::NONE;
        }
    }

    switch (this->activeOptionMenu) {
        case constants::MenuLinkedList::Button::CREATE_BUTTON:
            this->pollEventCreateMode(event, mousePosView);
            break;
        case constants::MenuLinkedList::Button::ADD_BUTTON:
            this->pollEventAddMode(event, mousePosView);
            break;
        case constants::MenuLinkedList::Button::DELETE_BUTTON:
            this->pollEventDeleteMode(event, mousePosView);
            break;
        case constants::MenuLinkedList::Button::UPDATE_BUTTON:
            this->pollEventUpdateMode(event, mousePosView);
            break;
        case constants::MenuLinkedList::Button::SEARCH_BUTTON:
            this->pollEventSearchMode(event, mousePosView);
            break;
        case constants::MenuLinkedList::Button::NONE:
            break;
    }
}

void MenuLinkedList::update() {
    if (this->activeOptionMenu != constants::MenuLinkedList::Button::NONE)
        this->buttons[this->activeOptionMenu]->setColor(constants::clickGreen);

    for (Button* button : this->buttons) {
        button->update();
    }

    switch (this->activeOptionMenu) {
        case constants::MenuLinkedList::Button::CREATE_BUTTON:
            this->updateCreateMode();
            break;
        case constants::MenuLinkedList::Button::ADD_BUTTON:
            this->updateAddMode();
            break;
        case constants::MenuLinkedList::Button::DELETE_BUTTON:
            this->updateDeleteMode();
            break;
        case constants::MenuLinkedList::Button::UPDATE_BUTTON:
            this->updateUpdateMode();
            break;
        case constants::MenuLinkedList::Button::SEARCH_BUTTON:
            this->updateSearchMode();
            break;
        case constants::MenuLinkedList::Button::NONE:
            break;
    }
}

void MenuLinkedList::render() {
    for (Button* button : this->buttons) {
        button->render();
    }

    switch (this->activeOptionMenu) {
        case constants::MenuLinkedList::Button::CREATE_BUTTON:
            this->renderCreateMode();
            break;
        case constants::MenuLinkedList::Button::ADD_BUTTON:
            this->renderAddMode();
            break;
        case constants::MenuLinkedList::Button::DELETE_BUTTON:
            this->renderDeleteMode();
            break;
        case constants::MenuLinkedList::Button::UPDATE_BUTTON:
            this->renderUpdateMode();
            break;
        case constants::MenuLinkedList::Button::SEARCH_BUTTON:
            this->renderSearchMode();
            break;
        case constants::MenuLinkedList::Button::NONE:
            break;
    }
}

Button *MenuLinkedList::getButton(int index) {
    return this->buttons[index];
}

void MenuLinkedList::resetActiveOptionMenu() {
    this->activeOptionMenu = constants::MenuLinkedList::Button::NONE;
    this->activeCreateMode = constants::MenuLinkedList::CreateMode::Button::NONE;
}

void MenuLinkedList::initCreateMode() {
    // init stuff for create mode
    this->activeCreateMode = constants::MenuLinkedList::CreateMode::Button::NONE;
    for (int i = 0; i < constants::MenuLinkedList::CreateMode::BUTTON_COUNT; i++) {
        sf::Vector2f position = sf::Vector2f(
                this->buttons[0]->getPosition().x + (constants::optionButtonSize.x +
constants::distance2ModeButtons) * static_cast<float>(i + 1),
                this->buttons[0]->getPosition().y
        );
        this->subCreateMode[i] = new Button(
                this->window,
                position,
                constants::optionButtonSize,
                constants::MenuLinkedList::CreateMode::BUTTON_NAMES[i],
                constants::MenuLinkedList::CreateMode::BUTTON_NAMES[i],
                constants::MenuLinkedList::CreateMode::NAME_SIZE,
                sf::Color::Black,
                constants::normalGray,
                constants::hoverGray,
                constants::clickGray
        );
        if (i < 2)
            this->createTextbox[i] = new CustomTextbox{
                    this->window,
                    sf::Vector2f(
                            this->subCreateMode[0]->getPosition().x,
                            this->subCreateMode[0]->getPosition().y + constants::optionButtonSize.y + constants::distance2ModeButtons
                    ),
                    20,
                    constants::MenuLinkedList::CreateMode::TEXTBOX_NAMES[i],
                    constants::MenuLinkedList::CreateMode::TEXTBOX_LENGTH[i],
            };
        this->createModeValue[i] = "None";
    }
    this->isOpenFileDialog = false;
}
void MenuLinkedList::pollEventCreateMode(sf::Event event, sf::Vector2f mousePosView) {
    if (this->activeCreateMode != constants::MenuLinkedList::CreateMode::Button::NONE)
        this->subCreateMode[this->activeCreateMode]->setColor(constants::normalGray);

    for (int i = 0; i < constants::MenuLinkedList::CreateMode::BUTTON_COUNT; i++) {
        if (this->subCreateMode[i]->pollEvent(mousePosView)) {
            this->activeCreateMode = static_cast<constants::MenuLinkedList::CreateMode::Button>(i);
            if (i == constants::MenuLinkedList::CreateMode::Button::FILE_BUTTON)
                this->isOpenFileDialog = true;
            std::cout « "Button " « i « " is clicked" « std::endl;
        }
    }

//    this->testTextbox->pollEvent(event);
    if (this->activeCreateMode < constants::MenuLinkedList::CreateMode::TEXTBOX_COUNT)
        this->createTextbox[this->activeCreateMode]->pollEvent(event, mousePosView);
}
void MenuLinkedList::updateCreateMode() {
    if (this->activeCreateMode != constants::MenuLinkedList::CreateMode::Button::NONE)
        this->subCreateMode[this->activeCreateMode]->setColor(constants::clickGreen);

    for (Button* button : this->subCreateMode) {
        button->update();
    }

//    this->testTextbox->update();
    if (this->activeCreateMode < constants::MenuLinkedList::CreateMode::TEXTBOX_COUNT) {
        this->createTextbox[this->activeCreateMode]->update();
        std::string inputUser = this->createTextbox[this->activeCreateMode]->getTextString();
        if (inputUser != "None") {
            std::cout « inputUser « std::endl;
            this->createTextbox[this->activeCreateMode]->resetInput();
        }
        this->createModeValue[this->activeCreateMode] = inputUser;
    } else if (this->activeCreateMode == constants::MenuLinkedList::CreateMode::FILE_BUTTON) {
        if (this->isOpenFileDialog) {
            auto f = pfd::open_file("Choose files to read", pfd::path::home(),
                                    {"Text Files (.txt .text)", "*.txt *.text",
                                     "All Files", "*"});

            // wait for the user to select a file unless the window will be not responsive
            while (!f.ready(100)) {
                sf::Event event{};
                this->window->pollEvent(event);
            }

            if (!f.result().empty()) {
                std::ifstream file(f.result()[0]);
                std::string line;
                file » line;
                this->createModeValue[this->activeCreateMode] = line;
            }
        }
        this->isOpenFileDialog = false;
    }
}
void MenuLinkedList::renderCreateMode() {
    for (Button* button : this->subCreateMode) {
        button->render();
    }

//    this->testTextbox->render();
    if (this->activeCreateMode < constants::MenuLinkedList::CreateMode::TEXTBOX_COUNT)
        this->createTextbox[this->activeCreateMode]->render();
}

```

```
constants::MenuLinkedList::CreateMode::Button MenuLinkedList::getActiveCreateMode() {
    return this->activeCreateMode;
}

constants::MenuLinkedList::Button MenuLinkedList::getActiveOptionMenu() {
    return this->activeOptionMenu;
}

void MenuLinkedList::initAddMode() {
    //init stuff for add mode
    this->activeAddMode = constants::MenuLinkedList::AddMode::Textbox::NONE;
    for (int i = 0; i < constants::MenuLinkedList::AddMode::TEXTBOX_COUNT; i++) {
        sf::Vector2f position = sf::Vector2f(
                this->buttons[1]->getPosition().x + (constants::optionButtonSize.x + constants::distance2ModeButtons),
                this->buttons[1]->getPosition().y
        );
        this->addTextbox[i] = new CustomTextbox{
                this->window,
                position,
                20,
                constants::MenuLinkedList::AddMode::TEXTBOX_NAMES[i],
                constants::MenuLinkedList::AddMode::TEXTBOX_LENGTH[i],
        };
        this->addModeValue[i] = "None";
    }
}
void MenuLinkedList::pollEventAddMode(sf::Event event, sf::Vector2f mousePosView) {
    if (this->activeAddMode == constants::MenuLinkedList::AddMode::NONE)
        this->activeAddMode = constants::MenuLinkedList::AddMode::POSITION_TEXTBOX;

    this->addTextbox[this->activeAddMode]->pollEvent(event, mousePosView);
}
void MenuLinkedList::updateAddMode() {
    if (this->activeAddMode == constants::MenuLinkedList::AddMode::NONE)
        this->activeAddMode = constants::MenuLinkedList::AddMode::POSITION_TEXTBOX;

    this->addTextbox[this->activeAddMode]->update();

    std::string inputUser = this->addTextbox[this->activeAddMode]->getTextString();
    // check if input is number
    bool isValid = true;
    for (char i : inputUser)
        if (!std::isdigit(i))
            isValid = false;
    if (isValid && inputUser != "None") {
        this->addModeValue[this->activeAddMode] = inputUser;
        std::cout « inputUser « std::endl;
        this->addTextbox[this->activeAddMode]->resetInput();
        this->activeAddMode = static_cast<constants::MenuLinkedList::AddMode::Textbox>(!this->activeAddMode);
    }
}
void MenuLinkedList::renderAddMode() {
    this->addTextbox[this->activeAddMode]->render();
}

void MenuLinkedList::initDeleteMode() {
    sf::Vector2f position = sf::Vector2f(
            this->buttons[2]->getPosition().x + (constants::optionButtonSize.x + constants::distance2ModeButtons),
            this->buttons[2]->getPosition().y
    );
    this->deleteTextbox = new CustomTextbox{
            this->window,
            position,
            20,
            constants::MenuLinkedList::DeleteMode::TEXTBOX_NAME,
            constants::MenuLinkedList::DeleteMode::TEXTBOX_LENGTH,
    };
    this->deleteModeValue = "None";
}
void MenuLinkedList::pollEventDeleteMode(sf::Event event, sf::Vector2f mousePosView) {
    this->deleteTextbox->pollEvent(event, mousePosView);
}
void MenuLinkedList::updateDeleteMode() {
    this->deleteTextbox->update();

    std::string inputUser = this->deleteTextbox->getTextString();
    // check if input is number
    bool isValid = true;
    for (char i : inputUser)
        if (!std::isdigit(i))
            isValid = false;
    if (isValid && inputUser != "None") {
        this->deleteModeValue = inputUser;
        std::cout « inputUser « std::endl;
        this->deleteTextbox->resetInput();
    }
}
void MenuLinkedList::renderDeleteMode() {
    this->deleteTextbox->render();
}

void MenuLinkedList::initUpdateMode() {
    // init stuff for update mode
    this->activeUpdateMode = constants::MenuLinkedList::UpdateMode::Textbox::NONE;
    for (int i = 0; i < constants::MenuLinkedList::UpdateMode::TEXTBOX_COUNT; i++) {
        sf::Vector2f position = sf::Vector2f(
                this->buttons[3]->getPosition().x + (constants::optionButtonSize.x +
constants::distance2ModeButtons),
                this->buttons[3]->getPosition().y
        );
        this->updateTextbox[i] = new CustomTextbox{
                this->window,
                position,
                20,
                constants::MenuLinkedList::UpdateMode::TEXTBOX_NAMES[i],
                constants::MenuLinkedList::UpdateMode::TEXTBOX_LENGTH[i],
        };
        this->updateModeValue[i] = "None";
    }
}
void MenuLinkedList::pollEventUpdateMode(sf::Event event, sf::Vector2f mousePosView) {
    if (this->activeUpdateMode == constants::MenuLinkedList::UpdateMode::NONE)
        this->activeUpdateMode = constants::MenuLinkedList::UpdateMode::POSITION_TEXTBOX;

    this->updateTextbox[this->activeUpdateMode]->pollEvent(event, mousePosView);
}
void MenuLinkedList::updateUpdateMode() {
    if (this->activeUpdateMode == constants::MenuLinkedList::UpdateMode::NONE)
        this->activeUpdateMode = constants::MenuLinkedList::UpdateMode::POSITION_TEXTBOX;

    this->updateTextbox[this->activeUpdateMode]->update();

    std::string inputUser = this->updateTextbox[this->activeUpdateMode]->getTextString();
    // check if input is number
    bool isValid = true;
    for (char i : inputUser)
        if (!std::isdigit(i))
            isValid = false;
    if (isValid && inputUser != "None") {
        this->updateModeValue[this->activeUpdateMode] = inputUser;
        std::cout « inputUser « std::endl;
        this->updateTextbox[this->activeUpdateMode]->resetInput();
        this->activeUpdateMode =
static_cast<constants::MenuLinkedList::UpdateMode::Textbox>(!this->activeUpdateMode);
    }
}
void MenuLinkedList::renderUpdateMode() {
    this->updateTextbox[this->activeUpdateMode]->render();
}

void MenuLinkedList::initSearchMode() {
    sf::Vector2f position = sf::Vector2f(
            this->buttons[4]->getPosition().x + (constants::optionButtonSize.x +
constants::distance2ModeButtons),
            this->buttons[4]->getPosition().y
    );
    this->searchTextbox = new CustomTextbox{
            this->window,
            position,
            20,
            constants::MenuLinkedList::SearchMode::TEXTBOX_NAME,
            constants::MenuLinkedList::SearchMode::TEXTBOX_LENGTH,
    };
    this->searchModeValue = "None";
}
void MenuLinkedList::pollEventSearchMode(sf::Event event, sf::Vector2f mousePosView) {
    this->searchTextbox->pollEvent(event, mousePosView);
}
void MenuLinkedList::updateSearchMode() {
    this->searchTextbox->update();

    std::string inputUser = this->searchTextbox->getTextString();
    // check if input is number
    bool isValid = true;
    for (char i : inputUser)
        if (!std::isdigit(i))
            isValid = false;
    if (isValid && inputUser != "None") {
        this->searchModeValue = inputUser;
        std::cout « inputUser « std::endl;
        this->searchTextbox->resetInput();
    }
}
void MenuLinkedList::renderSearchMode() {
    this->searchTextbox->render();
}
```

8.89 include/libScene/MenuLinkedList.hpp File Reference

```
#include <iostream>
#include <fstream>
#include <SFML/Graphics.hpp>
#include "core/FileDialog.h"
#include "stuff/button.hpp"
#include "stuff/CustomTextbox.hpp"
#include "Constants.hpp"
```

Classes

- class MenuLinkedList

8.90 MenuLinkedList.hpp

Go to the documentation of this file.

```
//
// Created by dirii on 30/03/2023.
//

#ifndef VISUALGO_CS162_MENULINKEDLIST_HPP
#define VISUALGO_CS162_MENULINKEDLIST_HPP

#include <iostream>
#include <fstream>
#include <SFML/Graphics.hpp>
#include "core/FileDialog.h"
#include "stuff/button.hpp"
#include "stuff/CustomTextbox.hpp"
#include "Constants.hpp"

class MenuLinkedList {
protected:
    sf::RenderWindow* window;
    Button* buttons[constants::MenuLinkedList::BUTTON_COUNT];

    constants::MenuLinkedList::Button activeOptionMenu;

    // stuff for create mode
    Button* subCreateMode[constants::MenuLinkedList::CreateMode::BUTTON_COUNT];
    CustomTextbox* createTextbox[constants::MenuLinkedList::CreateMode::BUTTON_COUNT];
    constants::MenuLinkedList::CreateMode::Button activeCreateMode;
    bool isOpenFileDialog = false;

    void initCreateMode();
    void pollEventCreateMode(sf::Event event, sf::Vector2f mousePosView);
    void updateCreateMode();
    void renderCreateMode();

    // stuff for add mode
    CustomTextbox* addTextbox[constants::MenuLinkedList::AddMode::TEXTBOX_COUNT];
    constants::MenuLinkedList::AddMode::Textbox activeAddMode;

    void initAddMode();
    void pollEventAddMode(sf::Event event, sf::Vector2f mousePosView);
    void updateAddMode();
    void renderAddMode();

    // stuff for delete mode
    CustomTextbox* deleteTextbox;

    void initDeleteMode();
    void pollEventDeleteMode(sf::Event event, sf::Vector2f mousePosView);
    void updateDeleteMode();
    void renderDeleteMode();

    // stuff for update mode
    CustomTextbox* updateTextbox[constants::MenuLinkedList::UpdateMode::TEXTBOX_COUNT];
    constants::MenuLinkedList::UpdateMode::Textbox activeUpdateMode;

    void initUpdateMode();
    void pollEventUpdateMode(sf::Event event, sf::Vector2f mousePosView);
    void updateUpdateMode();
    void renderUpdateMode();

    // stuff for search mode
    CustomTextbox* searchTextbox;

    void initSearchMode();
    void pollEventSearchMode(sf::Event event, sf::Vector2f mousePosView);
    void updateSearchMode();
    void renderSearchMode();

    void init();
    void initButtons();

public:
    // stuff public for create mode
    std::string createModeValue[constants::MenuLinkedList::CreateMode::BUTTON_COUNT];
    constants::MenuLinkedList::CreateMode::Button getActiveCreateMode();

    // stuff public for add mode
    std::string addModeValue[constants::MenuLinkedList::AddMode::TEXTBOX_COUNT];

    // stuff public for delete mode
    std::string deleteModeValue;

    // stuff public for update mode
    std::string updateModeValue[constants::MenuLinkedList::UpdateMode::TEXTBOX_COUNT];

    // stuff public for search mode
    std::string searchModeValue;

    explicit MenuLinkedList(sf::RenderWindow* window);
    ~MenuLinkedList() = default;

    void resetActiveOptionMenu();

    void pollEvents(sf::Event event, sf::Vector2f mousePosView);
    void update();
    void render();

    Button* getButton(int index);
    constants::MenuLinkedList::Button getActiveOptionMenu();
};

#endif //VISUALGO_CS162_MENULINKEDLIST_HPP
```

8.91 include/libScene/QueueScene.cpp File Reference

```
#include "QueueScene.hpp"
```

8.92 QueueScene.cpp

Go to the documentation of this file.

```
//
// Created by dirii on 29/03/2023.
//

#include "QueueScene.hpp"

QueueScene::QueueScene(sf::RenderWindow *window) : BaseScene(window) {
    this->init();
}
```

```

void QueueScene::update() {
    if (this->isMenuOpen) {
        this->menu->update();

        constants::MenuDataStructure::Button status = this->menu->getActiveOptionMenu();
        constants::MenuDataStructure::CreateMode::Button createMode;
        switch (status) {
            case constants::MenuDataStructure::Button::CREATE_BUTTON:
                createMode = this->menu->getActiveCreateMode();
                if (createMode == constants::MenuDataStructure::CreateMode::Button::RANDOM_BUTTON) {
                    if (this->menu->createModeValue[0] == "None")
                        break;
                    if (this->menu->createModeValue[0].empty())
                        this->menu->createModeValue[0] = "0";
                    int size = std::stoi(this->menu->createModeValue[0]);
                    this->linkedList->createLinkedList(size);
                } else if (createMode ==
constants::MenuDataStructure::CreateMode::Button::DEFINED_LIST_BUTTON) {
                    if (this->menu->createModeValue[1] == "None")
                        break;
                    std::vector<std::string> values;
                    std::string value = this->menu->createModeValue[1];
                    std::stringstream ss(value);
                    std::string token;
                    while (std::getline(ss, token, ',')) {
                        values.push_back(token);
                    }
                    this->linkedList->createLinkedList(values);
                } else if (createMode ==
constants::MenuDataStructure::CreateMode::Button::FILE_BUTTON) {
                    if (this->menu->createModeValue[2] == "None")
                        break;
                    std::vector<std::string> values;
                    std::string value = this->menu->createModeValue[2];
                    std::stringstream ss(value);
                    std::string token;
                    while (std::getline(ss, token, ','))
                        values.push_back(token);
                    this->linkedList->createLinkedList(values);
                    this->menu->createModeValue[2] = "None";
                }
                this->controlMenu->reset();
                break;
            case constants::MenuDataStructure::Button::PUSH_BUTTON:
                if (this->menu->pushModeValue == "None")
                    break;

                this->linkedList->addNode(
                        this->linkedList->getSize(),
                        this->menu->pushModeValue,
                        this->pushModeEvents(this->linkedList->getSize())
                );

                std::cout « "Pushed " « this->menu->pushModeValue « std::endl;
                this->menu->pushModeValue = "None";
                this->controlMenu->reset();
                break;
            case constants::MenuDataStructure::Button::POP_BUTTON:
                if (this->menu->getActiveOptionMenu() !=
constants::MenuDataStructure::Button::POP_BUTTON)
                    break;

                this->linkedList->deleteNode(
                        0,
                        this->popModeEvents(0)
                );

                std::cout « "Popped " « std::endl;
                this->menu->resetActiveOptionMenuOnly();
                this->controlMenu->reset();
                break;
            case constants::MenuDataStructure::Button::CLEAR_BUTTON:
                if (this->menu->getActiveOptionMenu() !=
constants::MenuDataStructure::Button::CLEAR_BUTTON)
                    break;

                this->linkedList->createLinkedList(0);

                std::cout « "Cleared " « std::endl;
                this->menu->resetActiveOptionMenuOnly();
                this->controlMenu->reset();
                break;
        }
    }

    this->controlMenu->update();

    this->linkedList->processControlMenu(this->controlMenu->getStatus());
    this->linkedList->setSpeed(this->controlMenu->getSpeed());

    this->linkedList->update();
}

void QueueScene::render() {
    if (this->isMenuOpen)
        this->menu->render();

    if (this->isDemoCodeOpen)
        this->linkedList->renderHighlighter();

    this->controlMenu->render();
    this->linkedList->render();
}

void QueueScene::pollEvent(sf::Event event, sf::Vector2f mousePosView) {
    if (this->isMenuOpen)
        this->menu->pollEvents(event, mousePosView);

    this->controlMenu->pollEvents(event, mousePosView);
}

void QueueScene::init() {
    this->menu = new MenuDataStructure(this->window);
    this->linkedList = new LinkedList(this->window, LinkedList::TypeLinkedList::SINGLY);
}

void QueueScene::reset() {
    this->menu->resetActiveOptionMenu();
}

std::vector<EventAnimation> QueueScene::pushModeEvents(int chosenNode) {
    this->linkedList->resetEvents();
    if (chosenNode < 0 || chosenNode > this->linkedList->getSize())
        return {};

    this->linkedList->initHighlighter(
            constants::Highlighter::SLL::CODES_PATH[0].second,
            constants::Highlighter::SLL::CODES_PATH[0].first
    );

    std::vector<EventAnimation> events;
    EventAnimation event;

    if (chosenNode)
        event.titleNodes = {
                {0, "head"},
                {chosenNode, "temp"}
        };
    else {
        event.titleNodes.emplace_back(chosenNode, "temp");
        if (this->linkedList->getSize())
            event.titleNodes.emplace_back(1, "head");
    }
    event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
    if (chosenNode && chosenNode == this->linkedList->getSize())
        event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
    event.colorNodes.push_back(chosenNode);
    event.statusChosenNode = NodeInfo::StatusNode::OutChain;
    event.lines = {0};

    events.emplace_back(event);

    if (chosenNode == 0) {
        if (this->linkedList->getSize()) {
            event.reset();
            event.titleNodes = {
                    {1, "head"},
                    {chosenNode, "temp"}
            };
            event.colorNodes = std::vector<int>{0};
            event.colorArrows.emplace_back(0, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.isPrintNormal = true;
            event.lines = {1, 2};

            events.emplace_back(event);
        }

        event.reset();
        event.titleNodes.emplace_back(chosenNode, "head|temp");
        event.lines = {3};
        event.statusChosenNode = NodeInfo::StatusNode::InChain;
        events.emplace_back(event);
    } else {
        event.reset();
        event.titleNodes = {
                {0, "head|current"},
                {chosenNode, "temp"}
        };
        event.colorNodes.push_back(0);
        event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
        if (chosenNode == this->linkedList->getSize())
            event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
        event.statusChosenNode = NodeInfo::StatusNode::OutChain;
        event.lines = {5};

        events.emplace_back(event);

        for (int i = 0; i < chosenNode; ++i) {
            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {chosenNode, "temp"},
                    {i, "current"}
            };
            event.colorNodes.push_back(i);
            event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
            if (chosenNode == this->linkedList->getSize())
                event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.lines = {6};

            events.emplace_back(event);

            if (i == chosenNode - 1) break;

            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {chosenNode, "temp"},
                    {i, "current"}
            };
            event.colorNodes.push_back(i);
            event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
            event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
            if (chosenNode == this->linkedList->getSize())
                event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.lines = {7};

            events.emplace_back(event);
        }

        if (chosenNode != this->linkedList->getSize()) {
            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {chosenNode, "temp"},
                    {chosenNode - 1, "current"}
            };
            event.colorNodes.push_back(chosenNode);
            event.colorArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.isPrintNormal = true;
            event.lines = {8};

            events.emplace_back(event);
        }

        event.reset();
        event.titleNodes = {
                {0, "head"},
                {chosenNode, "temp"}
        };
        event.statusChosenNode = NodeInfo::StatusNode::InChain;
        event.lines = {9};

        events.emplace_back(event);
    }

    return events;
}

std::vector<EventAnimation> QueueScene::popModeEvents(int chosenNode) {
    this->linkedList->resetEvents();
    if (chosenNode < 0 || chosenNode >= this->linkedList->getSize())
        return {};

    this->linkedList->initHighlighter(
            constants::Highlighter::SLL::CODES_PATH[1].second,
            constants::Highlighter::SLL::CODES_PATH[1].first
    );

    std::vector<EventAnimation> events;
    EventAnimation event;

    if (!chosenNode) {
        event.titleNodes.emplace_back(chosenNode, "head|temp");
        event.colorNodes.push_back(chosenNode);
        event.statusChosenNode = NodeInfo::StatusNode::InChain;
        event.lines = {0, 1};

        events.emplace_back(event);

        if (this->linkedList->getSize() > 1) {
            event.reset();
            event.titleNodes = {
                    {chosenNode, "temp"},
                    {1, "head"}
            };
            event.colorNodes.push_back(1);
            event.colorArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
            event.isPrintNormal = true;
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.lines = {2};

            events.emplace_back(event);
        }

        event.reset();
        event.titleNodes.emplace_back(1, "head");
        event.statusChosenNode = NodeInfo::StatusNode::Visible;
        event.lines = {3};

        events.emplace_back(event);
    } else {
        event.reset();
        event.titleNodes.emplace_back(0, "head|current");
        event.colorNodes.push_back(0);
        event.statusChosenNode = NodeInfo::StatusNode::InChain;
        event.lines = {5};

        events.emplace_back(event);

        for (int i = 0; i < chosenNode; ++i) {
            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {i, "current"}
            };
            event.colorNodes.push_back(i);
            event.statusChosenNode = NodeInfo::StatusNode::InChain;
            event.lines = {6};

            events.emplace_back(event);

            if (i == chosenNode - 1) break;

            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {i, "current"}
            };
            event.colorNodes.push_back(i);
            event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::InChain;
            event.lines = {7};

            events.emplace_back(event);
        }

        event.reset();
        event.titleNodes = {
                {0, "head"},
                {chosenNode, "temp"},
                {chosenNode - 1, "current"}
        };
        event.colorNodes.push_back(chosenNode);
        event.colorArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
        event.statusChosenNode = NodeInfo::StatusNode::InChain;
        event.lines = {8};

        events.emplace_back(event);

        if (chosenNode != this->linkedList->getSize() - 1) {
            event.reset();
            event.titleNodes = {
```

```
                        {0, "head"},
                        {chosenNode, "temp"},
                        {chosenNode - 1, "current"}
                };
                event.colorNodes.push_back(chosenNode);
                event.colorArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
                event.statusChosenNode = NodeInfo::StatusNode::OutChain;
                event.isPrintNormal = true;
                event.lines = {9};

                events.emplace_back(event);

                event.reset();
                event.titleNodes.emplace_back(0, "head");
                event.statusChosenNode = NodeInfo::StatusNode::Visible;
                event.lines = {10};

                events.emplace_back(event);
            } else {
                event.reset();
                event.titleNodes = {
                        {0, "head"},
                        {chosenNode, "temp"},
                        {chosenNode - 1, "current"}
                };
                event.colorNodes.push_back(chosenNode);
                event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
                event.statusChosenNode = NodeInfo::StatusNode::OutChain;
                event.lines = {9};

                events.emplace_back(event);

                event.reset();
                event.titleNodes.emplace_back(0, "head");
                event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
                event.statusChosenNode = NodeInfo::StatusNode::Visible;
                event.lines = {10};

                events.emplace_back(event);
            }
        }

        return events;
}
```

8.93 include/libScene/QueueScene.hpp File Reference

```
#include "BaseScene.hpp"
#include "MenuDataStructure.hpp"
#include "core/LinkedList.hpp"
```

Classes

- class QueueScene

8.94 QueueScene.hpp

Go to the documentation of this file.

```
//
// Created by dirii on 29/03/2023.
//

#ifndef VISUALGO_CS162_QUEUESCENE_HPP
#define VISUALGO_CS162_QUEUESCENE_HPP

#include "BaseScene.hpp"
#include "MenuDataStructure.hpp"
#include "core/LinkedList.hpp"

```

```
class QueueScene : public BaseScene{
private:
    MenuDataStructure* menu;
    LinkedList* linkedList;

    void init();

public:
    explicit QueueScene(sf::RenderWindow* window);

    void reset();

    void pollEvent(sf::Event event, sf::Vector2f mousePosView) override;
    void update() override;
    void render() override;

    std::vector<EventAnimation> pushModeEvents(int chosenNode);
    std::vector<EventAnimation> popModeEvents(int chosenNode);
};

#endif //VISUALGO_CS162_QUEUESCENE_HPP
```

8.95 include/libScene/SLLScene.cpp File Reference

```
#include "SLLScene.hpp"
```

8.96 SLLScene.cpp

Go to the documentation of this file.

```
//
// Created by dirii on 26/03/2023.
//

#include "SLLScene.hpp"

void SLLScene::pollEvent(sf::Event event, sf::Vector2f mousePosView) {
    if (this->isMenuOpen)
        this->menu->pollEvents(event, mousePosView);

    this->controlMenu->pollEvents(event, mousePosView);
}

void SLLScene::update() {
    if (this->isMenuOpen) {
        this->menu->update();

        constants::MenuLinkedList::Button status = this->menu->getActiveOptionMenu();
        constants::MenuLinkedList::CreateMode::Button createMode;
        switch (status){
            case constants::MenuLinkedList::Button::CREATE_BUTTON:
                createMode = this->menu->getActiveCreateMode();
                if (createMode == constants::MenuLinkedList::CreateMode::Button::RANDOM_BUTTON) {
                    if (this->menu->createModeValue[0] == "None")
                        break;
                    if (this->menu->createModeValue[0].empty())
                        this->menu->createModeValue[0] = "0";
                    int size = std::stoi(this->menu->createModeValue[0]);
                    this->linkedList->createLinkedList(size);
                } else if (createMode ==
constants::MenuLinkedList::CreateMode::Button::DEFINED_LIST_BUTTON) {
                    if (this->menu->createModeValue[1] == "None")
                        break;
                    std::vector<std::string> values;
                    std::string value = this->menu->createModeValue[1];
                    std::stringstream ss(value);
                    std::string token;
                    while (std::getline(ss, token, ',')) {
                        values.push_back(token);
                    }
                    this->linkedList->createLinkedList(values);
                } else if (createMode == constants::MenuLinkedList::CreateMode::Button::FILE_BUTTON) {
                    if (this->menu->createModeValue[2] == "None")
                        break;
                    std::vector<std::string> values;
                    std::string value = this->menu->createModeValue[2];
                    std::stringstream ss(value);
                    std::string token;
                    while (std::getline(ss, token, ','))
                        values.push_back(token);
                    this->linkedList->createLinkedList(values);
                    this->menu->createModeValue[2] = "None";
                }
                this->controlMenu->reset();
                break;
            case constants::MenuLinkedList::Button::ADD_BUTTON:
                if (this->menu->addModeValue[0] == "None" || this->menu->addModeValue[1] == "None" ||
this->menu->addModeValue[0].empty())
                    break;

                this->linkedList->addNode(
                        std::stoi(this->menu->addModeValue[0]),
                        this->menu->addModeValue[1],
                        this->addModeEvents(std::stoi(this->menu->addModeValue[0]))
                        );

                std::cout « "Add: " « this->menu->addModeValue[0] « " " « this->menu->addModeValue[1]
« std::endl;
                this->menu->addModeValue[0] = this->menu->addModeValue[1] = "None";
                this->controlMenu->reset();
                break;
            case constants::MenuLinkedList::Button::DELETE_BUTTON:
                if (this->menu->deleteModeValue == "None" || this->menu->deleteModeValue.empty())
                    break;

                this->linkedList->deleteNode(
                        std::stoi(this->menu->deleteModeValue),
                        this->deleteModeEvents(std::stoi(this->menu->deleteModeValue))
                        );

                std::cout « "Delete: " « this->menu->deleteModeValue « std::endl;
                this->menu->deleteModeValue = "None";
                this->controlMenu->reset();
                break;
            case constants::MenuLinkedList::Button::UPDATE_BUTTON:
                if (this->menu->updateModeValue[0] == "None" || this->menu->updateModeValue[1] ==
"None" || this->menu->updateModeValue[0].empty())
                    break;

                this->linkedList->updateNode(
                        std::stoi(this->menu->updateModeValue[0]),
                        this->menu->updateModeValue[1],
                        this->updateModeEvents(std::stoi(this->menu->updateModeValue[0]))
                        );

                std::cout « "Update: " « this->menu->updateModeValue[0] « " " «
this->menu->updateModeValue[1] « std::endl;
                this->menu->updateModeValue[0] = this->menu->updateModeValue[1] = "None";
                this->controlMenu->reset();
                break;
            case constants::MenuLinkedList::Button::SEARCH_BUTTON:
                if (this->menu->searchModeValue == "None" || this->menu->searchModeValue.empty())
                    break;

                this->linkedList->searchNode(

this->searchModeEvents(this->linkedList->findValue(this->menu->searchModeValue))
                        );

                std::cout « "Search: " « this->menu->searchModeValue « std::endl;
                this->menu->searchModeValue = "None";
                this->controlMenu->reset();
                break;
        }
    }

    this->controlMenu->update();

    this->linkedList->processControlMenu(this->controlMenu->getStatus());
    this->linkedList->setSpeed(this->controlMenu->getSpeed());

    this->linkedList->update();
}

void SLLScene::render() {
    if (this->isMenuOpen)
        this->menu->render();

    if (this->isDemoCodeOpen)
        this->linkedList->renderHighlighter();

    this->controlMenu->render();
    this->linkedList->render();
}

SLLScene::SLLScene(sf::RenderWindow *window) : BaseScene(window) {
    this->init();
}

void SLLScene::init() {
    this->menu = new MenuLinkedList(this->window);
    this->linkedList = new LinkedList(this->window, LinkedList::TypeLinkedList::SINGLY);
}

void SLLScene::reset() {
    this->menu->resetActiveOptionMenu();
}

std::vector<EventAnimation> SLLScene::addModeEvents(int chosenNode) {
    this->linkedList->resetEvents();
    if (chosenNode < 0 || chosenNode > this->linkedList->getSize())
        return {};

    this->linkedList->initHighlighter(
            constants::Highlighter::SLL::CODES_PATH[0].second,
            constants::Highlighter::SLL::CODES_PATH[0].first
    );

    std::vector<EventAnimation> events;
    EventAnimation event;

    if (chosenNode)
        event.titleNodes = {
                {0, "head"},
                {chosenNode, "temp"}
        };
    else {
        event.titleNodes.emplace_back(chosenNode, "temp");
        if (this->linkedList->getSize())
            event.titleNodes.emplace_back(1, "head");
    }
    event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
    if (chosenNode && chosenNode == this->linkedList->getSize())
        event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
    event.colorNodes.push_back(chosenNode);
    event.statusChosenNode = NodeInfo::StatusNode::OutChain;
    event.lines = {0};

    events.emplace_back(event);

    if (chosenNode == 0) {
        if (this->linkedList->getSize()) {
            event.reset();
            event.titleNodes = {
                    {1, "head"},
                    {chosenNode, "temp"}
            };
            event.colorNodes = std::vector<int>{0};
            event.colorArrows.emplace_back(0, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.isPrintNormal = true;
            event.lines = {1, 2};

            events.emplace_back(event);
        }

        event.reset();
        event.titleNodes.emplace_back(chosenNode, "head|temp");
        event.lines = {3};
        event.statusChosenNode = NodeInfo::StatusNode::InChain;
        events.emplace_back(event);
    } else {
        event.reset();
        event.titleNodes = {
                {0, "head|current"},
                {chosenNode, "temp"}
        };
        event.colorNodes.push_back(0);
        event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
        if (chosenNode == this->linkedList->getSize())
            event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
        event.statusChosenNode = NodeInfo::StatusNode::OutChain;
        event.lines = {5};

        events.emplace_back(event);

        for (int i = 0; i < chosenNode; ++i) {
            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {chosenNode, "temp"},
                    {i, "current"}
            };
            event.colorNodes.push_back(i);
            event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
            if (chosenNode == this->linkedList->getSize())
                event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.lines = {6};

            events.emplace_back(event);

            if (i == chosenNode - 1) break;

            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {chosenNode, "temp"},
                    {i, "current"}
            };
            event.colorNodes.push_back(i);
            event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
            event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
            if (chosenNode == this->linkedList->getSize())
                event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.lines = {7};

            events.emplace_back(event);
        }

        if (chosenNode != this->linkedList->getSize()) {
            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {chosenNode, "temp"},
                    {chosenNode - 1, "current"}
            };
            event.colorNodes.push_back(chosenNode);
            event.colorArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.isPrintNormal = true;
            event.lines = {8};

            events.emplace_back(event);
        }

        event.reset();
        event.titleNodes = {
                {0, "head"},
                {chosenNode, "temp"}
        };
        event.statusChosenNode = NodeInfo::StatusNode::InChain;
        event.lines = {9};

        events.emplace_back(event);
    }

    return events;
}

std::vector<EventAnimation> SLLScene::deleteModeEvents(int chosenNode) {
    this->linkedList->resetEvents();
    if (chosenNode < 0 || chosenNode >= this->linkedList->getSize())
        return {};

    this->linkedList->initHighlighter(
            constants::Highlighter::SLL::CODES_PATH[1].second,
            constants::Highlighter::SLL::CODES_PATH[1].first
    );

    std::vector<EventAnimation> events;
    EventAnimation event;

    if (!chosenNode) {
        event.titleNodes.emplace_back(chosenNode, "head|temp");
        event.colorNodes.push_back(chosenNode);
        event.statusChosenNode = NodeInfo::StatusNode::InChain;
        event.lines = {0, 1};

        events.emplace_back(event);

        if (this->linkedList->getSize() > 1) {
            event.reset();
            event.titleNodes = {
                    {chosenNode, "temp"},
                    {1, "head"}
            };
            event.colorNodes.push_back(1);
            event.colorArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
            event.isPrintNormal = true;
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.lines = {2};

            events.emplace_back(event);
        }

        event.reset();
        event.titleNodes.emplace_back(1, "head");
        event.statusChosenNode = NodeInfo::StatusNode::Visible;
        event.lines = {3};

        events.emplace_back(event);
    } else {
        event.reset();
        event.titleNodes.emplace_back(0, "head|current");
        event.colorNodes.push_back(0);
        event.statusChosenNode = NodeInfo::StatusNode::InChain;
        event.lines = {5};

        events.emplace_back(event);

        for (int i = 0; i < chosenNode; ++i) {
            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {i, "current"}
            };
            event.colorNodes.push_back(i);
            event.statusChosenNode = NodeInfo::StatusNode::InChain;
            event.lines = {6};

            events.emplace_back(event);

            if (i == chosenNode - 1) break;

            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {i, "current"}
            };
            event.colorNodes.push_back(i);
            event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::InChain;
            event.lines = {7};

            events.emplace_back(event);
        }

        event.reset();
        event.titleNodes = {
                {0, "head"},
                {chosenNode, "temp"},
                {chosenNode - 1, "current"}
        };
        event.colorNodes.push_back(chosenNode);
        event.colorArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
        event.statusChosenNode = NodeInfo::StatusNode::InChain;
        event.lines = {8};

        events.emplace_back(event);

        if (chosenNode != this->linkedList->getSize() - 1) {
            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {chosenNode, "temp"},
                    {chosenNode - 1, "current"}
            };
            event.colorNodes.push_back(chosenNode);
            event.colorArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.isPrintNormal = true;
            event.lines = {9};

            events.emplace_back(event);

            event.reset();
            event.titleNodes.emplace_back(0, "head");
            event.statusChosenNode = NodeInfo::StatusNode::Visible;
            event.lines = {10};

            events.emplace_back(event);
        } else {
            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {chosenNode, "temp"},
                    {chosenNode - 1, "current"}
            };
            event.colorNodes.push_back(chosenNode);
            event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.lines = {9};

            events.emplace_back(event);

            event.reset();
            event.titleNodes.emplace_back(0, "head");
            event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::Visible;
            event.lines = {10};

            events.emplace_back(event);
        }
    }

    return events;
}

std::vector<EventAnimation> SLLScene::updateModeEvents(int chosenNode) {
    this->linkedList->resetEvents();
    if (chosenNode < 0 || chosenNode >= this->linkedList->getSize())
        return {};

    this->linkedList->initHighlighter(
            constants::Highlighter::SLL::CODES_PATH[2].second,
            constants::Highlighter::SLL::CODES_PATH[2].first
    );

    std::vector<EventAnimation> events;
    EventAnimation event;

    event.titleNodes.emplace_back(0, "head|current");
    event.colorNodes.push_back(0);
    event.isPrintPreVal = true;
    event.lines = {0};

    events.emplace_back(event);

    if (chosenNode) {
        for (int i = 0; i <= chosenNode; ++i) {
            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {i, "current"}
            };
            event.colorNodes.push_back(i);
            event.isPrintPreVal = true;
            event.lines = {1};

            events.emplace_back(event);

            if (i == chosenNode) break;

            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {i, "current"}
            };
            event.colorNodes.push_back(i);
            event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
            event.isPrintPreVal = true;
            event.lines = {2};

            events.emplace_back(event);
        }
    }

    event.reset();
    if (chosenNode == 0)
        event.titleNodes.emplace_back(0, "head|current");
    else
        event.titleNodes = {
                {0, "head"},
                {chosenNode, "current"}
        };
    event.lines = {3};

    events.emplace_back(event);
 
    return events;
}

std::vector<EventAnimation> SLLScene::searchModeEvents(int chosenNode) {
    this->linkedList->resetEvents();
    this->linkedList->initHighlighter(
            constants::Highlighter::SLL::CODES_PATH[3].second,
            constants::Highlighter::SLL::CODES_PATH[3].first
    );

    std::vector<EventAnimation> events;
    EventAnimation event;

    event.titleNodes.emplace_back(0, "head|current");
    event.colorNodes.push_back(0);
    event.lines = {0};

    events.emplace_back(event);

    for (int i = 0; i <= chosenNode; ++i) {
        if (i == chosenNode && chosenNode == this->linkedList->getSize())
            break;

        event.reset();
        event.titleNodes = {
                {0, "head"},
                {i, "current"}
        };
        event.colorNodes.push_back(i);
        event.lines = {1};

        events.emplace_back(event);

        if (i == chosenNode) break;

        event.reset();
        event.titleNodes = {
                {0, "head"},
                {i, "current"}
        };
        event.colorNodes.push_back(i);
        event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
        event.lines = {4};

        events.emplace_back(event);
    }

    if (chosenNode == this->linkedList->getSize()) {
        event.reset();
        event.titleNodes.emplace_back(0, "head");
        event.lines = {5};

        events.emplace_back(event);
    } else {
        event.reset();
        event.titleNodes = {
                {0, "head"},
                {chosenNode, "current"}
        };
        event.colorNodes.push_back(chosenNode);
        event.lines = {2, 3};

        events.emplace_back(event);
    }

    return events;
}
```

8.97 include/libScene/SLLScene.hpp File Reference

```
#include "BaseScene.hpp"
#include "MenuLinkedList.hpp"
#include "core/LinkedList.hpp"
```

Classes

- class SLLScene

8.98 SLLScene.hpp

Go to the documentation of this file.

```
//
// Created by dirii on 26/03/2023.
//

#ifndef VISUALGO_CS162_SLLSCENE_HPP
#define VISUALGO_CS162_SLLSCENE_HPP

#include "BaseScene.hpp"
#include "MenuLinkedList.hpp"
#include "core/LinkedList.hpp"

class SLLScene : public BaseScene {
private:
    MenuLinkedList* menu;
    LinkedList* linkedList;

    void init();

public:
    explicit SLLScene(sf::RenderWindow* window);

    void reset();

    void pollEvent(sf::Event event, sf::Vector2f mousePosView) override;
    void update() override;
    void render() override;

    std::vector<EventAnimation> addModeEvents(int chosenNode);
    std::vector<EventAnimation> deleteModeEvents(int chosenNode);
    std::vector<EventAnimation> updateModeEvents(int chosenNode);
    std::vector<EventAnimation> searchModeEvents(int chosenNode);
};

#endif //VISUALGO_CS162_SLLSCENE_HPP
```

8.99 include/libScene/StackScene.cpp File Reference

`#include "StackScene.hpp"`

8.100 StackScene.cpp

Go to the documentation of this file.

```
//
// Created by dirii on 28/03/2023.
//

#include "StackScene.hpp"

StackScene::StackScene(sf::RenderWindow *window) : BaseScene(window) {
    this->init();
}

void StackScene::update() {
    if (this->isMenuOpen) {
        this->menu->update();

        constants::MenuDataStructure::Button status = this->menu->getActiveOptionMenu();
        constants::MenuDataStructure::CreateMode::Button createMode;
        switch (status) {
            case constants::MenuDataStructure::Button::CREATE_BUTTON:
                createMode = this->menu->getActiveCreateMode();
                if (createMode == constants::MenuDataStructure::CreateMode::Button::RANDOM_BUTTON) {
                    if (this->menu->createModeValue[0] == "None")
                        break;
                    if (this->menu->createModeValue[0].empty())
                        this->menu->createModeValue[0] = "0";
                    int size = std::stoi(this->menu->createModeValue[0]);
                    this->linkedList->createLinkedList(size);
                } else if (createMode == constants::MenuDataStructure::CreateMode::Button::DEFINED_LIST_BUTTON) {
                    if (this->menu->createModeValue[1] == "None")
                        break;
                    std::vector<std::string> values;
                    std::string value = this->menu->createModeValue[1];
                    std::stringstream ss(value);
                    std::string token;
                    while (std::getline(ss, token, ',')) {
                        values.push_back(token);
                    }
                    this->linkedList->createLinkedList(values);
                } else if (createMode == constants::MenuDataStructure::CreateMode::Button::FILE_BUTTON) {
                    if (this->menu->createModeValue[2] == "None")
                        break;
                    std::vector<std::string> values;
                    std::string value = this->menu->createModeValue[2];
                    std::stringstream ss(value);
                    std::string token;
                    while (std::getline(ss, token, ','))
                        values.push_back(token);
                    this->linkedList->createLinkedList(values);
                    this->menu->createModeValue[2] = "None";
                }
                this->controlMenu->reset();
                break;
            case constants::MenuDataStructure::Button::PUSH_BUTTON:
                if (this->menu->pushModeValue == "None")
                    break;

                this->linkedList->addNode(
                        0,
                        this->menu->pushModeValue,
                        this->pushModeEvents(0)
                        );

                std::cout « "Pushed " « this->menu->pushModeValue « std::endl;
                this->menu->pushModeValue = "None";
                this->controlMenu->reset();
                break;
            case constants::MenuDataStructure::Button::POP_BUTTON:
                if (this->menu->getActiveOptionMenu() != constants::MenuDataStructure::Button::POP_BUTTON)
                    break;

                this->linkedList->deleteNode(
                        0,
                        this->popModeEvents(0)
                        );

                std::cout « "Popped " « std::endl;
                this->menu->resetActiveOptionMenuOnly();
                this->controlMenu->reset();
                break;
            case constants::MenuDataStructure::Button::CLEAR_BUTTON:
                if (this->menu->getActiveOptionMenu() != constants::MenuDataStructure::Button::CLEAR_BUTTON)
                    break;

                this->linkedList->createLinkedList(0);

                std::cout « "Cleared " « std::endl;
                this->menu->resetActiveOptionMenuOnly();
                this->controlMenu->reset();
                break;
        }
    }

    this->controlMenu->update();

    this->linkedList->processControlMenu(this->controlMenu->getStatus());
    this->linkedList->setSpeed(this->controlMenu->getSpeed());

    this->linkedList->update();
}

void StackScene::render() {
    if (this->isMenuOpen)
        this->menu->render();

    if (this->isDemoCodeOpen)
        this->linkedList->renderHighlighter();

    this->controlMenu->render();
    this->linkedList->render();
}
```

```

void StackScene::pollEvent(sf::Event event, sf::Vector2f mousePosView) {
    if (this->isMenuOpen)
        this->menu->pollEvents(event, mousePosView);

    this->controlMenu->pollEvents(event, mousePosView);
}

void StackScene::init() {
    this->menu = new MenuDataStructure(this->window);
    this->linkedList = new LinkedList(this->window, LinkedList::TypeLinkedList::SINGLY);
}

void StackScene::reset() {
    this->menu->resetActiveOptionMenu();
}

std::vector<EventAnimation> StackScene::pushModeEvents(int chosenNode) {
    this->linkedList->resetEvents();
    if (chosenNode < 0 || chosenNode > this->linkedList->getSize())
        return {};

    this->linkedList->initHighlighter(
            constants::Highlighter::SLL::CODES_PATH[0].second,
            constants::Highlighter::SLL::CODES_PATH[0].first
    );

    std::vector<EventAnimation> events;
    EventAnimation event;

    if (chosenNode)
        event.titleNodes = {
                {0, "head"},
                {chosenNode, "temp"}
        };
    else {
        event.titleNodes.emplace_back(chosenNode, "temp");
        if (this->linkedList->getSize())
            event.titleNodes.emplace_back(1, "head");
    }
    event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
    if (chosenNode && chosenNode == this->linkedList->getSize())
        event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
    event.colorNodes.push_back(chosenNode);
    event.statusChosenNode = NodeInfo::StatusNode::OutChain;
    event.lines = {0};

    events.emplace_back(event);

    if (chosenNode == 0) {
        if (this->linkedList->getSize()) {
            event.reset();
            event.titleNodes = {
                    {1, "head"},
                    {chosenNode, "temp"}
            };
            event.colorNodes = std::vector<int>{0};
            event.colorArrows.emplace_back(0, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.isPrintNormal = true;
            event.lines = {1, 2};

            events.emplace_back(event);
        }

        event.reset();
        event.titleNodes.emplace_back(chosenNode, "head|temp");
        event.lines = {3};
        event.statusChosenNode = NodeInfo::StatusNode::InChain;
        events.emplace_back(event);
    } else {
        event.reset();
        event.titleNodes = {
                {0, "head|current"},
                {chosenNode, "temp"}
        };
        event.colorNodes.push_back(0);
        event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
        if (chosenNode == this->linkedList->getSize())
            event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
        event.statusChosenNode = NodeInfo::StatusNode::OutChain;
        event.lines = {5};

        events.emplace_back(event);

        for (int i = 0; i < chosenNode; ++i) {
            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {chosenNode, "temp"},
                    {i, "current"}
            };
            event.colorNodes.push_back(i);
            event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
            if (chosenNode == this->linkedList->getSize())
                event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.lines = {6};

            events.emplace_back(event);

            if (i == chosenNode - 1) break;

            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {chosenNode, "temp"},
                    {i, "current"}
            };
            event.colorNodes.push_back(i);
            event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
            event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
            if (chosenNode == this->linkedList->getSize())
                event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.lines = {7};

            events.emplace_back(event);
        }

        if (chosenNode != this->linkedList->getSize()) {
            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {chosenNode, "temp"},
                    {chosenNode - 1, "current"}
            };
            event.colorNodes.push_back(chosenNode);
            event.colorArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.isPrintNormal = true;
            event.lines = {8};

            events.emplace_back(event);
        }

        event.reset();
        event.titleNodes = {
                {0, "head"},
                {chosenNode, "temp"}
        };
        event.statusChosenNode = NodeInfo::StatusNode::InChain;
        event.lines = {9};

        events.emplace_back(event);
    }

    return events;
}

std::vector<EventAnimation> StackScene::popModeEvents(int chosenNode) {
    this->linkedList->resetEvents();
    if (chosenNode < 0 || chosenNode >= this->linkedList->getSize())
        return {};

    this->linkedList->initHighlighter(
            constants::Highlighter::SLL::CODES_PATH[1].second,
            constants::Highlighter::SLL::CODES_PATH[1].first
    );

    std::vector<EventAnimation> events;
    EventAnimation event;

    if (!chosenNode) {
        event.titleNodes.emplace_back(chosenNode, "head|temp");
        event.colorNodes.push_back(chosenNode);
        event.statusChosenNode = NodeInfo::StatusNode::InChain;
        event.lines = {0, 1};

        events.emplace_back(event);

        if (this->linkedList->getSize() > 1) {
            event.reset();
            event.titleNodes = {
                    {chosenNode, "temp"},
                    {1, "head"}
            };
            event.colorNodes.push_back(1);
            event.colorArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
            event.isPrintNormal = true;
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.lines = {2};

            events.emplace_back(event);
        }

        event.reset();
        event.titleNodes.emplace_back(1, "head");
        event.statusChosenNode = NodeInfo::StatusNode::Visible;
        event.lines = {3};

        events.emplace_back(event);
    } else {
        event.reset();
        event.titleNodes.emplace_back(0, "head|current");
        event.colorNodes.push_back(0);
        event.statusChosenNode = NodeInfo::StatusNode::InChain;
        event.lines = {5};

        events.emplace_back(event);

        for (int i = 0; i < chosenNode; ++i) {
            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {i, "current"}
            };
            event.colorNodes.push_back(i);
            event.statusChosenNode = NodeInfo::StatusNode::InChain;
            event.lines = {6};

            events.emplace_back(event);

            if (i == chosenNode - 1) break;

            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {i, "current"}
            };
            event.colorNodes.push_back(i);
            event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::InChain;
            event.lines = {7};

            events.emplace_back(event);
        }

        event.reset();
        event.titleNodes = {
                {0, "head"},
                {chosenNode, "temp"},
                {chosenNode - 1, "current"}
        };
        event.colorNodes.push_back(chosenNode);
        event.colorArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
        event.statusChosenNode = NodeInfo::StatusNode::InChain;
        event.lines = {8};

        events.emplace_back(event);

        if (chosenNode != this->linkedList->getSize() - 1) {
            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {chosenNode, "temp"},
                    {chosenNode - 1, "current"}
            };
            event.colorNodes.push_back(chosenNode);
            event.colorArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.isPrintNormal = true;
            event.lines = {9};

            events.emplace_back(event);

            event.reset();
            event.titleNodes.emplace_back(0, "head");
            event.statusChosenNode = NodeInfo::StatusNode::Visible;
            event.lines = {10};

            events.emplace_back(event);
        } else {
            event.reset();
            event.titleNodes = {
                    {0, "head"},
                    {chosenNode, "temp"},
                    {chosenNode - 1, "current"}
            };
            event.colorNodes.push_back(chosenNode);
            event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::OutChain;
            event.lines = {9};

            events.emplace_back(event);

            event.reset();
            event.titleNodes.emplace_back(0, "head");
            event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
            event.statusChosenNode = NodeInfo::StatusNode::Visible;
            event.lines = {10};

            events.emplace_back(event);
        }
    }

    return events;
}
```

8.101 include/libScene/StackScene.hpp File Reference

```
#include "BaseScene.hpp"
#include "MenuDataStructure.hpp"
#include "core/LinkedList.hpp"
```

Classes

- class StackScene

8.102 StackScene.hpp

Go to the documentation of this file.

```
//
// Created by dirii on 28/03/2023.
//

#ifndef VISUALGO_CS162_STACKSCENE_HPP
#define VISUALGO_CS162_STACKSCENE_HPP

#include "BaseScene.hpp"
#include "MenuDataStructure.hpp"
#include "core/LinkedList.hpp"

class StackScene : public BaseScene{
private:
    MenuDataStructure* menu;
    LinkedList* linkedList;

    void init();

public:
    explicit StackScene(sf::RenderWindow* window);

    void reset();

    void pollEvent(sf::Event event, sf::Vector2f mousePosView) override;
    void update() override;
    void render() override;

    std::vector<EventAnimation> pushModeEvents(int chosenNode);
    std::vector<EventAnimation> popModeEvents(int chosenNode);
};

#endif //VISUALGO_CS162_STACKSCENE_HPP
```

8.103 include/libScene/StaticArrayScene.cpp File Reference

```
#include "StaticArrayScene.hpp"
```

8.104 StaticArrayScene.cpp

Go to the documentation of this file.

```
//
// Created by dirii on 27/03/2023.
//

#include "StaticArrayScene.hpp"

StaticArrayScene::StaticArrayScene(sf::RenderWindow *window) : BaseScene(window) {
    this->init();
}

void StaticArrayScene::update() {
    if (this->isMenuOpen) {
        this->menu->update();

        constants::MenuArray::Button status = this->menu->getActiveOptionMenu();
        constants::MenuArray::CreateMode::Button createMode;
        switch (status){
            case constants::MenuArray::Button::CREATE_BUTTON:
                createMode = this->menu->getActiveCreateMode();
                if (createMode == constants::MenuArray::CreateMode::Button::RANDOM_BUTTON) {
                    if (this->menu->createModeValue[0] == "None")
                        break;
                    if (this->menu->createModeValue[0].empty())
                        this->menu->createModeValue[0] = "0";
                    int size = std::stoi(this->menu->createModeValue[0]);
                    this->array->createArray(size);
                } else if (createMode ==
constants::MenuArray::CreateMode::Button::DEFINED_LIST_BUTTON) {
                    if (this->menu->createModeValue[1] == "None")
                        break;
                    std::vector<std::string> values;
                    std::string value = this->menu->createModeValue[1];
                    std::stringstream ss(value);
                    std::string token;
                    while (std::getline(ss, token, ',')) {
                        values.push_back(token);
                    }
                    this->array->createArray(values);
                } else if (createMode == constants::MenuArray::CreateMode::Button::FILE_BUTTON) {
                    if (this->menu->createModeValue[2] == "None")
                        break;
                    std::vector<std::string> values;
                    std::string value = this->menu->createModeValue[2];
                    std::stringstream ss(value);
                    std::string token;
                    while (std::getline(ss, token, ','))
                        values.push_back(token);
                    this->array->createArray(values);
                    this->menu->createModeValue[2] = "None";
                }
                this->controlMenu->reset();
                break;
            case constants::MenuArray::Button::ADD_BUTTON:
                if (this->menu->addModeValue[0] == "None" || this->menu->addModeValue[1] == "None" ||
this->menu->addModeValue[0].empty())
                    break;

                this->array->addSquare(
                        std::stoi(this->menu->addModeValue[0]),
                        this->menu->addModeValue[1],
                        this->addModeEvents(std::stoi(this->menu->addModeValue[0]))
                );

                std::cout « "Add: " « this->menu->addModeValue[0] « " " « this->menu->addModeValue[1]
« std::endl;
                this->menu->addModeValue[0] = this->menu->addModeValue[1] = "None";
                this->controlMenu->reset();
                break;
            case constants::MenuArray::Button::DELETE_BUTTON:
                if (this->menu->deleteModeValue == "None" || this->menu->deleteModeValue.empty())
```

```
                    break;

                this->array->deleteSquare(
                        std::stoi(this->menu->deleteModeValue),
                        this->deleteModeEvents(std::stoi(this->menu->deleteModeValue))
                );

                std::cout « "Delete: " « this->menu->deleteModeValue « std::endl;
                this->menu->deleteModeValue = "None";
                this->controlMenu->reset();
                break;
            case constants::MenuArray::Button::UPDATE_BUTTON:
                if (this->menu->updateModeValue[0] == "None" || this->menu->updateModeValue[1] == "None" || this->menu->updateModeValue[0].empty())
                    break;

                this->array->updateSquare(
                        std::stoi(this->menu->updateModeValue[0]),
                        this->menu->updateModeValue[1],
                        this->updateModeEvents(std::stoi(this->menu->updateModeValue[0]))
                );

                std::cout « "Update: " « this->menu->updateModeValue[0] « " " « this->menu->updateModeValue[1] « std::endl;
                this->menu->updateModeValue[0] = this->menu->updateModeValue[1] = "None";
                this->controlMenu->reset();
                break;
            case constants::MenuArray::Button::SEARCH_BUTTON:
                if (this->menu->searchModeValue == "None" || this->menu->searchModeValue.empty())
                    break;

                this->array->searchSquare(
                        this->searchModeEvents(this->array->findValue(this->menu->searchModeValue))
                );

                std::cout « "Search: " « this->menu->searchModeValue « std::endl;
                this->menu->searchModeValue = "None";
                this->controlMenu->reset();
                break;
        }
    }

    this->controlMenu->update();

    this->array->processControlMenu(this->controlMenu->getStatus());
    this->array->setSpeed(this->controlMenu->getSpeed());

    this->array->update();
}

void StaticArrayScene::render() {
    if (this->isMenuOpen)
        this->menu->render();

    if (this->isDemoCodeOpen)
        this->array->renderHighlighter();

    this->controlMenu->render();
    this->array->render();
}

void StaticArrayScene::pollEvent(sf::Event event, sf::Vector2f mousePosView) {
    if (this->isMenuOpen)
        this->menu->pollEvents(event, mousePosView);

    this->controlMenu->pollEvents(event, mousePosView);
}

void StaticArrayScene::init() {
    this->menu = new MenuArray(this->window, constants::MenuArray::Type::STATIC);
    this->array = new Array(this->window, Array::TypeArray::STATIC);
}

void StaticArrayScene::reset() {
    this->menu->resetActiveOptionMenu();
}

std::vector<EventAnimation> StaticArrayScene::addModeEvents(int chosenNode) {
    this->array->resetEvents();
    if (chosenNode < 0 || chosenNode > this->array->getSize())
        return {};

    // init highlighter
    // ...

    int size = this->array->getSize() + 1,
            squaresSize = this->array->getSquaresSize();
    if (size > squaresSize) --size;
    if (!squaresSize) return {};

    std::vector<EventAnimation> events;
    EventAnimation event;

    if (size < squaresSize) {
        event = EventAnimation();
        event.eventSquares.assign(squaresSize, EventSquare());
        for (auto &square: event.eventSquares) {
            square.status = Square::Status::active;
            square.isPrintPreVal = true;
        }
        for (int i = size - 1; i < squaresSize; ++i)
            event.eventSquares[i].status = Square::Status::inactive;
        if (size > 1)
            event.eventSquares[size - 2].title = "n";

        events.emplace_back(event);

        event = EventAnimation();
        event.eventSquares.assign(squaresSize, EventSquare());
        for (auto &square : event.eventSquares) {
            square.status = Square::Status::active;
            square.isPrintPreVal = true;
        }
        for (int i = size; i < squaresSize; ++i)
            event.eventSquares[i].status = Square::Status::inactive;
        event.eventSquares[size - 1].title = "n";

        events.emplace_back(event);
    }

    for (int i = size - 1; i >= chosenNode; --i) {
        event = EventAnimation();
        event.eventSquares.assign(squaresSize, EventSquare());
        for (auto &square: event.eventSquares) {
            square.status = Square::Status::active;
            square.isPrintPreVal = true;
        }
        for (int j = size; j < squaresSize; ++j)
            event.eventSquares[j].status = Square::Status::inactive;
        event.eventSquares[size - 1].title = "n";
        for (int j = size - 1; j > i; --j)
            event.eventSquares[j].isPrintPreVal = false;
        event.eventSquares[i].status = Square::Status::chosen;

        events.emplace_back(event);

        event.eventSquares[i].isPrintPreVal = false;
        if (i > chosenNode)
            event.eventSquares[i - 1].status = Square::Status::chosen;

        events.emplace_back(event);
    }

    return events;
}

std::vector<EventAnimation> StaticArrayScene::deleteModeEvents(int chosenNode) {
    this->array->resetEvents();
    if (chosenNode < 0 || chosenNode >= this->array->getSize())
        return {};

    // init highlighter
    // ...

    int size = this->array->getSize(),
            squaresSize = this->array->getSquaresSize();
    std::vector<EventAnimation> events;
//    events.reserve(100);
    EventAnimation event;

    for (int i = chosenNode; i < size - 1; ++i) {
        event = EventAnimation();
        event.eventSquares.assign(squaresSize, EventSquare());
        for (auto &square : event.eventSquares) {
            square.status = Square::Status::active;
            square.isPrintPreVal = true;
        }
        for (int j = size; j < squaresSize; ++j)
            event.eventSquares[j].status = Square::Status::inactive;
        for (int j = 0; j < i; ++j)
            event.eventSquares[j].isPrintPreVal = false;
        event.eventSquares[i].status = Square::Status::chosen;
        for (auto &square : event.eventSquaresTemp)
            square.status = Square::Status::hidden;
        event.eventSquares[size - 1].title = "n";

        events.emplace_back(event);

        event.eventSquares[i].isPrintPreVal = false;
        event.eventSquares[i + 1].status = Square::Status::chosen;

        events.emplace_back(event);
    }

    event = EventAnimation();
    event.eventSquares.assign(squaresSize, EventSquare());
    for (int i = 0; i < size - 1; ++i) {
        event.eventSquares[i].status = Square::Status::active;
        if (i == size - 2)
            event.eventSquares[i].title = "n";
    }
    for (int i = size - 1; i < squaresSize; ++i)
        event.eventSquares[i].status = Square::Status::inactive;

    events.emplace_back(event);

    return events;
}

std::vector<EventAnimation> StaticArrayScene::updateModeEvents(int chosenNode) {
    this->array->resetEvents();
    if (chosenNode < 0 || chosenNode >= this->array->getSize())
        return {};

    // init highlighter
    // ...

    std::vector<EventAnimation> events;
    EventAnimation event;

    event = EventAnimation();
    event.eventSquares.assign(this->array->getSquaresSize(), EventSquare());
    for (int i = 0; i < this->array->getSize(); ++i) {
        event.eventSquares[i].status = Square::Status::active;
        if (i == this->array->getSize() - 1)
            event.eventSquares[this->array->getSize() - 1].title = "n";
    }
    event.eventSquares[chosenNode].status = Square::Status::chosen;
    event.eventSquares[chosenNode].isPrintPreVal = true;

    events.emplace_back(event);

    event.eventSquares[chosenNode].isPrintPreVal = false;

    events.emplace_back(event);

    return events;
}

std::vector<EventAnimation> StaticArrayScene::searchModeEvents(int chosenNode) {
    this->array->resetEvents();

    // init highlighter
    // ...

    int size = this->array->getSize(),
            squaresSize = this->array->getSquaresSize();
    std::vector<EventAnimation> events;
    EventAnimation event;

    for (int i = 0; i <= chosenNode; ++i) {
        if (i == size) break;

        event = EventAnimation();
        event.eventSquares.assign(squaresSize, EventSquare());
        for (int j = 0; j < size; ++j) {
            event.eventSquares[j].status = Square::Status::active;
            if (j == size - 1)
                event.eventSquares[size - 1].title = "n";
        }
        event.eventSquares[i].status = Square::Status::chosen;

        events.emplace_back(event);
    }

    if (chosenNode == size) {
        event = EventAnimation();
        event.eventSquares.assign(squaresSize, EventSquare());
        for (int j = 0; j < size; ++j) {
            event.eventSquares[j].status = Square::Status::active;
            if (j == size - 1)
```

```
                event.eventSquares[size - 1].title = "n";
        }

        events.emplace_back(event);
    }

    return events;
}
```

8.105 include/libScene/StaticArrayScene.hpp File Reference

```
#include "BaseScene.hpp"
#include "MenuArray.hpp"
#include "core/Array.hpp"
```

Classes

- class StaticArrayScene

8.106 StaticArrayScene.hpp

Go to the documentation of this file.

```
//
// Created by dirii on 27/03/2023.
//

#ifndef VISUALGO_CS162_STATICARRAYSCENE_HPP
#define VISUALGO_CS162_STATICARRAYSCENE_HPP

#include "BaseScene.hpp"
#include "MenuArray.hpp"
#include "core/Array.hpp"

class StaticArrayScene : public BaseScene{
private:
    MenuArray* menu;
    Array* array;

    void init();

public:
    explicit StaticArrayScene(sf::RenderWindow* window);

    void reset();

    void pollEvent(sf::Event event, sf::Vector2f mousePosView) override;
    void update() override;
    void render() override;

    std::vector<EventAnimation> addModeEvents(int chosenNode);
    std::vector<EventAnimation> deleteModeEvents(int chosenNode);
    std::vector<EventAnimation> updateModeEvents(int chosenNode);
    std::vector<EventAnimation> searchModeEvents(int chosenNode);
};

#endif //VISUALGO_CS162_STATICARRAYSCENE_HPP
```

8.107 include/MousePosition.cpp File Reference

```
#include "MousePosition.hpp"
```

8.108 MousePosition.cpp

Go to the documentation of this file.

```cpp
//
// Created by dirii on 23/03/2023.
//

#include "MousePosition.hpp"

void MousePosition::updateMousePosition() {
    this->mousePos = sf::Mouse::getPosition(*this->relativeWindow);
    this->mousePosView = this->relativeWindow->mapPixelToCoords(this->mousePos);
}
```

8.109 include/MousePosition.hpp File Reference

```cpp
#include <SFML/Graphics.hpp>
```

Classes

- class MousePosition

8.110 MousePosition.hpp

Go to the documentation of this file.

```cpp
//
// Created by dirii on 23/03/2023.
//

#ifndef VISUALGO_CS162_MOUSEPOSITION_HPP
#define VISUALGO_CS162_MOUSEPOSITION_HPP

#include <SFML/Graphics.hpp>

class MousePosition{
protected:
    sf::RenderWindow* relativeWindow;

    sf::Vector2i mousePos;
    sf::Vector2f mousePosView;
public:
    void updateMousePosition();
};

#endif //VISUALGO_CS162_MOUSEPOSITION_HPP
```

8.111 include/stuff/button.cpp File Reference

```cpp
#include "button.hpp"
#include <utility>
```

8.112 button.cpp

Go to the documentation of this file.

```
//
// Created by dirii on 25/03/2023.
//

#include "button.hpp"

#include <utility>

void Button::init() {
    this->isHover = this->isClick = false;

    this->button.setSize(this->size);
    this->button.setFillColor(this->color);
    this->button.setPosition(this->position);

    // set rounded corners
    this->button.setCornerPointCount(constants::CORNER_POINT_COUNT_BUTTON);
    this->button.setCornersRadius(constants::CORNER_RADIUS_BUTTON);


    this->text.setFont(this->font);
    this->text.setString(this->textString);
    this->text.setCharacterSize(this->textSize);
    this->text.setFillColor(this->textColor);
    this->text.setPosition(this->position.x + this->size.x / 2.0 - this->text.getGlobalBounds().width
    / 2.0,
                           this->position.y + this->size.y / 2.0 - this->text.getGlobalBounds().height
    / 1.1);
}

Button::Button(sf::RenderWindow *window, sf::Vector2f position, sf::Vector2f size,
               std::string textString, std::string changedTextString, int textSize,
               sf::Color textColor, sf::Color color, sf::Color hoverColor, sf::Color clickColor) {
    this->window = window;
    this->position = position;
    this->size = size;
    this->textString = std::move(textString);
    this->changedTextString = std::move(changedTextString);
    this->textSize = textSize;
    this->color = color;
    this->textColor = textColor;
    this->hoverColor = hoverColor;
    this->clickColor = clickColor;
    this->font.loadFromFile(constants::fontPath);

    this->init();
}

bool Button::pollEvent(sf::Vector2f mousePosView){
    bool hasClicked = false;

    if (this->isHover and this->isClick and !sf::Mouse::isButtonPressed(sf::Mouse::Left)) {
        hasClicked = true;
        std::swap(this->textString, this->changedTextString);
    }

    this->isHover = this->button.getGlobalBounds().contains(mousePosView);
    this->isClick = sf::Mouse::isButtonPressed(sf::Mouse::Left);

    return hasClicked;
}

void Button::update() {
    if (this->isHover) {
        this->button.setFillColor(this->hoverColor);

        if (this->isClick) {
            this->button.setFillColor(this->clickColor);
        }
    } else {
        this->button.setFillColor(this->color);
    }

    this->text.setString(this->textString);
}

void Button::render() {
    this->window->draw(this->button);
    this->window->draw(this->text);
}

bool Button::checkClicked() const {
    return this->isClick and this->isHover;
}

std::string Button::getTextString() const {
    return this->textString;
}

void Button::setColor(sf::Color _color) {
    this->color = _color;
}

Button::Button() {}

sf::Vector2f Button::getPosition() const {
    return this->position;
}

sf::Vector2f Button::getSize() const {
    return this->size;
}
```

8.113 include/stuff/button.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include "RoundedRectangleShape.hpp"
#include "Constants.hpp"
```

Classes

- class Button

8.114 button.hpp

Go to the documentation of this file.

```
//
// Created by dirii on 25/03/2023.
//

#ifndef VISUALGO_CS162_BUTTON_HPP
#define VISUALGO_CS162_BUTTON_HPP

#include <SFML/Graphics.hpp>
#include "RoundedRectangleShape.hpp"
#include "Constants.hpp"

class Button{
private:
    sf::RenderWindow* window;
    sf::RoundedRectangleShape button;
    sf::Text text;
    sf::Font font;
    sf::Color color;
    sf::Color textColor;
    sf::Color hoverColor;
    sf::Color clickColor;
    sf::Vector2f position;
    sf::Vector2f size;
    int textSize;
    std::string textString;
    std::string changedTextString;
    bool isHover;
    bool isClick;

    void init();

public:
    Button();
    Button(
            sf::RenderWindow* window,
            sf::Vector2f position,
            sf::Vector2f size,
            std::string textString,
            std::string changedTextString,
            int textSize,
            sf::Color textColor,
            sf::Color color,
            sf::Color hoverColor,
            sf::Color clickColor
            );

    bool pollEvent(sf::Vector2f mousePosView);
    void update();
    void render();

    void setColor(sf::Color _color);
    std::string getTextString() const;
    sf::Vector2f getPosition() const;
    sf::Vector2f getSize() const;

    bool checkClicked() const;
};

#endif //VISUALGO_CS162_BUTTON_HPP
```

8.115 include/stuff/CustomTextbox.cpp File Reference

`#include "CustomTextbox.hpp"`

8.116 CustomTextbox.cpp

Go to the documentation of this file.

```
//
// Created by dirii on 04/04/2023.
//

#include "CustomTextbox.hpp"

CustomTextbox::CustomTextbox(sf::RenderWindow *window, sf::Vector2f position, int size,
                             std::string _titleString, int maxLength) {
    this->window = window;
    this->titleString = std::move(_titleString);
    this->position = position;

    font.loadFromFile(constants::fontPath);
    this->title.setFont(font);
    this->title.setCharacterSize(size);
    this->title.setFillColor(sf::Color::Black);
    this->title.setString(this->titleString);
    this->title.setPosition(this->position);

    float width = this->title.findCharacterPos(this->titleString.size() - 1).x -
this->title.findCharacterPos(0).x;

//    std::cout « width « ' ' « this->title.getString().getSize() « std::endl;

    this->maxLength = maxLength;

    this->textbox = new TextBox(
        this->window,
        sf::Vector2f(this->position.x + width + 10, this->position.y),
        20,
        sf::Color::Black,
        sf::Color::White,
        this->maxLength
        );

    this->goButton = new Button(
        this->window,
        sf::Vector2f(this->textbox->getBox().getPosition().x + this->textbox->getBox().getSize().x +
10, this->position.y),
        constants::goButtonSize,
        "Go",
        "Go",
        20,
        sf::Color::Black,
        constants::normalGray,
        constants::hoverGray,
        constants::clickGray
        );

    this->isGoButtonClicked = false;
}

void CustomTextbox::pollEvent(sf::Event event, sf::Vector2f mousePosView) {
    this->textbox->pollEvent(event);
    if (this->goButton->pollEvent(mousePosView)) {
        this->isGoButtonClicked = true;
//          std::cout « "Go button clicked!\n";
    }
}

void CustomTextbox::update() {
    this->textbox->update();
    this->goButton->update();
}

void CustomTextbox::render() {
    this->window->draw(this->title);
    this->textbox->render();
    this->goButton->render();
}

std::string CustomTextbox::getTextString(){
    if (this->isGoButtonClicked) {
        this->isGoButtonClicked = false;
        return this->textbox->getTextString();
    }
    return "None";
}

void CustomTextbox::resetInput() {
    this->textbox->resetInput();
}
```

8.117 include/stuff/CustomTextbox.hpp File Reference

```
#include "Textbox.hpp"
#include "button.hpp"
```

Classes

- class CustomTextbox

8.118 CustomTextbox.hpp

Go to the documentation of this file.

```
//
// Created by dirii on 04/04/2023.
//

#ifndef VISUALGO_CS162_CUSTOMTEXTBOX_HPP
#define VISUALGO_CS162_CUSTOMTEXTBOX_HPP

#include "Textbox.hpp"
#include "button.hpp"

class CustomTextbox{
private:
    sf::RenderWindow* window;
    sf::Vector2f position;
    Button* goButton;
    TextBox* textbox;
    sf::Font font;
    sf::Text title;
    std::string titleString;
    int maxLength;
    bool isGoButtonClicked;

public:
    CustomTextbox(sf::RenderWindow* window, sf::Vector2f position, int size, std::string titleString,
int maxLength);
    ~CustomTextbox() = default;

    void pollEvent(sf::Event event, sf::Vector2f mousePosView);
    void update();
    void render();

    std::string getTextString();
    void resetInput();
};

#endif //VISUALGO_CS162_CUSTOMTEXTBOX_HPP
```

8.119 include/stuff/RoundedRectangleShape.cpp File Reference

```
#include "RoundedRectangleShape.hpp"
#include <cmath>
```

Namespaces

- namespace sf

8.120 RoundedRectangleShape.cpp

Go to the documentation of this file.

```

//
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.
//
// Permission is granted to anyone to use this software for any purpose,
// including commercial applications, and to alter it and redistribute it freely,
// subject to the following restrictions:
//
// 1. The origin of this software must not be misrepresented;
// you must not claim that you wrote the original software.
// If you use this software in a product, an acknowledgment
// in the product documentation would be appreciated but is not required.
//
// 2. Altered source versions must be plainly marked as such,
// and must not be misrepresented as being the original software.
//
// 3. This notice may not be removed or altered from any source distribution.
//

// Headers
#include "RoundedRectangleShape.hpp"
#include <cmath>

namespace sf
{
    RoundedRectangleShape::RoundedRectangleShape(const Vector2f& size, float radius, unsigned int
cornerPointCount)
    {
        mySize = size;
        myRadius = radius;
        myCornerPointCount = cornerPointCount;
        update();
    }

    void RoundedRectangleShape::setSize(const Vector2f& size)
    {
        mySize = size;
        update();
    }

    const Vector2f& RoundedRectangleShape::getSize() const
    {
        return mySize;
    }

    void RoundedRectangleShape::setCornersRadius(float radius)
    {
        myRadius = radius;
        update();
    }

    float RoundedRectangleShape::getCornersRadius() const
    {
        return myRadius;
    }

    void RoundedRectangleShape::setCornerPointCount(unsigned int count)
    {
        myCornerPointCount = count;
        update();
    }

    std::size_t RoundedRectangleShape::getPointCount() const
    {
        return myCornerPointCount*4;
    }

    sf::Vector2f RoundedRectangleShape::getPoint(std::size_t index) const
    {
        if(index >= myCornerPointCount*4)
            return sf::Vector2f(0,0);

        float deltaAngle = 90.0f/(myCornerPointCount-1);
        sf::Vector2f center;
        unsigned int centerIndex = index/myCornerPointCount;
        static const float pi = 3.141592654f;

        switch(centerIndex)
        {
            case 0: center.x = mySize.x - myRadius; center.y = myRadius; break;
            case 1: center.x = myRadius; center.y = myRadius; break;
            case 2: center.x = myRadius; center.y = mySize.y - myRadius; break;
            case 3: center.x = mySize.x - myRadius; center.y = mySize.y - myRadius; break;
        }

        return sf::Vector2f(myRadius*cos(deltaAngle*(index-centerIndex)*pi/180)+center.x,
                            -myRadius*sin(deltaAngle*(index-centerIndex)*pi/180)+center.y);
    }
} // namespace sf
```

8.121 include/stuff/RoundedRectangleShape.hpp File Reference

```
#include <SFML/Graphics/Shape.hpp>
```

Classes

- class sf::RoundedRectangleShape

  Specialized shape representing a rectangle with rounded corners.

Namespaces

- namespace sf

8.122 RoundedRectangleShape.hpp

Go to the documentation of this file.

```

//
// This software is provided 'as-is', without any express or implied warranty.
// In no event will the authors be held liable for any damages arising from the use of this software.
//
// Permission is granted to anyone to use this software for any purpose,
// including commercial applications, and to alter it and redistribute it freely,
// subject to the following restrictions:
//
// 1. The origin of this software must not be misrepresented;
// you must not claim that you wrote the original software.
// If you use this software in a product, an acknowledgment
// in the product documentation would be appreciated but is not required.
//
// 2. Altered source versions must be plainly marked as such,
// and must not be misrepresented as being the original software.
//
// 3. This notice may not be removed or altered from any source distribution.
//

#ifndef ROUNDEDRECTANGLESHAPE_HPP
#define ROUNDEDRECTANGLESHAPE_HPP

// Headers
#include <SFML/Graphics/Shape.hpp>

namespace sf
{
    class RoundedRectangleShape : public sf::Shape
    {
    public:
        explicit RoundedRectangleShape(const Vector2f& size = Vector2f(0, 0), float radius = 0,
unsigned int cornerPointCount = 0);

        void setSize(const Vector2f& size);

        const Vector2f& getSize() const;

        void setCornersRadius(float radius);

        float getCornersRadius() const;

        void setCornerPointCount(unsigned int count);

        virtual std::size_t getPointCount() const;

        virtual sf::Vector2f getPoint(std::size_t index) const;

    private:
        // Member data
        Vector2f mySize;
        float myRadius;
        unsigned int myCornerPointCount;
    };
}
#endif // ROUNDEDRECTANGLESHAPE_HPP


```

8.123 include/stuff/Textbox.cpp File Reference

```
#include "Textbox.hpp"
```

8.124 Textbox.cpp

Go to the documentation of this file.

```
//
// Created by dirii on 01/04/2023.
```

```
//

#include "Textbox.hpp"

TextBox::TextBox(sf::RenderWindow* window, sf::Vector2f position, int size, sf::Color textColor,
                 sf::Color boxColor, int maxLength) {
    this->window = window;

    this->cursor = "|";

    this->box.setPosition(position);
    this->box.setSize(sf::Vector2f(static_cast<float>((maxLength + 1) * 12), static_cast<float>(size * 1.5)));
    this->box.setFillColor(boxColor);
    this->box.setOutlineColor(sf::Color::Black);
    this->box.setOutlineThickness(1);

    this->font.loadFromFile(constants::fontPath);
    this->text.setFont(this->font);
    this->text.setCharacterSize(size);
    this->text.setFillColor(textColor);
    this->text.setPosition(position);

    this->maxLength = maxLength;
    this->textColor = textColor;
    this->boxColor = boxColor;

    this->cursorVisible = true;
    this->flickerClock.restart();
}

void TextBox::pollEvent(sf::Event event) {
    if (event.type == sf::Event::TextEntered)
    {
        if (event.text.unicode == '\b')
        {
            if (!this->inputString.empty())
            {
                this->inputString.pop_back();
            }
        }
        else if (((48 <= event.text.unicode && event.text.unicode <= 57) || event.text.unicode == static_cast<int>(',')) && this->inputString.size() < this->maxLength)
        {
            this->inputString += static_cast<char>(event.text.unicode);
        }

        this->text.setString(this->inputString);
    }

//     if (event.type == sf::Event::Resized)
//     {
//         box.setPosition(
//                 static_cast<float>(this->window->getSize().x) / 2 - box.getSize().x / 2,
//                 static_cast<float>(this->window->getSize().y) / 2 - box.getSize().y / 2
//                 );
//         text.setPosition(box.getPosition().x + 10, box.getPosition().y);
//         cursor.setPosition(text.getGlobalBounds().width + text.getPosition().x, cursor.getPosition().y);
//     }
}

void TextBox::update() {
    if (this->flickerClock.getElapsedTime().asSeconds() >= 0.5)
    {
        this->cursorVisible = !this->cursorVisible;
        this->flickerClock.restart();
    }

    if (this->cursorVisible)
    {
        this->text.setString(this->inputString + this->cursor);
    }
    else
    {
        this->text.setString(this->inputString);
    }
}

void TextBox::render() {
    this->window->draw(this->box);
    this->window->draw(this->text);
}

std::string TextBox::getTextString() const {
    return this->inputString;
}
```

```
sf::RectangleShape TextBox::getBox() const {
    return this->box;
}

void TextBox::resetInput() {
    this->inputString = "";
    this->text.setString(this->inputString);
}
```

8.125 include/stuff/Textbox.hpp File Reference

```
#include "Constants.hpp"
#include <SFML/Graphics.hpp>
#include <iostream>
#include <string>
```

Classes

- class TextBox

8.126 Textbox.hpp

Go to the documentation of this file.

```
//
// Created by dirii on 01/04/2023.
//

#ifndef VISUALGO_CS162_TEXTBOX_HPP
#define VISUALGO_CS162_TEXTBOX_HPP

#include "Constants.hpp"
#include <SFML/Graphics.hpp>
#include <iostream>
#include <string>

class TextBox {
public:
    TextBox(sf::RenderWindow* window, sf::Vector2f position, int size, sf::Color textColor, sf::Color
boxColor, int maxLength);

    void pollEvent(sf::Event event);
    void update();
    void render();

    std::string getTextString() const;
    sf::RectangleShape getBox() const;
    void resetInput();

private:
    sf::RenderWindow* window;

    std::string cursor;
    sf::RectangleShape box;
    sf::Font font;
    sf::Text text;

    std::string inputString;

    int maxLength;
    sf::Color textColor;
    sf::Color boxColor;

    bool cursorVisible;
    sf::Clock flickerClock;
};

#endif //VISUALGO_CS162_TEXTBOX_HPP
```

8.127 include/stuff/ToStringWithPrecision.hpp File Reference

```
#include <sstream>
```

Functions

- template<typename T >
  std::string to_string_with_precision (const T a_value, const int n=2)

8.127.1 Function Documentation

8.127.1.1 to_string_with_precision()

```
template<typename T >
std::string to_string_with_precision (
            const T a_value,
            const int n = 2 )
```

Definition at line 11 of file ToStringWithPrecision.hpp.

```
00012 {
00013     std::ostringstream out;
00014     out.precision(n);
00015     out « std::fixed « a_value;
00016     return std::move(out).str();
00017 }
```

8.128 ToStringWithPrecision.hpp

Go to the documentation of this file.

```
00001 //
00002 // Created by dirii on 14/04/2023.
00003 //
00004 
00005 #ifndef VISUALGO_CS162_TOSTRINGWITHPRECISION_HPP
00006 #define VISUALGO_CS162_TOSTRINGWITHPRECISION_HPP
00007 
00008 #include <sstream>
00009 
00010 template <typename T>
00011 std::string to_string_with_precision(const T a_value, const int n = 2)
00012 {
00013     std::ostringstream out;
00014     out.precision(n);
00015     out « std::fixed « a_value;
00016     return std::move(out).str();
00017 }
00018 
00019 #endif //VISUALGO_CS162_TOSTRINGWITHPRECISION_HPP
```

8.129 include/Window.cpp File Reference

```
#include "Window.hpp"
```

8.130 Window.cpp

Go to the documentation of this file.

```
//
// Created by dirii on 23/03/2023.
//

#include "Window.hpp"

void Window::init() {
    this->relativeWindow = this->window;
    this->currentScene = constants::sceneVariables::MAIN_MENU_SCENE;

    // init buttons
    this->submenuButton = new Button(
            this->window,
            constants::submenuButtonPos,
            constants::sideButtonSize,
            ">",
            "<",
            15,
            sf::Color::Black,
            constants::normalGray,
            constants::hoverGray,
            constants::clickGray
            );

    this->demoCodeButton = new Button(
            this->window,
            constants::demoCodeButtonPos,
            constants::sideButtonSize,
            "<",
            ">",
            15,
            sf::Color::Black,
            constants::normalGray,
            constants::hoverGray,
            constants::clickGray
            );
}

void Window::initWindow() {
    this->videoMode.width = constants::Width;
    this->videoMode.height = constants::Height;
    this->window = new sf::RenderWindow(
            this->videoMode,
            constants::titleWindow,
            sf::Style::Titlebar | sf::Style::Close);

    this->window->setFramerateLimit(constants::fps);
}

Window::Window() {
    this->initWindow();
    this->initScenes();
    this->init();
}

const bool Window::running() const {
    return this->window->isOpen();
}

void Window::pollEvent() {
    // event polling
    while (this->window->pollEvent(this->event)) {
        switch (this->event.type) {
            case sf::Event::Closed:
                this->window->close();
                break;
            case sf::Event::KeyPressed:
                if (this->event.key.code == sf::Keyboard::Q) {
                    std::cout « "You have pressed Q!\n";
                }
                if (this->event.key.code == sf::Keyboard::W) {
                    std::cout « "You have pressed W!\n";
                }
                break;
            default:
                break;
        }

        if (this->submenuButton->pollEvent(this->mousePosView)) {
            std::cout « "You have clicked on submenu button!\n";
            this->scenes[this->currentScene]->isMenuOpen = (this->submenuButton->getTextString() ==
"<");
```

```
        }

        if (this->demoCodeButton->pollEvent(this->mousePosView)) {
            std::cout « "You have clicked on demo code button!\n";
            this->scenes[this->currentScene]->isDemoCodeOpen = (this->demoCodeButton->getTextString() == ">");
        }

        for (int i = 1; i < constants::sceneVariables::SCENE_COUNT; i++) {
            if (this->scenes[i]->modeButton->pollEvent(this->mousePosView)) {
                std::cout « "You have clicked on " « constants::sceneVariables::SCENE_NAMES[i] « " scene!\n";
                this->currentScene = static_cast<constants::sceneVariables::Scene>(i);
                this->scenes[this->currentScene]->isMenuOpen = (this->submenuButton->getTextString() == "<");
                this->scenes[this->currentScene]->isDemoCodeOpen = (this->demoCodeButton->getTextString() == ">");
            }
        }

        this->scenes[this->currentScene]->pollEvent(this->event, this->mousePosView);
    }
}

void Window::update() {
    this->scenes[this->currentScene]->modeButton->setColor(constants::normalGray);

    this->updateMousePosition();
    this->pollEvent();

    this->submenuButton->update();
    this->demoCodeButton->update();
    this->scenes[this->currentScene]->modeButton->setColor(constants::hoverGreen);

    for (int i = 1; i < constants::sceneVariables::SCENE_COUNT; i++) {
        this->scenes[i]->modeButton->update();
    }

    this->scenes[this->currentScene]->update();
}

void Window::render() {
    /*
 * clear old frames
 * create objects
 * display it
 */

    this->window->clear(sf::Color::White);

    // drawing game
    this->submenuButton->render();
    this->demoCodeButton->render();
    for (int i = 1; i < constants::sceneVariables::SCENE_COUNT; i++) {
        this->scenes[i]->modeButton->render();
    }

    this->scenes[this->currentScene]->render();

    this->window->display();
}

void Window::initScenes() {
    this->scenes[constants::sceneVariables::MAIN_MENU_SCENE] = new MainMenu(this->window);
    this->scenes[constants::sceneVariables::SINGLY_LINKED_LIST_SCENE] = new SLLScene(this->window);
    this->scenes[constants::sceneVariables::DOUBLY_LINKED_LIST_SCENE] = new DLLScene(this->window);
    this->scenes[constants::sceneVariables::CIRCULAR_LINKED_LIST_SCENE] = new CLLScene(this->window);
    this->scenes[constants::sceneVariables::STACK_SCENE] = new StackScene(this->window);
    this->scenes[constants::sceneVariables::QUEUE_SCENE] = new QueueScene(this->window);
    this->scenes[constants::sceneVariables::STATIC_ARRAY_SCENE] = new StaticArrayScene(this->window);
    this->scenes[constants::sceneVariables::DYNAMIC_ARRAY_SCENE] = new DynamicArrayScene(this->window);

    for (int i = 1; i < constants::sceneVariables::SCENE_COUNT; i++) {
        this->scenes[i]->createModeButton(
                sf::Vector2f(
                        constants::modeButtonPos.x * static_cast<float>(i) +
                            (constants::distance2ModeButtons + constants::modeButtonSize.x) * static_cast<float>(i - 1),
                        constants::modeButtonPos.y
                        ),
                        constants::sceneVariables::NAME_MODE_BUTTON[i]
                );
    }
}
```

8.131 include/Window.hpp File Reference

```
#include <iostream>
#include <SFML/Graphics.hpp>
#include "MousePosition.hpp"
#include "Constants.hpp"
#include "stuff/button.hpp"
#include "libScene/AllScenes.hpp"
```

Classes

- class Window

8.132 Window.hpp

Go to the documentation of this file.

```
//
// Created by dirii on 23/03/2023.
//

#ifndef VISUALGO_CS162_WINDOW_HPP
#define VISUALGO_CS162_WINDOW_HPP

#include <iostream>
#include <SFML/Graphics.hpp>
#include "MousePosition.hpp"
#include "Constants.hpp"
#include "stuff/button.hpp"
#include "libScene/AllScenes.hpp"

class Window : public MousePosition{
private:
    sf::RenderWindow* window{};
    sf::VideoMode videoMode;
    sf::Event event{};

    // scenes
    class BaseScene* scenes[constants::sceneVariables::SCENE_COUNT];
    constants::sceneVariables::Scene currentScene;

    // buttons
    Button* submenuButton,
            *demoCodeButton;

    void initWindow();
    void initScenes();
    void init();

public:
    Window();
    ~Window() = default;

    const bool running() const;

    void pollEvent();
    void update();
    void render();
};

#endif //VISUALGO_CS162_WINDOW_HPP
```

8.133 main.cpp File Reference

```
#include "Window.hpp"
```

Functions

- int main ()

8.133.1 Function Documentation

8.133.1.1 main()

```
int main ( )
```

Definition at line 3 of file main.cpp.

```
            {
    Window window;

    while (window.running()) {
        window.update();

        window.render();
    }

    return 0;
}
```

8.134 main.cpp

Go to the documentation of this file.

```
#include "Window.hpp"

int main() {
    Window window;

    while (window.running()) {
        window.update();

        window.render();
    }

    return 0;
}
```

8.135 README.md File Reference

Index